41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2021, December 15–17, 2021, Virtual Conference

Edited by

Mikołaj Bojańczyk
Chandra Chekuri

LIPIcs

Editors

Mikołaj Bojańczyk
University of Warsaw, Poland
bojan@mimuw.edu.pl

Chandra Chekuri
University of Illinois, Urbana-Champaign, IL, US
chekuri@illinois.edu

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

ISBN 978-3-95977-215-0

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/978-3-95977-215-0.

Publication date
December, 2021

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

Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2021.0

ISBN 978-3-95977-215-0 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021). The conference was originally planned to be held on December 15–17, 2021 in BITS Pilani, K K Birla Goa Campus, Goa, India. Due to the COVID-19 pandemic, the conference was moved to a virtual format, with the same dates.

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 (Chandra Chekuri for Track A and Mikołaj Bojańczyk 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 to all.

The conference comprises of 5 invited talks, 27 contributed talks in Track A, and 20 in Track B. This volume contains all the contributed papers from the two tracks, and the abstracts of all the invited talks. The conference received a total of 122 submissions with 73 in Track A and 49 in Track B. This edition of FSTTCS implemented, for the first time, an author rebuttal phase during the paper review and selection process. We thank all the authors who submitted their papers to FSTTCS 2021. 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: Scott Aaronson (University of Texas at Austin), Javier Esparza (Technische Universität München), Leslie Ann Goldberg (University of Oxford), Huijia (Rachel) Lin (University of Washington), and Rahul Savani (University of Liverpool).

The main conference is to be accompanied by four workshops: *iVerif: Artificial Intelligence and Verification* (organized by Shibashis Guha and Guillermo A. Perez), *QISE: Quantum Information Science and Engineering* (organized by Jaikumar Radhakrishnan and Manoj Nambiar), *Trends in Transformations* (organized by Emmanuel Filiot and S. Krishna), and *VeriCrypt* (organized by Karthikeyan Bhargavan and Aseem Rastogi).

We are indebted to the organising committee members: A. Baskar (BITS Pilani), Pritam Bhattacharya (BITS Pilani), Amaldev Manuel (IIT Goa), and A.V. Sreejith (IIT Goa) for managing the logistics of the conference and affiliated workshops. They made all the necessary arrangements for the virtual conference, and they are doing this for a second year in row due to the Covid pandemic. We thank S.P. Suresh (CMI, Chennai) for maintaining the conference web page and promptly addressing our update requests. We thank the friendly staff at Dagstuhl LIPIcs, Michael Didas and Michael Wagner, for helping us put together the proceedings. Finally, we thank the members of the Steering Committee, especially Jaikumar Radhakrishnan, and the PC Chairs from FSTTCS 2020 (Nitin Saxena and Sunil Simon), for providing pertinent information and advice about various aspects of the conference.

Mikołaj Bojańczyk and Chandra Chekuri
November 2021

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Track A

- Amey Bhangale (University of California, Riverside)
- Chandra Chekuri (University of Illinois, Urbana-Champaign) — co-chair
- Ashish Chiplunkar (Indian Institute of Technology, Delhi)
- Keerti Choudhary (Indian Institute of Technology, Delhi)
- Omar Fawzi (INRIA, Lyon)
- Uriel Feige (Weizmann Institute of Science)
- Anna Gál (University of Texas, Austin)
- Sushmita Gupta (Institute of Mathematical Sciences, Chennai)
- Valentine Kabanets (Simon Fraser University)
- Sanjeev Khanna (University of Pennsylvania)
- Sudeshna Kolay (Indian Institute of Technology, Kharagpur)
- Ravishankar Krishnaswamy (Microsoft Research India)
- Kamesh Munagala (Duke University)
- Sriram Pemmaraju (University of Iowa)
- Rahul Saladi (Indian Institute of Science, Bengaluru)
- Swagato Sanyal (Indian Institute of Technology, Kharagpur)

Track B

- S Akshay (IIT Bombay)
- Mikołaj Bojańczyk (University of Warsaw) — co-chair
- Dmitry Chistikov (University of Warwick)
- Thomas Colcombet (IRIF, Paris)
- Anuj Dawar (University of Cambridge)
- Manfred Droste (University of Leipzig)
- Barbara König (University of Duisburg-Essen)
- Rupak Majumdar (MPI-SWS)
- Filip Mazowiecki (MPI-SWS)
- Andrzej Murawski (University of Oxford)
- Joanna Ochremiak (CNRS, Bordeaux)
- M Praveen (Chennai Mathematical Institute)
- Karin Quaas (University of Leipzig)
- Ocan Sankur (IRISA, Rennes)
- Helmut Seidl (Technical University Munich)
- Georg Zetzsche (MPI-SWS)

List of External Reviewers: Track A

Akanksha Agrawal
Amer Mouawad
Anastasios Sidiropoulos
Arijit Ghosh
Aritra Banik
Ashutosh Gupta
Bhaswar Bhattacharya
Chaitanya Swamy
Daniel Lokshtanov
Daniel Stefankovic
Denis Pankratov
Devvrit K
Diptarka Chakraborty
Dishant Goyal
Edin Husic
Eduard Eiben
Fahad Panolan
Gil Cohen
Gopinath Mishra
Gramoz Goranci
Huan Li
Janani Sundaresan
Jannik Peters
Jayadev Acharya
Jie Xue
Joeseph Mitchell
Josh Alman
Kasturi Varadarajan
Katarina Cechlarova
Lawqueen Kanesh
Lenwood Heath
Meirav Zehavi
Michael Lampis
Michal Wlodarczyk
Mrinal Kumar
Nikhil Balaji
Nikhil Mande
Nithin Varma
Oliver Kullmann
Palash Dey
Pallavi Jain
Pankaj Agarwal
Pingan Cheng
Pooja Kulkarni
Pradeesha Ashok
Prajakta Nimbhorkar
Pranabendu Misra
Pratibha Choudhary
Ramprasad Saptharishi
Rohit Gurjar
Rohit Vaish
Roohani Sharma
Sagar Kale
Sahil Singla
Sai Sandeep
Sándor Kisfaludi-Bak
Sanjukta Roy
Sathish Govindarajan
Satyadev Nandakumar
Sayan Bandyapadhyay
Sayantan Chakraborty
Sepehr Assadi
Shahbaz Khan
Shahin Kamali
Shreyas Pai
Soumen Maity
Sourya Roy
Sujata Ghosh
Sundar Vishwanathan
Syamantak Das
Tanmay Inamdar
Tatiana Starikovskaya
Tom van der Zanden
Varun Gupta
Vibha Sahlot
Vijaykrishna Gurunathan
Waldo Gálvez
Xinhang Lu
Yanyi Liu
Yassine Hamoudi

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of External Reviewers: Track B

Mohamed Faouzi Atig
Pascal Baumann
Damien Busatto-Gaston
Olivier Carton
Antonio Casares
Frank Drewes
Petter Ericson
Nathanaël Fijalkow
Marie Fortin
Giovanna Guaiana
Shibashis Guha
Christoph Haase
Willem Heijltjes
Loic Helouet
Naohiko Hoshino
Rasmus Ibsen-Jensen
Petr Jancar
Arthur Jaquard
Kohei Kishida
Dietrich Kuske
Florin Manea
Tomas Masopust
Karla Messing
Richard Mörbitz
Torsten Mütze
Masaki Nakamura
Naoki Nishida
Vincent Penelle
Guillermo Perez
Gabriele Puppis
Ritam Raha
R. Ramanujam
Alexander Rubtsov
Arnaud Sangnier
Markus L. Schmid
Lia Schütze
Anastasia Sofronova
S P Suresh
Lidia Tendera
K. S. Thejaswini
Ramanathan Thinniyam Srinivasan
Marie Van Den Bogaard
Gerco van Heerdt
Dominic Verdon
Fabio Zanasi
Damien Zufferey

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

BQP After 28 Years

Scott Aaronson ✉ 🏠
University of Texas, Austin, TX, USA

Abstract

I will discuss the now-ancient question of where BQP, Bounded-Error Quantum Polynomial-Time, fits in among classical complexity classes. After reviewing some basics from the 90s, I will discuss the Forrelation problem that I introduced in 2009 to yield an oracle separation between BQP and PH, and the dramatic completion of that program by Ran Raz and Avishay Tal in 2018. I will then discuss very recent work, with William Kretschmer and DeVon Ingram, which leverages the Raz-Tal theorem, along with a new "quantum-aware" random restriction method, to obtain results that illustrate just how differently BQP can behave from BPP. These include oracles relative to which $\mathsf{NP}^{\mathsf{BQP}} \not\subset \mathsf{BQP}^{\mathsf{PH}}$ – solving a 2005 open problem of Lance Fortnow – and conversely, relative to which $\mathsf{BQP}^{\mathsf{NP}} \not\subset \mathsf{PH}^{\mathsf{BQP}}$; an oracle relative to which $\mathsf{P} = \mathsf{NP}$ and yet $\mathsf{BQP} \neq \mathsf{QCMA}$; an oracle relative to which $\mathsf{NP} \subseteq \mathsf{BQP}$ yet PH is infinite; an oracle relative to which $\mathsf{P} = \mathsf{NP} \neq \mathsf{BQP} = \mathsf{PP}$; and an oracle relative to which $\mathsf{PP} = \mathsf{PostBQP} \not\subset \mathsf{QMA}^{\mathsf{QMA}^{\cdots}}$. By popular demand, I will also speculate about the status of BQP in the unrelativized world.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory

Keywords and phrases quantum computing, complexity theory, oracle separations, circuit lower bounds

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.1

Category Invited Talk

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 1; pp. 1:1–1:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

State Complexity of Population Protocols

Javier Esparza
Technische Universität München, Germany

Abstract

Population protocols were introduced by Angluin et al. in 2004 to study the theoretical properties of networks of mobile sensors with very limited computational resources. They have also been proposed as a natural computing model, with molecules, cells, or microorganisms playing the role of sensors.

In a population protocol an arbitrary number of indistinguishable, finite-state agents interact randomly in pairs to collectively decide if their initial global configuration satisfies a given property. The property is formalized as a predicate that maps each initial configuration to an output, 0 or 1. Starting from an initial configuration, the agents eventually agree to the correct output almost surely, and continue producing it forever. The protocol is said to *stabilize* to the correct output.

It is well known that population protocols can decide exactly the semilinear predicates, or, equivalently, the predicates expressible in Presburger arithmetic. Current research concentrates on investigating the amount of resources needed to decide a given predicate. The standard resources, time and memory, translate for population protocols into expected time to stabilization, usually called *parallel runtime*, and number of states of each agent. In this talk we concentrate on the latter.

A variant of population protocols allows for a *leader*, a distinguished finite-state agent that is added to the initial configuration and, intuitively, helps the other agents to organize the computation. In the last years my collaborators and I have obtained upper and lower bounds for the state complexity of population protocols with and without a leader. Define the *state complexity* of a predicate as the minimal number of states of a protocol that decides the predicate, and $STATE(\eta)$ as the maximum state complexity of the predicates of size at most η, where predicates are encoded as quantifier-free formulas of Presburger arithmetic with coefficients written in binary. Using techniques from the theory of Petri nets and Vector Addition Systems, we have shown that $STATE(\eta)$ is polynomially bounded, even for leaderless protocols; this improves on the exponential bound given in 2004 by Angluin and collaborators. We have also proved that $STATE(\eta) \in \Omega(\log\log\eta)$ for leaderless protocols, even for those deciding very simple predicates of the form $x \geq c$ for some constant c. In the talk I report on these results, and on two very recent, still unpublished results. Modulo the pending peer-review confirmation, the first result shows the existence of leaderless protocols with a polynomial number of states *and* linear parallel runtime, and the second, due to Leroux, gives a $\Omega((\log\log\eta)^{1/3})$ lower bound for protocols with a leader.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Automata over infinite objects

Keywords and phrases Population protocols, state complexity, Petri nets

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.2

Category Invited Talk

Funding *Javier Esparza*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS).

Acknowledgements I want to thank my collaborators Michael Blondin, Philipp Czerner, Blaise Genest, Roland Guttenberg, Martin Helfrich, and Stefan Jaax.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 2; pp. 2:1–2:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Approximately Counting Graph Homomorphisms and Retractions

Leslie Ann Goldberg
University of Oxford, UK

Abstract

A homomorphism from a graph G to a graph H is a function from the vertices of G to the vertices of H that preserves the edges of G in the sense that every edge of G is mapped to an edge of H. By changing the target graph H, we can capture interesting structures in G. For example, homomorphisms from G to a k-clique H correspond to the proper k-colourings of G. There has been a lot of algorithmic work on the problem of (approximately) counting homomorphisms. The goal is to figure out for which graphs H the problem of approximately counting homomorphisms to H is algorithmically feasible. This talk will survey what is known. Despite much work, there are still plenty of open problems. We will discuss the problem of approximately counting *list homomorphisms* (where the input specifies, for each vertex of G, the list of vertices of H to which it can be mapped). Because the lists add extra expressibility, it is easier to prove that counting homomorphisms to a particular graph H is intractable. In fact, we have a full trichotomy (joint work with Galanis and Jerrum, 2017). Here, the complexity of homomorphism-counting is related to certain hereditary graph classes. The trichotomy will be explained in the talk – no prior knowledge of the area will be assumed. In more recent work, with Focke and Živný, we have investigated the complexity of counting *retractions* to H – this problem falls between homomorphism-counting and list-homomorphism counting. Here we have only a partial classification, which applies to all square-free graphs H. So again, there are plenty of open problems.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Graph homomorphisms, counting

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.3

Category Invited Talk

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 3; pp. 3:1–3:1
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Indistinguishability Obfuscation from Well-Founded Assumptions

Huijia (Rachel) Lin
Paul G. Allen School of Computer Science & Engineering,
University of Washington, Seattle, WA, USA

Abstract

Indistinguishability obfuscation, introduced by Barak et al. [Crypto 2001], aims to compile programs into unintelligible ones while preserving functionality. It is a fascinating and powerful object that has been shown to enable a host of new cryptographic goals and beyond. However, constructions of indistinguishability obfuscation have remained elusive, with all other proposals relying on heuristics or newly conjectured hardness assumptions. In this work, we show how to construct indistinguishability obfuscation from the subexponential hardness of three well-founded assumptions. We prove the following.

▶ **Theorem 1** (Informal). *Assume sub-exponential hardness for the following:*

- *the Learning Parity with Noise (*LPN*) assumption over general prime fields $\mathbb{F}_p$ with polynomially many* LPN *samples and error rate $1/k^\delta$, where k is the dimension of the* LPN *secret, and $\delta > 0$ is any constant;*
- *the existence of a Boolean Pseudo-Random Generator (*PRG*) in* NC^0 *with stretch $n^{1+\tau}$, where n is the length of the* PRG *seed, and $\tau > 0$ is any constant;*
- *the Decision Linear (*DLIN*) assumption on symmetric bilinear groups of prime order.*

Then, (subexponentially secure) indistinguishability obfuscation for all polynomial-size circuits exist.

As a corollary, all cryptographic goals that can be achieved using indistinguishability obfuscation can now be achieved assuming the above three assumptions. This includes fully homomorphic encryption, functional encryption, multiparty non-interactive key-exchange, succinct garbled random access machine, and many others.

This is joint work with Aayush Jain (UCLA and NTT Research) and Amit Sahai (UCLA).

2012 ACM Subject Classification Theory of computation → Cryptographic primitives

Keywords and phrases Cryptography, indistinguishability obfuscation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.4

Category Invited Talk

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 4; pp. 4:1–4:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Complexity of Gradient Descent

Rahul Savani
Department of Computer Science, University of Liverpool, UK

Abstract

PPAD and PLS are successful classes that capture the complexity of important game-theoretic problems. For example, finding a mixed Nash equilibrium in a bimatrix game is PPAD-complete, and finding a pure Nash equilibrium in a congestion game is PLS-complete. Many important problems, such as solving a Simple Stochastic Game or finding a mixed Nash equilibrium of a congestion game, lie in both classes. It was strongly believed that their intersection, PPAD ∩ PLS, does not have natural complete problems. We show that it does: any problem that lies in both classes can be reduced in polynomial time to the problem of finding a stationary point of a continuously differentiable function on the domain $[0,1]^2$. Thus, as PPAD captures problems that can be solved by Lemke-Howson type complementary pivoting algorithms, and PLS captures problems that can be solved by local search, we show that PPAD ∩ PLS exactly captures problems that can be solved by Gradient Descent.

This is joint work with John Fearnley, Paul Goldberg, and Alexandros Hollender. It appeared at STOC'21, where it was given a Best Paper Award [4].

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Mathematical optimization; Mathematics of computing → Continuous functions

Keywords and phrases Computational Complexity, Continuous Optimization, TFNP, PPAD, PLS, CLS, UEOPL

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.5

Category Invited Talk

Related Version *Full Version*: https://doi.org/10.1145/3406325.3451052

1 Talk summary

This talk is about the computational complexity of Gradient Descent, one of the oldest and most widely-used algorithmic approaches to doing optimisation. The approach dates all the way back to an 1847 paper of Cauchy.

When Gradient Descent is constrained to a bounded domain, there are not one but two reasons why it must terminate at an approximate stationary point or boundary point where the gradient is trying to take it outside the domain:

- We are always going downhill, altitude must "bottom out". This puts the search for a solution in the complexity class PLS (polynomial local search).
- Gradient Descent maps any point to a nearby point in the direction of the negative gradient. Brouwer's Fixed Point Theorem guarantees that such a mapping has a point mapped to itself. This puts the search for a solution in the complexity class PPAD.

PPAD and PLS correspond to existence-of-solution proof principles that guarantee solutions, but in a computationally-inefficient way. Both classes have become successful through the fact that they have been shown to exactly characterise the complexity of important problems. Our main result shows that the Gradient Descent solution-existence principle tastefully combines the PLS principle with the PPAD principle:

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 5; pp. 5:1–5:2
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

> We show how to efficiently reduce any problem that is in both PPAD and PLS to the problem of finding a stationary point of a continuously differentiable function from $[0,1]^2$ to $[0,1]$.

This is the first natural problem to be shown complete for PPAD ∩ PLS. Our results also imply that the class CLS (Continuous Local Search) [2] – which was defined by Daskalakis and Papadimitriou as a more "natural" counterpart to PPAD ∩ PLS and contains many interesting problems – is itself equal to PPAD ∩ PLS.

Our result has been used to show that computing a mixed equilibrium of a congestion game is also complete for PPAD ∩ PLS [1], and, as we discuss in [4], it opens up the possibility of PPAD ∩ PLS hardness for other important problems, such as finding Tarski fixed points [3, 6] or finding solutions that are guaranteed to exist by the Colorful Carathéodory theorem [7]. Several of the other problems in the original CLS paper [2], such as the P-matrix Linear Complementarity Problem and finding the fixed point of (piecewise linear) Contraction map, have unique solutions. For these problems, we believe that another class, called UEOPL, for Unique End of Potential Line, is more likely than PPAD ∩ PLS to be the correct class to capture their complexity [5].

References

1 Yakov Babichenko and Aviad Rubinstein. Settling the Complexity of Nash Equilibrium in Congestion Games. In *Proc. of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1426–1437, 2021.

2 Constantinos Daskalakis and Christos H. Papadimitriou. Continuous Local Search. In *Proc. of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 790–804, 2011.

3 Kousha Etessami, Christos H. Papadimitriou, Aviad Rubinstein, and Mihalis Yannakakis. Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria. In *Proc. of the 11th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 18:1–18:19, 2020.

4 John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. The Complexity of Gradient Descent: CLS = PPAD ∩ PLS. In *Proc. of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 46–59, 2021.

5 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique End of Potential Line. *Journal of Computer and System Sciences*, 114:1–35, 2020.

6 John Fearnley and Rahul Savani. A Faster Algorithm for Finding Tarski Fixed Points. In *Proc. of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 29:1–29:16, 2021.

7 Frédéric Meunier, Wolfgang Mulzer, Pauline Sarrabezolles, and Yannik Stein. The Rainbow at the End of the Line – A PPAD Formulation of the Colorful Carathéodory Theorem with applications. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1342–1351, 2017.

Scheduling in the Secretary Model

Susanne Albers
Department of Computer Science, Technische Universität München, Germany

Maximilian Janke
Department of Computer Science, Technische Universität München, Germany

Abstract

This paper studies online makespan minimization in the secretary model. Jobs, specified by their processing times, are presented in a uniformly random order. The input size n is known in advance. An online algorithm has to non-preemptively assign each job permanently and irrevocably to one of m parallel and identical machines such that the expected time it takes to process them all, the makespan, is minimized.

We give two deterministic algorithms. First, a straightforward adaptation of the semi-online strategy LightLoad [4] provides a very simple approach retaining its competitive ratio of 1.75. A new and sophisticated algorithm is 1.535-competitive. These competitive ratios are not only obtained in expectation but, in fact, for all but a very tiny fraction of job orders.

Classically, online makespan minimization only considers the worst-case order. Here, no competitive ratio below 1.885 for deterministic algorithms and 1.581 using randomization is possible. The best randomized algorithm so far is 1.916-competitive. Our results show that classical worst-case orders are quite rare and pessimistic for many applications.

We complement our results by providing first lower bounds. A competitive ratio obtained on nearly all possible job orders must be at least 1.257. This implies a lower bound of 1.043 for both deterministic and randomized algorithms in the general model.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Scheduling, makespan minimization, online algorithm, competitive analysis, lower bound, random-order, secretary problem

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.6

Related Version *Full Version*: `https://arxiv.org/abs/2103.16340`

Funding Work Supported by the European Research Council, Grant Agreement No. 691672, Project APEG.

1 Introduction

We study one of the most basic scheduling problems, the classic problem of makespan minimization. For the classic makespan minimization problem, one is given an input set $\mathcal{J}$ of n jobs, which have to be scheduled onto m identical and parallel machines. Preemption is not allowed. Each job $J \in \mathcal{J}$ runs on precisely one machine. The goal is to find a schedule minimizing the *makespan*, i.e. the completion time of the last job. This problem admits a long line of research and countless practical applications in both, its offline variant see e.g. [31, 34] and references therein, as well as in the online setting studied in this paper.

In the online setting, jobs are revealed one by one and each has to be scheduled by an online algorithm A immediately and irrevocably without knowing the sizes of future jobs. The makespan of online algorithm A, denoted by $A(\mathcal{J}^\sigma)$, may depend on both the job set $\mathcal{J}$ and the job order σ. The optimum makespan $\mathrm{OPT}(\mathcal{J})$ only depends on the former. Traditionally, one measures the performance of A in terms of competitive analysis. The input set $\mathcal{J}$ as well as the job order σ are chosen by an adversary whose goal is to maximize the ratio $\frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$. The maximum ratio, $c = \sup_{\mathcal{J},\sigma} \frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$, is the *(adversarial) competitive ratio*. The goal is to find online algorithms obtaining small competitive ratios.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 6; pp. 6:1–6:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the classical secretary problem, the goal is to hire the best secretary out of a linearly ordered set S of candidates. Its size n is known. Secretaries appear one by one in a uniformly random order. An online algorithm can only compare secretaries it has seen so far. It has to decide irrevocably for each new arrival whether this is the single one it wants to hire. Once a candidate is hired, future ones are automatically rejected even if they are better. The algorithm fails unless it picks the best secretary. Similar to makespan minimization this problem has been long studied, see [21, 24, 25, 35, 44, 46, 47] and references therein.

This paper studies makespan minimization under the input model of the secretary problem. The adversary determines a job set of known size n. Similar to the secretary problem, these jobs are presented to an online algorithm A one by one in a uniformly random order. Again, A has to schedule each job without knowledge of the future. The expected makespan is considered. The *competitive ratio in the secretary (or random-order) model* is $c = \sup_{\mathcal{J}} \mathbf{E}_\sigma \left[\frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}\right] = \sup_{\mathcal{J}} \frac{1}{n!} \sum_{\sigma \in S_n} \frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$, the maximum ratio between the expected makespan of A and the optimum makespan. The goal is again to obtain small competitive ratios.

We propose the term *secretary model* to set this result apart from [6] where we provide a 1.8478-competitive where n, the number of jobs, is not known in advance. Not knowing n is quite restrictive and has never been considered in any other scheduling algorithm designed with random-order arrival in mind [3, 28, 51, 52]. We hope to raise attention to these two surprisingly different models. Even though for the adversarial model such information is useless; the secretary-model requires novel and significantly different approaches and leads to, as our results show, vastly better performance guarantees.

Frameworks similar to the secretary model received a lot of recent attention in the research community sparking the area of random-order analysis. Random-order analysis has been successfully applied to numerous problems such as matching [29, 36, 38, 48], various generalizations of the secretary problem [9, 24, 25, 33, 35, 44, 46], knapsack problems [10], bin packing [42], facility location [49], packing LPs [43], convex optimization [32], welfare maximization [45], budgeted allocation [50] and recently scheduling [3, 6, 28, 51, 52]. We refer to the chapter [8] for a general overview over random-order models.

For makespan minimization, the role of randomization is poorly understood. The lower bound of 1.581 from [14, 55] is considered pessimistic and exhibits quite a big gap towards the best randomized ratio of 1.916 from [2]. A main consequence of the paper is that random-order arrival allows to beat the lower bound of 1.581. This formally sets the secretary model apart from the classical adversarial setting even if randomization is involved.

Previous work. Online makespan minimization and variants of the secretary problem have been studied extensively. We only review results most relevant to this work, beginning with the traditional deterministic adversarial setting. For m identical machines, Graham [31] showed 1966 that the greedy strategy, which schedules each job onto a least loaded machine, is $\left(2 - \frac{1}{m}\right)$-competitive. This was subsequently improved in a long line of research [27, 11, 37, 1] leading to the currently best competitive ratio by Fleischer and Wahl [26], which approaches 1.9201 for $m \to \infty$. Chen et al. [15] presented a deterministic algorithm whose competitive ratio is at most $(1 + \varepsilon)$-times the optimum one, although the actual ratio remains to be determined. For general m, lower bounds are provided in [23, 12, 30, 53]. The currently best bound is due to Rudin III [53] who shows that no deterministic online algorithm can be better than 1.88-competitive.

The role of randomization in this model is not well understood. The currently best randomized ratio of 1.916 [2] barely beats deterministic guarantees. In contrast, the best lower bound approaches $\frac{e}{e-1} > 1.581$ for $m \to \infty$ [14, 55]. There has been considerable research interest in tightening these bounds.

Recent results for makespan minimization consider variants where the online algorithm obtains extra resources. In semi-online settings, additional information on the job sequence is given in advance, such as the optimum makespan [13, 39] or the total processing time of jobs [4, 16, 41, 40]. In the former model, the optimum competitive ratio lies in the interval $[1.333, 1.5]$, see [13], while for the latter the optimum competitive ratio is known to be 1.585 cf. [4, 40]. Taking this further, the advice complexity setting allows the algorithm to receive a certain number of advice bits from an offline oracle [5, 20, 41]. Other algorithms can migrate jobs [54] or offer a buffer, which they use to reorder the job sequence [22, 41].

The secretary problem is even older than scheduling [25]. We only summarize the work most relevant to this paper. Lindley [47] and Dynkin [21] first show that the optimum strategy finds the best secretary with probability $1/e$ for $n \to \infty$. Recent research focusses on many variants, among others generalizations to several secretaries [7, 44] or even matroids [9, 24, 46]. A modern version considers adversarial orders but allows prior sampling [18, 35, 8, 33]. Related models are prophet inequalities and the game of googol [17, 19].

So far, little is known for scheduling in the secretary model. Osborn and Torng [52] prove that Graham's greedy strategy is still not better than 2-competitive for $m \to \infty$. We study makespan minimization in the restricted random-order model where n is not known in advance [6] and the dual problem, Machine Covering, in the secretary model [3]. Molinaro [51] studies a very general scheduling problem. His algorithm uses n to restart itself after half the jobs are seen and has expected makespan $(1+\varepsilon)\mathrm{OPT} + O(\log(m)/\varepsilon)$. Göbel et al. [28] study scheduling on a single machine where the goal is to minimize weighted completion times. Their competitive ratio is $O(\log(n))$ whereas they show that adversarial models allow no sublinear competitive ratios.

Our contribution. We study makespan minimization for the secretary (or random-order) model in depth. We show that basic sampling ideas allow to adapt a fairly simple algorithm from the literature [4] to be 1.75-competitive. A more sophisticated algorithm vastly improves this competitive ratio to 1.535. Both algorithms are deterministic. This ratio of 1.535 beats all lower bounds for adversarial scheduling, including the bound of 1.582 for randomized algorithms. [14, 55]

Our main results focus on large number of machines, $m \to \infty$. This is in line with most recent adversarial results [3, 2, 26] and all random-order scheduling results [6, 28, 51, 52], excluding [28] who study scheduling on one machine. While adversarial guarantees are known to improve for small numbers of machines, nobody has ever, to the best of our knowledge, explored guarantees for random-order arrival on small number of machines. We prove that our simple algorithm is $\left(1.75 + O(\frac{1}{\sqrt{m}})\right)$-competitive. Explicit bounds on the term hidden in the O-notation are provided. This result indicates that the focus of contemporary analyses on the limit case is sensible and does not hide unreasonably large terms.

All upper bounds in this paper abide to the stronger measure of *nearly competitiveness* from [6]. An algorithm is required to achieve its competitive ratio not only in expectation but on nearly all input permutations. Thus, input sequences where the competitive ratios are not obtained can be considered extremely rare and pathological. Moreover, we require worst-case guarantees even for such pathological inputs. This is relevant to practical applications, where we do not expect fully random inputs. Both algorithms hold up to this stronger measure of nearly competitiveness.

A basic approach in secretary models uses sampling statistics; a small part of the input allows to predict the rest. Sampling lets us include techniques from semi-online and advice settings with two further challenges. On the one hand, the advice is imperfect and may be, albeit with low probability, totally wrong. On the other hand, the advice has to be learned, rather than being available right from the start. In the beginning "mistakes" cannot be avoided. This makes it impossible to adapt better semi-online algorithms than LightLoad, namely [33, 16, 41, 40] to our model. These algorithms need to know the total processing volume right from the start. The advanced algorithm in this paper out-competes the optimum competitive ratio of 1.585 these semi-online algorithms can achieve [1, 40]. We conjecture that this is not possible for order oblivious algorithms that solely use sampling. Order oblivious algorithms first observe a random sample and then treat the input sequence in an adversarial order [8, 33]. Our analysis indicates that LightLoad can be adapted as an order-oblivious algorithm. The 1.535-competitive algorithm does not maintain its competitive ratio in such a setting.

The 1.535-competitive main algorithm is based on a modern point of view, which, analogous to kernelization, reduces complex inputs to sets of critical jobs. A set of critical jobs is estimated using sampling. Critical jobs impose a lower bound on the optimum makespan. If the bound is high, an enhanced version of Graham's greedy strategy suffices; called the Least-Loaded-Strategy. Else, it is important to schedule critical jobs correctly. The Critical-Job-Strategy, based on sampling, estimates the critical jobs and schedules them ahead of time. An easy heuristic suffices due to uncertainty involved in the estimates. Uncertainty poses not only the main challenge in the design of the Critical-Job-Strategy. On a larger scale, it also makes it hard to decide, which of the two strategies to use. Sometimes the Critical-Job-Strategy is chosen wrongly. These cases comprise the crux of the analysis and require using random-order arrival in a novel way beyond sampling.

The analyses of both algorithms follow three steps, which leads to the situation depicted in Figure 1. In the first step, adversarial analyses give worst-case guarantees and take care of *simple job sets*. These simple sets lack structure to be exploited via random reordering but do not pose problems to online algorithms. We thus are reduced to non-simple inputs. Non-simple random sequences have useful properties with high probability. They are "sampleable" and do not have too many problematic jobs clustered at the end of the sequence. A second step formalizes this, introducing *stable sequences*. Non-stable sequences are rare and negligible, we are thus reduced to stable sequences. The third step is a classical adversarial analysis that uses the properties of stable sequences to again establish worst-case guarantees.

The paper concludes with lower bounds. We show that no algorithm, deterministic or randomized, is better than nearly 1.257-competitive. This immediately implies a lower bound of 1.043 in the general secretary model.

Notation. We use the notation $[\mathcal{J}]$ or $[\mathcal{J}^\sigma]$ to highlight values that depend on the job set $\mathcal{J}$ or the ordered job sequence $\mathcal{J}^\sigma$. Such appendage is omitted when the dependency needs not be highlighted. In similar vein, we may write OPT for OPT$(\mathcal{J})$.

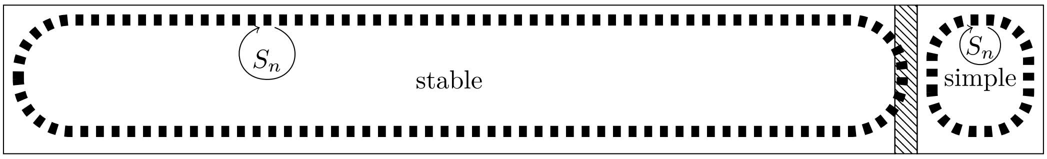


Figure 1 The lay of the land in our analysis. The algorithm is $(c+\varepsilon)$-competitive on simple and stable sequences. Only the small unstable remainder (hashed) is problematic. Dashed lines mark orbits under the action of the permutation group S_n. Simple sequences stay simple under permutation. Non-simple orbits have at most an ε-fraction, which is unstable (hashed). Thus, the algorithm is $(c+\varepsilon)$-competitive with probability at least $1-\varepsilon$ after random permutation.

2 A strong measure of random-order competitiveness

Consider a job set $\mathcal{J} = \{J_1, \dots, J_n\}$ of known size n. Each job is fully defined[1] by its non-negative size (or processing time) $p_1, \dots, p_n$. Let S_n be the group of permutations of the integers from 1 to n, which we consider a probability space under the uniform distribution. We pick each permutation with probability $1/n!$. Each permutation $\sigma \in S_n$, called an *order*, gives us a *job sequence* $\mathcal{J}^\sigma = J_{\sigma(1)}, \dots, J_{\sigma(n)}$. Recall that traditionally an online algorithm A is called *c-competitive* for some $c \geq 1$ if we have for all job sets $\mathcal{J}$ and job orders σ that $A(\mathcal{J}^\sigma) \leq c\mathrm{OPT}(\mathcal{J})$. We call this the *adversarial model.*

In the *secretary model* we consider the expected makespan of A under a uniformly random job order, i.e. $\mathbf{E}_{\sigma \sim S_n}[A(\mathcal{J}^\sigma)] = \frac{1}{n!}\sum_{\sigma \in S_n} A(\mathcal{J}^\sigma)$. We use the term *secretary model*, to distinguish this setting from the *random-order model* in [6] where the input size n is not known in advance. The algorithm A is *c-competitive in the secretary model* if we have $\mathbf{E}_{\sigma \sim S_n}[A(\mathcal{J}^\sigma)] \leq c\mathrm{OPT}(\mathcal{J})$ for all input sets $\mathcal{J}$.

The secretary model tries to lower the impact of particularly badly ordered sequences by looking at competitive ratios only in expectation. Interestingly, the scheduling problem allows for a stronger measure of random-order competitiveness for large m, called *nearly competitiveness* [6]. One requires the given competitive ratio to be obtained on nearly all sequences – not only in expectation – as well as a bound on the adversarial competitive ratio as well. We recall the definition and the main fact, that an algorithm is already c-competitive in the secretary model if it is nearly c-competitive.

▶ **Definition 1.** *A deterministic online algorithm A is called* nearly c-competitive *if the following two conditions hold.*

- *The algorithm A achieves a constant competitive ratio in the adversarial model.*
- *For every $\varepsilon > 0$, we can find $m(\varepsilon)$ such that for all machine numbers $m \geq m(\varepsilon)$ and all job sets $\mathcal{J}$ there holds $\mathbf{P}_{\sigma \sim S_n}[A(\mathcal{J}^\sigma) \geq (c+\varepsilon)OPT(\mathcal{J})] \leq \varepsilon$.*

▶ **Lemma 2.** *If a deterministic online algorithm is nearly c-competitive, then it is c-competitive in the secretary model for $m \to \infty$, i.e. for its competitive ratio c_m on m machines holds $\lim_{m\to\infty} c_m = c$.*

3 Basic properties

Let us fix an input set $\mathcal{J}$. Graham [31] establishes that his greedy strategy is 2-competitive. He considers the *average load* $L = L[\mathcal{J}] = \frac{1}{m}\sum_{i=1}^m p_i$, which is the same for any schedule of the jobs in $\mathcal{J}$, and the maximum size of any job $p_{\max} = \max_i p_i$. Both are lower bounds for OPT. Indeed, even the best schedule cannot have all machines loads below average, i.e. smaller than L, and the machine containing the largest job has load at least $p_{\max}$. Now, Graham observes that the smallest load in any schedule cannot exceed the average load L. Greedily using the least loaded machine causes makespan at most $L + p_{\max} \leq 2\mathrm{OPT}$. The greedy strategy is thus 2-competitive.

Graham's argument builds the foundation for subsequent work on scheduling problems. The following proposition guarantees a (small) constant adversarial ratio for almost every sensible random-order algorithm, which is necessary for obtaining nearly competitiveness.

[1] We propose for completeness that jobs of similar size are indistinguishable. A unique identification, say the index or a hash value, could in theory be used to derandomize a randomized algorithm. All of the results in this paper hold independently of whether such identification is possible.

▶ **Proposition 3.** *Assume job J is scheduled on a machine M such that at most $i-1$ machines have strictly smaller load than M. Then load of M is at most $\left(\frac{m}{m-i}+1\right) OPT$ afterwards.*

Proof. Let l be the load of M prior to receiving job J. By assumption at least $m-i$ machines have load l. Thus $L \geq \frac{m-i}{m} l$. We schedule job J of size at most OPT on machine M of load at most $l \leq \frac{m}{m-i} L \leq \frac{m}{m-i}\mathrm{OPT}$. The resulting load is at most $\left(\frac{m}{m-i}+1\right) OPT$. ◀

The previous result cannot be improved in general. The most difficult adversarial sequences have $L \approx p_{\max} \approx \mathrm{OPT}$. Random-order arrival faces further challenges. Certain degenerate sequences, where few jobs carry all the load, are not suited for reordering arguments. See Figure 2. This "degeneracy" is measured by $R(\mathcal{J}) = \min(\frac{L}{p_{\max}}, 1)$. Adapting the previous arguments we obtain the following result, which indicates good performance in almost all situations if $R(\mathcal{J})$ is small.

▶ **Proposition 4.** *Let M be a machine such that at most $i-1$ machines have strictly smaller load than M. If M receives a job, its load is at most $\left(\frac{m}{m-i}R(\mathcal{J})+1\right) OPT$ afterwards.*

Proof. Adapt the previous proof using that $L \leq R(\mathcal{J})\mathrm{OPT}$. ◀

Proposition 3 and 4 form the basis of our analyses. They give conditions when to use the Least-Loaded-Strategy in the main algorithm, establish most of our worst-case guarantees and explain why we can exclude simple sequences like the one in Figure 2. In the full paper, we generalize these propositions further, which is required for the main algorithm.

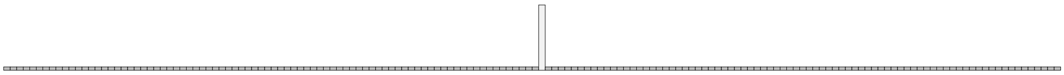


Figure 2 A surprisingly difficult sequence for random-order arguments. The big job carries most of the processing volume. Other jobs are negligible. Thus, all permutations look basically the same. Such "simple" job sets need to be excluded before the main analysis.

3.1 Sampling for Scheduling Problems

We now explain how we use sampling in the secretary model. Consider any input permutation $\mathcal{J}^\sigma = J_{\sigma(1)} \dots J_{\sigma(n)}$. A standard technique is to sample the φ-fraction of jobs, $J_{\sigma(1)} \dots J_{\sigma(\lceil \varphi n \rceil)}$, to make predictions about $\mathcal{J}^\sigma$. The previous section gives two prime candidates for sampling which relate to OPT, namely L and $p_{\max}$. Directly "sampling" OPT is futile.

The size $p_{\max}$ is best estimated by $p_{\max}^{\varphi t} = \max(p_{\sigma(t')} \mid \sigma(t') < \varphi n + 1)$. This corresponds to how we try to estimate the best secretary in the secretary-problem. Of course, $p_{\max}^{\varphi t}$ may vastly underestimate $p_{\max}$. If the sequence contains only a single huge job, this job is unlikely to be observed in the sample. Still, only very few jobs can have size exceeding $p_{\max}^{\varphi t}$ on random-order sequences; only $1/\varphi$ in expectation. The main algorithm uses reserve machines to catch these "exceptional" jobs.

For L we can get an unbiased[2] estimator from the sample: $L_\varphi = \frac{1}{\varphi m} \sum_{\sigma(t) \leq \varphi n} p_i$. Of course, we still need to determine how close L_φ is to L. Can we say that with high probability $L_\varphi \approx L$? For the sequence in Figure 2 such a statement cannot be true. The main observation

[2] The estimator is unbiased, i.e. $E[L_\varphi] = L$, if φn is a natural number. For general n, we could have replaced the factor $\frac{1}{\varphi m}$ in the definition of L_φ by the more complicated expression $\frac{n}{\lceil \varphi n \rceil m}$.

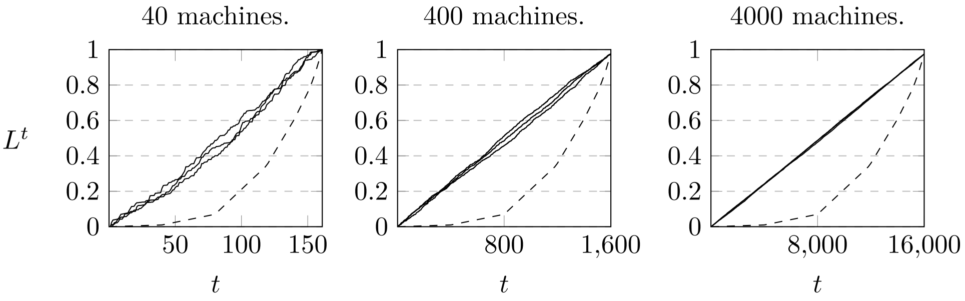


Figure 3 A graphic depicting the average load over time on the classical lower bound sequence from [1] for 40, 400 and 4000 machines. The dashed line corresponds to the original adversarial order. The three solid lines, corresponding to random permutations, clearly approximate a straight line. Thus, sampling allows to predict the (final) average load.

is that these counterexamples tend to have a small value $R(\mathcal{J})$. Given a lower bound $R_{\text{low}} > 0$ on $R(\mathcal{J})$ the following Load Lemma establishes $L_\varphi \approx L$. We have seen in the previous section that sequences with $R(\mathcal{J}) < R_{\text{low}}$ pose no major obstruction. The results in the previous section guarantee arbitrarily good performance if we choose $R_{\text{low}} > 0$ small enough.

The Load Lemma is quite potent and thus fundamental to random-order makespan minimization. It may be somewhat surprising to researchers on related problems since it makes implicit use of having non-small input sizes. Note that for our problem small inputs of size less than m are trivially scheduled optimally.

▶ **Lemma 5** (Load Lemma [6]). *Let $R_{\text{low}} = R_{\text{low}}(m) > 0$, $1 \geq \varphi = \varphi(m) > 0$ and $\varepsilon = \varepsilon(m) > 0$ be three functions in m such that $\varepsilon^{-4}\varphi^{-1}R_{\text{low}}^{-1} = o(m)$. Then there exists a variable $m(R_{\text{low}}, \varphi, \varepsilon)$, depending on these three functions, such that for $m \geq m(R_{\text{low}}, \varphi, \varepsilon)$ machines and all job sets $\mathcal{J}$ with $R(\mathcal{J}) \geq R_{\text{low}}$ and $|\mathcal{J}| \geq m$:*

$$\mathbf{P}_{\sigma\sim S_n}\left[\left|\frac{L_\varphi[\mathcal{J}^\sigma]}{L[\mathcal{J}]} - 1\right| \geq \varepsilon\right] < \varepsilon.$$

A less general version of the Load Lemma already appeared in [6]. While the Load Lemma gives only asymptotic guarantees simulations show that it requires not very large numbers of machines. Figure 4 shows the expected value of $\left|\frac{L_{1/4}[\mathcal{J}^\sigma]}{L[\mathcal{J}]} - 1\right|$ on a suitable benchmark sequence.

For our more sophisticated algorithm we also use sampling to estimate the size of critical jobs. Consider a job class $\mathcal{C}$ of size $n_{\mathcal{C}} \in O(m)$. A consequence of Chebyshev's inequality, detailed in the full version, shows that we can estimate $n_{\mathcal{C}}$ up to an additive summand of $m^{3/4}$ after sampling a $\frac{1}{\log(m)}$-fraction of the sequence. In fact the load lemma is proven by sampling job classes obtained through geometric rounding.

4 A simple 1.75-competitive algorithm

We modify the semi-online algorithm LightLoad from the literature to obtain a very simple nearly 1.75-competitive algorithm. For any $0 \leq t \leq n$, let M_{mid}^t be a machine having the $\lfloor m/2 \rfloor$-lowest load at time t, i.e. right before job J_{t+1} is scheduled. Let l_{mid}^t be its load and let l_{low}^t be the smallest load of any machine.

Let $\delta = \delta(m)$ be a certain *margin of error* our algorithm allows. It is optimal to set $\delta = 0$ but then the analysis requires a generalization of the result in [4]. In order for our main result to be self-contained one may set $\delta = \frac{1}{\log(m)}$, which allows to use results from [4] as a black box.

Given an input sequence $\mathcal{J}^\sigma$ we know from Section 3 that $\hat{L}_{\text{pre}} = \hat{L}_{\text{pre}}[\mathcal{J}^\sigma] = \frac{L_{1/4}[\mathcal{J}^\sigma]}{1-\delta}$ provides a good estimate of the (final) average load $L = \frac{1}{m}\sum_{i=1}^{m} p_i$. We use the index "pre" since our main algorithm later will use a slightly different guess $\hat{L}$. Consider the following adaptation LightLoadROM of the algorithm LightLoad from Albers and Hellwig [4].

Algorithm 1 The algorithm LightLoadROM.

```
Let $J_t$ be the job to be scheduled and let $p_t$ be its size.
if $t < n/4$ or $l_{\text{low}}^{t-1} \leq 0.25\hat{L}_{\text{pre}}$ or $l_{\text{mid}}^{t-1} + p_t > 1.75\hat{L}_{\text{pre}}$ then
    Schedule $J_t$ on any least loaded machine;
else schedule $J_t$ on $M_{\text{mid}}^{t-1}$;
```

▶ Remark 6. The first condition in the **if**-statement, $t < n/4$, already implies $l_{\text{low}}^{t-1} \leq 0.25\hat{L}_{\text{pre}}$ and is thus technically superfluous. We added it to clarify that LightLoadROM can be implemented as an online algorithm and only needs to know $\hat{L}_{\text{pre}}$ once $t \geq n/4$.

If we replace $\hat{L}_{\text{pre}}$ in the previous pseudocode by the average load L, we recover the semi-online algorithm LightLoad for makespan minimization, which has been analyzed by Albers and Hellwig [4]. They show that the algorithm is 1.75-competitive for $L = \hat{L}_{\text{pre}}$. We can show that the algorithm can also be used for general values $\hat{L} \approx L$. The performance gracefully decreases with $|L - \hat{L}_{\text{pre}}|$.

▶ **Theorem 7.** *Let $\mathcal{J}^\sigma$ be any (ordered) input sequence. The makespan of* LightLoadROM *on $\mathcal{J}^\sigma$ is at most* $1.75\left(1 + \frac{|\hat{L}_{\text{pre}}[\mathcal{J}^\sigma] - L|}{L}\right)\text{OPT}$.

Proof Sketch. For $\hat{L}_{\text{pre}} = L$, this is the main result in [4].

ItFor $\hat{L}_{\text{pre}} \geq L$, we can reduce ourselves to the case $\hat{L}_{\text{pre}} = L$. Consider any machine M in the optimum schedule of $\mathcal{J}$ that has load $l_M < \max(\hat{L}_{\text{pre}}, \text{OPT})$. We assign an additional job J_M of size $p_M = \max(\hat{L}_{\text{pre}}, \text{OPT}(\mathcal{J})) - l_M$ to this machine. For the resulting job set $\mathcal{J}'$ clearly $\text{OPT}(\mathcal{J}') = L(\mathcal{J}') = \max(\hat{L}_{\text{pre}}, \text{OPT})$. We can apply the main result of [4] to see that LightLoad has makespan at most $1.75\max(\hat{L}_{\text{pre}}, \text{OPT}(\mathcal{J}))$ if it first schedules the jobs $\mathcal{J}^\sigma$ (in order σ) followed by the additional jobs. But on the prefix $\mathcal{J}^\sigma$ LightLoad behaves precisely like LightLoadROM on input $\mathcal{J}^\sigma$. Thus, LightLoadROM has makespan at most $1.75\max(\hat{L}_{\text{pre}}[\mathcal{J}^\sigma], \text{OPT}(\mathcal{J}))$. Then, the theorem follows for $\hat{L}_{\text{pre}} \geq L$ since $\hat{L}_{\text{pre}} \leq \left(1 + \frac{\hat{L}_{\text{pre}} - L}{L}\right)L \leq \left(1 + \frac{|\hat{L}_{\text{pre}}[\mathcal{J}^\sigma] - L|}{L}\right)\text{OPT}$.

If $\hat{L}_{\text{pre}} \leq L$, the statement of the theorem still holds. can be derived similar to the analysis in [4]. Unfortunately, it cannot be immediately deduced from their results. Instead, their proofs need to be adapted. We sketch the necessary adaptations in the full version. ◀

The previous theorem already establishes a constant adversarial competitive ratio of 7. Use that $0 \leq \hat{L}_{\text{pre}} \leq L_{1/4} \leq 4L$ implies $|\hat{L}_{\text{pre}}[\mathcal{J}^\sigma] - L| \leq 3L$. We can improve this result, most importantly, if $R(\mathcal{J})$ is small.

▶ **Lemma 8.** *For any (ordered) job sequence $\mathcal{J}^\sigma$ the makespan of* LightLoadROM *is at most $(1 + 2R(\mathcal{J}))\text{OPT}(\mathcal{J})$. In particular, it is at most $3\,\text{OPT}(\mathcal{J})$ in general and at most $1.75\,\text{OPT}(\mathcal{J})$ for $R(\mathcal{J}) < 3/8$.*

Proof. Since LightLoadROM only considers the least or the $\lfloor m/2 \rfloor$-th least loaded machine, the lemma follows from Proposition 4. ◀

We now establish the competitive ratio of LightLoadROM in the strong model of nearly competitiveness. Corollary 10 follows immediately by Lemma 2.

▶ **Theorem 9.** *The algorithm* LightLoadROM *is nearly* 1.75*-competitive.*

▶ **Corollary 10.** LightLoadROM *is* 1.75*-competitive in the secretary model for* $m \to \infty$*.*

Proof of Theorem 9. Our analysis forms a triad outlining how we analyze the more sophisticated 1.535-competitive main algorithm. See Figure 1 for an illustration. Since we only prove the case $\hat{L} \geq L$ of Theorem 7, we will not rely on the case $L \leq \hat{L}$ in this proof. For this, we need to set $\delta(m) = \frac{1}{\log(m)}$.

Analysis basics. By Lemma 8 algorithm LightLoadROM is 3-competitive in the adversarial model. The first condition of nearly competitiveness is satisfied. We call input set $\mathcal{J}$ *simple* if $|\mathcal{J}| \leq m$ or $R[\mathcal{J}] < \frac{3}{8}$. Observe that LightLoadROM is (adversarially) 1.75-competitive on simple job sets. Indeed, if $|\mathcal{J}| < m$ LightLoadROM assigns every job to an empty least-loaded machine, which is obviously optimal. If $R[\mathcal{J}] < \frac{3}{8}$, Lemma 8 bounds the competitive ratio by $1 + 2R[\mathcal{J}] < 1.75$. We thus are left to consider non-simple, so called *proper*, job sets.

Stable job sequences. We call a sequence $\mathcal{J}^\sigma$ *stable* if $L \leq \hat{L}_{\text{pre}} \leq \frac{1+\delta(m)}{1-\delta(m)} L$. If a sequence is proper, it fulfills the conditions of the Load Lemma with $\varphi = 1/4$, $R_{\text{low}} = \frac{3}{8}$ and $\varepsilon(m) = \delta(m) = 1/\log(m) \in \omega(m^{-1/4})$. The Load Lemma guarantees that for m large enough, $\mathbf{P}_{\sigma \sim S_n}\left[\left|\frac{L_\varphi[\mathcal{J}^\sigma]}{L[\mathcal{J}]} - 1\right| \geq \delta\right] < \delta$. Note that $\left|\frac{L_\varphi[\mathcal{J}^\sigma]}{L[\mathcal{J}]} - 1\right| < \delta$ is equivalent to $(1-\delta)L < L_\varphi[\mathcal{J}^\sigma] < (1+\delta)L$, which in turn implies that $L \leq \hat{L}_{\text{pre}} \leq \frac{1+\delta(m)}{1-\delta(m)} L$. Thus, the probability of the sequence $\mathcal{J}^\sigma$ being stable is at least $1 - \delta$ for m large enough and $\mathcal{J}$ proper.

Adversarial Analysis. By Theorem 7, the makespan of LightLoadROM on stable sequences with $L \leq \hat{L}_{\text{pre}} \leq \frac{1+\delta(m)}{1-\delta(m)} L$ is at most $1.75 \cdot \frac{1+\delta(m)}{1-\delta(m)} \text{OPT} = \left(1.75 + \frac{3.5 \cdot \delta(m)}{1-\delta(m)}\right) \text{OPT}(\mathcal{J})$. We only require the easy case, $L \leq \hat{L}_{\text{pre}}$ of Theorem 7, which is fully proven in this paper.

Conclusion. Let $\varepsilon > 0$. Since $\delta(m) \to 0$, we can choose m large enough such that $\frac{3.5\delta(m)}{1-\delta(m)} \leq \varepsilon$. In particular $\mathbf{P}_{\sigma \sim S_n}[\text{LightLoadROM}(\mathcal{J}^\sigma) \geq (1.75 + \varepsilon) OPT(\mathcal{J})] \leq \delta(m) \leq \varepsilon$ since the only sequences where the inequality does not hold are proper but not stable. This concludes the second condition of nearly competitivity.

The δ-term. Setting $\delta = 0$ can increase the $\frac{|\hat{L}_{\text{pre}}[\mathcal{J}^\sigma] - L|}{L}$-term in Theorem 7 by at most $1/\log(m)$, which vanishes for $m \to \infty$. Of course, in reality LightLoadROM improves for $\delta = 0$. ◀

Analyzing the algorithm LightLoadROM on small numbers of machines.

From now on, we consider LightLoadROM with $\delta = 0$. Thus, the average load L is estimated by $\hat{L}_{\text{pre}} = L_{1/4}$. The *normalized absolute mean deviation* of $\hat{L}_{\text{pre}} = L_{1/4}$ is defined as $\text{NMD}(\hat{L}_{\text{pre}}) = \mathbf{E}_{\sigma \sim S_n}\left[\frac{|\hat{L}_{\text{pre}}[\mathcal{J}^\sigma] - L|}{L}\right]$. The following is a consequence of Theorem 7 .

▶ **Theorem 11.** *On input set $\mathcal{J}$ the competitive ratio of* LightLoadROM *in the secretary model is at most* $1.75(1 + \text{NMD}(\hat{L}_{\text{pre}}))$.

In the full version we give an estimation on $\text{NMD}(\hat{L}_{\text{pre}})$, which leads to the following result.

▶ **Theorem 12.** *The competitive ratio of* LightLoadROM *is* $1.75 + \frac{18}{\sqrt{m}} + O\left(\frac{1}{m}\right)$.

The techniques presented in this section can, in theory, be extended to analyze the main algorithm in the next section. This is impractical due to the complexity of the analysis at hand. We are certain that the error term involved will be of the form $m^{-1/a}$ for a small.

The constant summand $\frac{18}{\sqrt{m}}$ in Theorem 12 is pessimistic. We discuss several avenues of further improvement in the full version. The best we are aware of allows for a competitive ratio as small as $\frac{4.4}{\sqrt{m}} + \frac{7}{m} + O\left(\frac{1}{m^{3/2}}\right)$ but there are ways to improve even further. The terms still hidden in the O-notation result from the Stirling-approximation and are known to be tiny. Figure 4 shows $\mathrm{NMD}(\hat{L}_{\mathrm{pre}})$ on the lower bound from [1], which is a sensible benchmark.

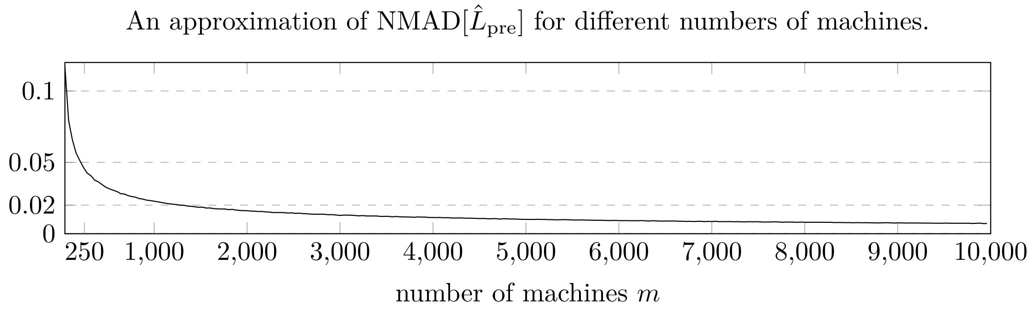


Figure 4 The extra cost for small numbers of machines. The graph shows an estimation of $\mathrm{NMD}(\hat{L}_{\mathrm{pre}})$ on the lower bound sequence from [1] based on $10,000$ random samples. Theorem 11 this indicates good performance of LightLoadROM in practice.

5 The nearly 1.535-competitive algorithm

The new main algorithm achieves a competitive ratio of $c = \frac{1+\sqrt{13}}{3} \approx 1.535$. It consists of three components: a sampling phase, the Least-Loaded-Strategy and the Critical-Job-Strategy. We now give a simplified description of the algorithm.

The sampling phase. A few jobs are sampled to predict the whole sequence. These Jobs are scheduled greedily with a some machines kept in reserve. This phase is uninformed and "mistakes" are unavoidable. Such mistakes are few, since the processing volume scheduled is small – at least if we exclude worst-case sequences. First, we sample B, which tries to estimate $\max(p_{\max}, L) \leq \mathrm{OPT}$. We then use sampling to predict *critical* jobs of size in between $(c-1)B$ and B. Intuitively, jobs smaller than $(c-1)B$ are too small to pose a problem. Jobs larger than B are also critical but cannot be predicted since they did not appear during sampling. This in turn means that they are rare. We keep a few reserve machines to safely process them.

The Critical-Job-Strategy. Our plan is to assign critical jobs ahead of time. Formally, placeholder jobs are used to reserve space for jobs yet to come. Critical jobs are assigned according to an easy heuristic: Each machine gets either one big or two medium jobs. Reserve machines handle errors in the predictions and unexpected huge jobs.

The Least-Loaded-Strategy. Sometimes the Critical-Job-Strategy is not feasible; there simply are too many critical jobs. This may already by apparent from sampling predictions, but for some job sets this cannot be predicted. The latter input sets form the crux of the analysis. Once we find out, we pick the Least-Loaded-Strategy, which enhances a Graham's greedy approach by still maintaining reserve machines for particularly large jobs. Intuitively, many critical jobs make it even for OPT impossible to schedule all jobs efficiently, which is why we rely on this less sophisticated strategy.

Further challenges. Algorithm design and analysis have to deal with three further issues. First, the Critical-Job-Strategy needs to take scheduling decisions made during sampling into account. Second, a consequence of sampling is that no value is exact, small sources of errors are imminent. Third, we need a constant competitive ratio against an adversary. All these challenges impact details of the algorithm design in rather subtle ways.

5.1 Formal Description

Let $\delta = \delta(m) = \frac{1}{\log(m)}$ be the *margin of error our algorithm allows.* Most of the time, it is sensible to treat δ as a constant and forget about its dependency on m. Our algorithm maintains a set of $\lceil \delta m \rceil$ *reserve machines.* Their complement are the *principal machines.* Let us fix an input sequence $\mathcal{J}^\sigma$. Let $\hat{L} = \hat{L}[\mathcal{J}^\sigma] = L_{\delta^2}[\mathcal{J}^\sigma]$. For simplicity, we hide the dependency on $\mathcal{J}^\sigma$ whenever possible. Our online algorithm uses $B = \max\left(p_{\max}^{\delta^2 n}, \hat{L}\right)$ as an *estimated lower bound for* OPT. This bound is known after the first $\lfloor \delta^2 n \rfloor$ jobs are treated. Our algorithm uses geometric rounding implicitly. Given a job J_t of size p_t let $f(p_t) = (1+\delta)^{\lfloor \log_{1+\delta} p_t \rfloor}$ be its *rounded size.* We also call J_t an $f(p_t)$-job.

Using rounded sizes, we introduce job classes. Let $p_{\text{small}} = c - 1 = \frac{\sqrt{13}-2}{3} \approx 0.535$ and $p_{\text{big}} = \frac{c}{2} = \frac{1+\sqrt{13}}{6} \approx 0.768$. We call job J_t

- *small* if $f(p_t) \leq p_{\text{small}} B$ and *critical* else,
- *big* if $f(p_t) > p_{\text{big}} B$,
- *medium* if J is neither small nor big, i.e. $p_{\text{small}} B \leq f(p_t) \leq p_{\text{big}} B$,
- *huge* if its (not-rounded) size exceeds B, i.e. $B < p_t$, and *normal* else.

Consider the set $\mathcal{P} = \{(1+\delta)^i \mid p_{\text{small}} B \leq (1+\delta)^i \leq B\}$ corresponding to rounded sizes of critical jobs. Given $p \in \mathcal{P}$ let n_p be the total number of p-jobs. We could estimate n_p by $\delta^{-2}\hat{n}_p$ after sampling where $\hat{n}_p = |\{J_{\sigma(j)} \mid \sigma(j) \leq \delta^2 n \wedge J_{\sigma(j)} \text{ is a } p\text{-job}\}|$ after sampling. In practice we need a more complicated guess: $c_p = \max\left(\left\lfloor \left(\delta^{-2}\hat{n}_p - m^{3/4}\right) w(p) \right\rfloor, \hat{n}_p\right) w(p)^{-1}$. It has two advantages. The value c_p is close to n_p with high probability, but unlikely to exceed it. Overestimating n_p turns out to be far worse than underestimating it. It also simplifies the description of the algorithm allowing medium jobs to always "pair up".

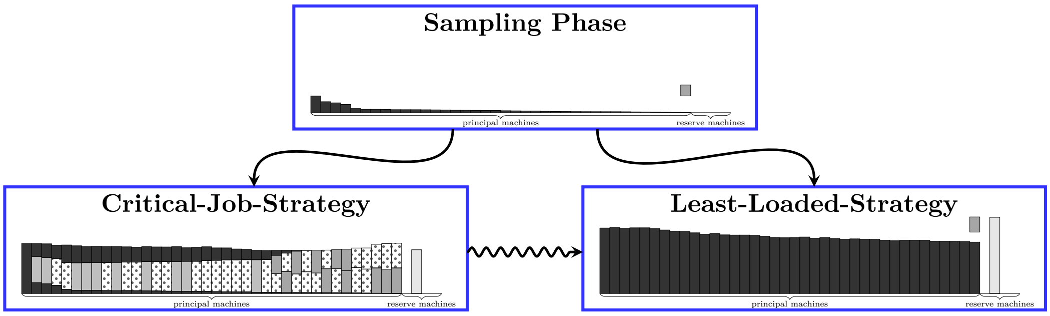


Figure 5 The 1.535-competitive algorithm. First, few jobs are sampled. Then, the algorithm decides between two strategies. The Critical-Job-Strategy tries to schedule critical jobs ahead of time. The Least-Loaded-Strategy follows a greedy approach, which reserves some machines for large jobs. Sometimes, we realize very late that the Critical-Job-Strategy does not work and have to switch to the Least-Loaded-Strategy "on the fly". We never switch in the other direction.

Statement of the algorithm. If there are less jobs than machines, each job is placed onto a separate machine. This is optimal. Else, a short sampling phase greedily assigns each of the first $\lfloor \delta^2 n \rfloor$ jobs to the least loaded principal machine. Now, B and $(c_p)_{p\in\mathcal{P}}$ are known. We choose the Least-Loaded-Strategy if we predict the Critical-Job-Strategy to be infeasible. Formally, if $\sum_{p\in\mathcal{P}} w(p)c_p > m$, where $w(p) = 1/2$ for $p \le p_{\text{big}}$ and $1 > p \le p_{\text{big}}$. If $\sum_{p\in\mathcal{P}} w(p)c_p \le m$, we choose the Critical-Job-Strategy. The Critical-Job-Strategy requires a one-time preparation. It may later switch to the Least-Loaded-Strategy but it never switches the other way around.

The Least-Loaded-Strategy

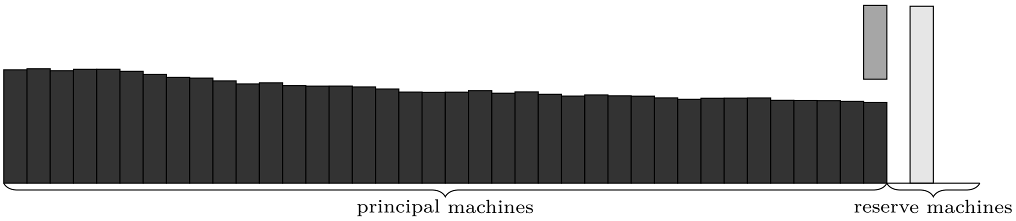


Figure 6 The Least-Loaded-Strategy schedules jobs greedily. A few machines are reserved for unexpected huge jobs. For example the largest job, which is unlikely to arrive in the sampling phase.

The Least-Loaded-Strategy places any normal job on a least loaded principal machine. Huge jobs are scheduled on a least loaded reserve machine. This reserve machine will be empty, unless we consider rare and pathological worst-case orders.

The Critical-Job-Strategy

For the Critical-Job-Strategy we introduce *p-placeholder-jobs* for every size $p \in \mathcal{P}$. Sensibly, the size of a p-placeholder-job is p. During the Critical-Job-Strategy we treat placeholder-jobs similar to *real* jobs. They are assigned in the **Preparation for the Critical-Job-Strategy**. We try to pair off medium jobs, some of which already arrived during sampling. Moreover it is important to assign fewer processing volume to those machines, which have a higher load after the sampling phase.

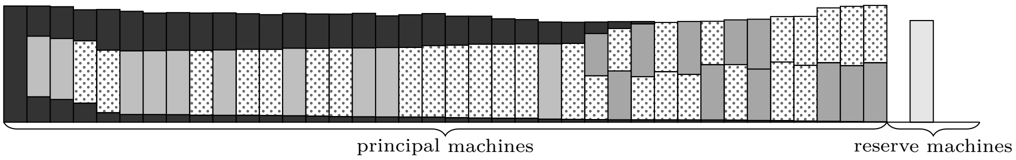


Figure 7 The Critical-Job-Strategy. Each machine gets either two medium, one large or no critical job. Placeholder jobs (dotted) reserve space for critical jobs yet to come. Processing volume of small jobs (dark) on the bottom arrived during the sampling phase. Reserve machines accommodate huge jobs or, possibly, jobs without matching placeholders.

The **Critical-Job-Strategy** places small jobs on least-loaded principal machines taking placeholders into account. Critical jobs of rounded size $p \in \mathcal{P}$ replace p-placeholder-jobs whenever possible. If no matching placeholder is found or if the current job is exceptional, the reserve machines are used. Again, medium jobs are paired up. If the reserve machines are full, the algorithm *fails*. It switches to the Least-Loaded-Strategy.

The full description of the main algorithm is provided in Appendix B.

6 Analysis of the algorithm

Theorem 13 is main result of the paper.

▶ **Theorem 13.** *Our algorithm is nearly c-competitive. Recall that $c = \frac{1+\sqrt{13}}{3} \approx 1.535$.*

Due to Lemma 2 this competitive ratio also holds in the general secretary model.

▶ **Corollary 14.** *Our algorithm is c-competitive in the secretary model as $m \to \infty$.*

The analysis of the algorithm proceeds along the same three reduction steps used in the proof of Theorem 9. First, we assert that our algorithm has a constant adversarial competitive ratio, which approaches 1 as $R(\mathcal{J}) \to 0$. Not only does this lead to the first condition of nearly competitiveness, it also enables us to introduce *simple* job sets on which we perform well due to basic considerations resulting from Section 3.

▶ **Definition 15.** *A job set $\mathcal{J}$ is called* simple *if $R(\mathcal{J}) \leq \frac{(1-\delta)\delta^2}{\delta^2+1}(c-1)$ or if it consists of at most m jobs. Else, we call it* proper*. We call any ordered input sequence $\mathcal{J}^\sigma$* simple *respectively* proper *if the underlying set $\mathcal{J}$ has this property.*

▶ **Main Lemma 16.** In the adversarial model our algorithm has competitive ratio $4 + O(\delta)$ on general input sequences and $c + O(\delta)$ on simple sequences.

We are thus reduced to treating *proper* job sets. In the second reduction we introduce *stable* sequences. These have many desirable properties. Most notably, they are suited to sampling. Their formal definition can be found later in Definition 22. The second reduction shows that stable sequences arise with high probability if the order of a proper job set $\mathcal{J}$ is picked uniformly randomly.

Formally, for m the number of machines, let $P(m)$ be the maximum probability by which the permutation of any proper sequence may not be stable, i.e.

$$P(m) = \sup_{\mathcal{J} \text{ proper}} \mathbf{P}_{\sigma\sim S_n}\left[\mathcal{J}^\sigma \text{ is not stable}\right].$$

The second main lemma asserts that this probability vanishes as $m \to \infty$.

▶ **Main Lemma 17.** $\lim_{m\to\infty} P(m) = 0$.

In other words, non-stable sequences are very rare and of negligible impact in random-order analyses. Thus, we only need to consider stable sequences. In the final, third, step we analyze our algorithm on stable sequences. This analysis is quite general. In particular, it does not rely further on random-order arrival. Instead, we work with worst-case stable inputs, i.e. we allow an adversary to present any stable input sequence.

▶ **Main Lemma 18.** Our algorithm is adversarially $(c+O(\delta))$-competitive on stable sequences.

These three main lemmas allow us to conclude the proof of Theorem 13.

Proof of Theorem 13. Recall that $\delta(m) \to 0$ for $m \to \infty$. By Main Lemma 16, the first condition of nearly competitiveness holds, i.e. our algorithm has a constant competitive ratio. Moreover, by Main Lemma 16 and Main Lemma 18, given $\varepsilon > 0$, we can pick $m_0(\varepsilon)$ such that our algorithm is $(c+\varepsilon)$-competitive on all sequences that are stable or simple, if there are at least $m_0(\varepsilon)$ machines. This implies that for $m \geq m_0(\varepsilon)$ the probability of our algorithm not being $(c+\varepsilon)$-competitive is at most $P(m)$, the maximum probability with which a random permutation of a proper input sequence is not stable. By Main Lemma 17, we can find $m(\varepsilon) \geq m_0(\varepsilon)$ such that $P(m) \leq \varepsilon$ for $m \geq m(\varepsilon)$. This choice of $m(\varepsilon)$ satisfies the second condition of nearly competitiveness. ◀

Proof sketch of Main Lemma 16

The *anticipated load* $\tilde{l}_M^t$ of a machine M at time t denotes its load including placeholder-jobs. We can obtain the following two bounds on the average anticipated load $\tilde{L} = \sup_t \frac{1}{m} \sum_M \tilde{l}_M^t$.

▶ **Lemma 19.** *We have $\tilde{L} \leq L + 2p_{\max}$, as well as $\tilde{L} \leq \left(1 + \frac{1}{\delta^2}\right) L$.*

Thus the average anticipated load $\tilde{L}$ relates to the original values L, $p_{\max}$. In the full version, we generalize Proposition 4 to anticipated loads. We can then use Lemma 19 to conclude Main Lemma 16. The only exception are reserve machines, which receives its last job using the Critical-Job-Strategy. Their load needs to be bounded using different techniques.

Formally, we can prove the following two statements, which imply Main Lemma 16.

▶ **Proposition 20.** *The main algorithm is adversarially $\left(1 + \frac{3}{1-\delta} + 2\delta\right)$-competitive.*

▶ **Proposition 21.** *The main algorithm has makespan at most $(c + 2\delta)\mathrm{OPT}$ on simple sequences $\mathcal{J}^\sigma$.*

Stable job sequences and a proof sketch of Main Lemma 17

We introduce the class of *stable* job sequences. The first two conditions state that all estimates our algorithm makes are *accurate*, i.e. sampling works. By the third condition there are less exceptional jobs than reserve machines and the fourth condition states that these jobs are distributed evenly. The final condition is a technicality. Stable sequences are useful since they occur with high probability if we randomly order a proper job set.

▶ **Definition 22.** *A job sequence $\mathcal{J}^\sigma$ is* stable *if the following conditions hold:*

- *The estimate $\hat{L}$ for L is accurate, i.e. $(1-\delta)L \leq \hat{L} \leq (1+\delta)L$.*
- *The estimate c_p for n_p is accurate, i.e. $c_p \leq n_p \leq c_p + 2m^{3/4}$ for all $p \in \mathcal{P}$.*
- *There are at most $\lceil \delta m \rceil$ exceptional jobs in $\mathcal{J}^\sigma$.*
- *Let $\tilde{t}$ be the time the last exceptional job arrived and let $n_{p,\tilde{t}}$ be the number of p-jobs scheduled at that time for a given $p \in \mathcal{P}$. Then $n_{p,\tilde{t}} \leq \left(1 - \delta^3\right) n_p$ for every $p \in \mathcal{P}$.*
- $\delta^3 \left\lfloor \left(1 - \delta - 2\delta^2\right) m / |\mathcal{P}| \right\rfloor \geq 2|\mathcal{P}| m^{3/4}$.

Proof sketch of Main Lemma 17. The first condition follows from Lemma 5. The second condition can be derived using Chebyshev's inequality as discussed at the end of Section 3.1. Both conditions require that only proper sequences are considered. The third condition is equivalent to demanding one of the $\lceil \delta m \rceil$ largest jobs to occur during the sampling phase. This is extremely likely. In expectation the rank of the largest job occurring in the sampling phase is δ^{-2}, a constant. The fourth condition states that the exceptional jobs are evenly spread throughout the sequence compared to the p-jobs for any $p \in \mathcal{P}$. Again, this is expected of a random sequence and corresponds to how one would determine randomness statistically. For the final condition it suffices to choose the number of machines m large enough. One technical problem arises since the set of rounded critical job sizes $\mathcal{P} = \mathcal{P}[\mathcal{J}^\sigma]$ is defined using the value $B[\mathcal{J}^\sigma]$. It thus highly depends on the input permutation σ. We rectify this by passing over to a larger class $\hat{\mathcal{P}}$ such that $\mathcal{P} \subset \hat{\mathcal{P}}$ with high probability. ◀

Proof sketch of Main Lemma 18

We first consider the Critical-Job-Strategy. Main Lemma 18 holds as long as it is employed.

▶ **Lemma 23.** *The makespan of our algorithm is at most $(c + O(\delta)) \max(B, L, p_{\max})$ on stable sequences till it employs the Least-Loaded-Strategy (or till the end of the sequence).*

Proof sketch. Let us only consider critical jobs at any time the Least-Loaded-Strategy is not employed. We can then show that a machine contains either one big job or at most two medium jobs. In the first case we bound the size of this big, possibly exceptional, job by $p_{\max}$. Else, if the machine contains two medium jobs their total weight is at most $2(1+\delta)p_{\text{big}}B = (1+\delta)cB$. The factor $(1+\delta)$ arises since we use rounded sizes in the definition of medium jobs. Thus, critical jobs may cause a load of at most $\max(p_{\max}, (c + O(\delta))B)$.

Analyzing the load increase caused by small jobs requires techniques similar to the proof of Main Lemma 16. ◀

Note that for stable sequences $\hat{L} \leq (1+\delta)L \leq (1+\delta)\text{OPT}$, in particular $\max(B, L, p_{\max}) = \max\left(p_{\max}^{\delta^2 n}, \hat{L}, L, p_{\max}\right) \leq (1+\delta)\text{OPT}$. This proves the following corollary to Lemma 23.

▶ **Corollary 24.** *Till our algorithm uses the Least-Loaded-Strategy its makespan is less than* $(c + O(\delta))\text{OPT}$ *on stable sequences.*

Hence, we are left to consider the Least-Loaded-Strategy. We say the algorithm *fails* if it has to switch from the Critical-Job-Strategy to the Least-Loaded-Strategy. The following lemma is crucial and relies deeply on the properties of stable sequences, in particular the fourth one.

▶ **Lemma 25.** *If the algorithm fails, every exceptional job has been scheduled.*

The lemma shows that the Least-Loaded-Strategy only needs to deal with exceptional jobs if it is picked immediately. In this case, all reserve machines are empty. The third property of stable sequences ensures that there are enough reserve machines so that every exceptional job is assigned to an empty machine.

Non-exceptional jobs, i.e. jobs of size at most B, are scheduled onto a least loaded principal machine. This machine was among the $\delta m + 1$ least loaded machines and had load at most $mL/(m - \delta m + 1)$ by Proposition 4. Afterwards, its load was at most $mL/(m-\delta m+1) + B \leq (2 + O(\delta))B$ since $(1 - \delta)L \leq B$ for stable sequences. The following lemma concludes the proof of Main Lemma 18 since it implies that $(2 + O(\delta))B \leq (c + O(\delta))\text{OPT}$.

▶ **Lemma 26.** *If the Least-Loaded-Strategy is applied on a stable sequence,* $B \leq \frac{c}{2}\text{OPT}$.

The proof of Lemma 26 is left to the full version.

7 Lower bounds

We establish the following theorem using two lower bound sequences. These results generalize to randomized algorithms using appropriate notions of (nearly) competitiveness.

▶ **Theorem 27.** *For every online algorithm A, deterministic or randomized, there exists a job set $\mathcal{J}$ such that* $\mathbf{P}_{\sigma\sim S_n}\left[A(\mathcal{J}^\sigma) \geq \frac{\sqrt{73}-1}{6}\text{OPT}(\mathcal{J})\right] \geq \frac{1}{6}$. *If A is randomized the previous probability also includes its random choices.*

The lower bound highlights the inability of the main algorithm to decide between the Least-Loaded-Strategy and the Critical-Job-Strategy. If we could communicate this decision, say through a single advice bit, our main algorithm would become nearly optimal, i.e. nearly 1-competitive, on the lower bound sets. Theorem 27 implies the following lower bounds.

▶ **Corollary 28.** *If an online algorithm A is nearly c-competitive, then* $c \geq \frac{\sqrt{73}-1}{6} \approx 1.257$.

▶ **Corollary 29.** *The best competitive ratio possible in the secretary model is* $\frac{\sqrt{73}+29}{36} \approx 1.043$.

The lower bounds are proven in the appendix.

References

1 S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999. Publisher: SIAM.

2 S. Albers. On randomized online scheduling. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 134–143, 2002.

3 S. Albers, W. Gálvez, and M. Janke. Machine covering in the random-order model. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

4 S. Albers and M. Hellwig. Semi-online scheduling revisited. *Theoretical Computer Science*, 443:1–9, 2012. Publisher: Elsevier.

5 S. Albers and M. Hellwig. Online makespan minimization with parallel schedules. *Algorithmica*, 78(2):492–520, 2017. Publisher: Springer.

6 S. Albers and M. Janke. Scheduling in the Random-Order Model. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. unpublished, 2020.

7 S. Albers and L. Ladewig. New results for the k-secretary problem. *arXiv preprint*, 2020. arXiv:2012.00488.

8 Pablo Azar, Robert Kleinberg, and S. Weinberg. Prophet inequalities with limited information. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, July 2013. doi:10.1137/1.9781611973402.100.

9 M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg. Matroid Secretary Problems. *Journal of the ACM (JACM)*, 65(6):1–26, 2018. Publisher: ACM New York, NY, USA.

10 M. Babaioff, N. Immorlica, D. Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *Approximation, randomization, and combinatorial optimization. Algorithms and techniques*, pages 16–28. Springer, 2007.

11 Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 51–58, 1992.

12 Y. Bartal, H. J. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Inf. Process. Lett.*, 50(3):113–116, 1994.

13 Martin Böhm, Jiří Sgall, Rob Van Stee, and Pavel Veselỳ. A two-phase algorithm for bin stretching with stretching factor 1.5. *Journal of Combinatorial Optimization*, 34(3):810–828, 2017.

14 B. Chen, A. van Vliet, and G. J. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Information Processing Letters*, 51(5):219–222, 1994. Publisher: Elsevier.

15 L. Chen, D. Ye, and G. Zhang. Approximating the optimal algorithm for online scheduling problems via dynamic programming. *Asia-Pacific Journal of Operational Research*, 32(01):1540011, 2015. Publisher: World Scientific.

16 T.C.E. Cheng, H. Kellerer, and V. Kotov. Semi-on-line multiprocessor scheduling with given total processing time. *Theoretical computer science*, 337(1-3):134–146, 2005. Publisher: Elsevier.

17 J. Correa, A. Cristi, B. Epstein, and J. Soto. The two-sided game of googol and sample-based prophet inequalities. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2066–2081. SIAM, 2020.

18 J. Correa, A. Cristi, L. Feuilloley, T. Oosterwijk, and A. Tsigonias-Dimitriadis. The secretary problem with independent sampling. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2047–2058. SIAM, 2021.

19 J. Correa, P. Dütting, F. Fischer, and K. Schewior. Prophet inequalities for iid random variables from an unknown distribution. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, pages 3–17, 2019.

20 J. Dohrau. Online makespan scheduling with sublinear advice. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 177–188. Springer, 2015.

21 E. B. Dynkin. The optimum choice of the instant for stopping a Markov process. *Soviet Mathematics*, 4:627–629, 1963.

22 M. Englert, D. Özmen, and M. Westermann. The power of reordering for online minimum makespan scheduling. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 603–612. IEEE, 2008.

23 U. Faigle, W. Kern, and G. Turán. On the performance of on-line algorithms for partition problems. *Acta cybernetica*, 9(2):107–119, 1989.

24 M. Feldman, O. Svensson, and R. Zenklusen. A simple O (log log (rank))-competitive algorithm for the matroid secretary problem. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1189–1201. SIAM, 2014.

25 T. S. Ferguson. Who solved the secretary problem? *Statistical science*, 4(3):282–289, 1989. Publisher: Institute of Mathematical Statistics.

26 R. Fleischer and M. Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000. Publisher: Wiley Online Library.

27 G. Galambos and G. J. Woeginger. An on-line scheduling heuristic with better worst-case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22(2):349–355, 1993. Publisher: SIAM.

28 O. Göbel, T. Kesselheim, and A. Tönnis. Online appointment scheduling in the random order model. In *Algorithms-ESA 2015*, pages 680–692. Springer, 2015.

29 G. Goel and A. Mehta. Online budgeted matching in random input models with applications to Adwords. In *SODA*, volume 8, pages 982–991, 2008.

30 T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 564–565, 2000.

31 R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966. Publisher: Wiley Online Library.

32 A. Gupta, R. Mehta, and M. Molinaro. Maximizing Profit with Convex Costs in the Random-order Model. *arXiv preprint*, 2018. `arXiv:1804.08172`.

33 A. Gupta and S. Singla. Random-order models. In Tim Roughgarden, editor, *Beyond worst-case analysis*. Cambridge University Press, 2020.

34 D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987. Publisher: ACM New York, NY, USA.

35 H. Kaplan, D. Naori, and D. Raz. Competitive Analysis with a Sample and the Secretary Problem. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2082–2095. SIAM, 2020.

36 C. Karande, A. Mehta, and P. Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 587–596, 2011.

37 D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20(2):400–430, 1996. Publisher: Elsevier.

38 R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, 1990.

39 H. Kellerer and V. Kotov. An efficient algorithm for bin stretching. *Operations Research Letters*, 41(4):343–346, 2013. Publisher: Elsevier.

40 H. Kellerer, V. Kotov, and M. Gabay. An efficient algorithm for semi-online multiprocessor scheduling with given total processing time. *Journal of Scheduling*, 18(6):623–630, 2015.

41 H. Kellerer, V. Kotov, M. Grazia Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, 1997. Publisher: Elsevier.

42 C. Kenyon. Best-Fit Bin-Packing with Random Order. In *SODA*, volume 96, pages 359–364, 1996.

43 T. Kesselheim, A. Tönnis, K. Radke, and B. Vöcking. Primal beats dual on online packing LPs in the random-order model. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 303–312, 2014.

44 R. D. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *SODA*, volume 5, pages 630–631, 2005.

45 N. Korula, V. Mirrokni, and M. Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. *SIAM Journal on Computing*, 47(3):1056–1086, 2018. Publisher: SIAM.

46 O. Lachish. O (log log rank) competitive ratio for the matroid secretary problem. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 326–335. IEEE, 2014.

47 D. V. Lindley. Dynamic programming and decision theory. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 10(1):39–51, 1961. Publisher: Wiley Online Library.

48 M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 597–606, 2011.

49 A. Meyerson. Online facility location. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 426–431. IEEE, 2001.

50 V.S. Mirrokni, S. O. Gharan, and M. Zadimoghaddam. Simultaneous approximations for adversarial and stochastic online budgeted allocation. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1690–1701. SIAM, 2012.

51 M. Molinaro. Online and random-order load balancing simultaneously. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1638–1650. SIAM, 2017.

52 C. J. Osborn and E. Torng. List's worst-average-case or WAC ratio. *Journal of Scheduling*, 11(3):213–215, 2008. Publisher: Springer.

53 J. Rudin III. *Improved bounds for the on-line scheduling problem.* PhD thesis, University of Phoenix, 2001.

54 P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009. Publisher: INFORMS.

55 J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63(1):51–55, 1997. Publisher: Citeseer.

A Lower bounds

We establish the following theorem using two lower bound sequences.

▶ **Theorem 27.** *For every online algorithm A, deterministic or randomized, there exists a job set $\mathcal{J}$ such that $\mathbf{P}_{\sigma\sim S_n}\left[A(\mathcal{J}^\sigma) \geq \frac{\sqrt{73}-1}{6}\mathrm{OPT}(\mathcal{J})\right] \geq \frac{1}{6}$. If A is randomized the previous probability also includes its random choices.*

Theorem 27 implies the following lower bounds.

▶ **Corollary 28.** *If an online algorithm A is nearly c-competitive, then $c \geq \frac{\sqrt{73}-1}{6} \approx 1.257$.*

▶ **Corollary 29.** *The best competitive ratio possible in the secretary model is $\frac{\sqrt{73}+29}{36} \approx 1.043$.*

Let us now prove these results. For this section let $c = \frac{\sqrt{73}-1}{6}$ be our main lower bound on the competitive ratio. We consider three types of jobs:

1. *negligible jobs* of size 0 (or a tiny size $\varepsilon > 0$ if one were to insist on positive sizes).
2. *big jobs* of size $1 - \frac{c}{3} = \frac{17-\sqrt{37}}{18} \approx 0.581$.
3. *small jobs* of size $\frac{c}{3} = \frac{1+\sqrt{37}}{18} \approx 0.419$

Let $\mathcal{J}$ be the job set consisting of m jobs of each type.

▶ **Lemma 30.** *There exists a schedule of $\mathcal{J}$ where every machine has load 1. Every schedule that has a machine with smaller load has makespan at least c.*

Proof. This schedule is achieved by scheduling a type 2 and a type 3 job onto each machine. The load of each machine is then 1. Every schedule which allocates these jobs differently must have at least one machine M which contains at least three jobs of type 2 or 3 by the pigeonhole principle. The load of M is then at least $3\frac{c}{3} = c$. ◀

Given a permutation $\mathcal{J}^\sigma$ of $\mathcal{J}$ and an online algorithm A, which expects $3m+1$ jobs to arrive in total. Let $A(\mathcal{J}^\sigma, 3m+1)$ denote its makespan after it processes $\mathcal{J}^\sigma$ expecting yet another job to arrive. Let $P = \mathbf{P}[A(\mathcal{J}^\sigma, 3m+1) = 1]$ be the probability that A achieves the optimal schedule where every machine has load 1 under these circumstances. Depending on P we pick one out of two input sets on which A performs bad.

Let $j \in \{1, 2\}$. We now consider the job set $\mathcal{J}_j$ consisting of m jobs of each type plus one additional job of type j, i.e. a negligible job if $j = 1$ and a big one if $j = 2$. We call an ordering $\mathcal{J}_j^\sigma$ of $\mathcal{J}_j$ *good* if it ends with a job of type j or, equivalently, if its first $3m$ jobs are a permutation of $\mathcal{J}$. Note that the probability of $\mathcal{J}^\sigma$ being good is $\frac{m+1}{3m+1} \geq \frac{1}{3}$ for $\sigma \sim S_{3m+1}$.

▶ **Lemma 31.** *For job set $\mathcal{J}_1$ we have $\mathbf{P}_{\sigma\sim S_n}[A(\mathcal{J}_1^\sigma) \geq c\mathrm{OPT}(\mathcal{J})] \geq \frac{1-P}{3}$ and for job set $\mathcal{J}_2$ furthermore $\mathbf{P}_{\sigma\sim S_n}[A(\mathcal{J}_2^\sigma) \geq c\mathrm{OPT}(\mathcal{J})] \geq \frac{P}{3}$.*

Proof. Consider a good permutation of $\mathcal{J}_1$. Then with probability $1-P$ the algorithm A does have makespan c even before the last job is scheduled. On the other hand $\mathrm{OPT}(\mathcal{J}_1) = 1$. Thus with probability $\frac{1-P}{3}$ we have $A(\mathcal{J}_1^\sigma) = c = c\mathrm{OPT}(\mathcal{J}_1)$.

Now consider a good permutation of $\mathcal{J}_2$. Then, with probability P, algorithm A has to schedule the last job on a machine of size 1. Its makespan is thus $2 - \frac{c}{3} = c^2$ by our choice of c. The optimum algorithm may schedule two big jobs onto one machine, incurring load $2 - \frac{2c}{3} < c$, three small jobs onto another one, incurring load c and one job of each type onto the remaining machines, causing load $1 < c$. Thus $\mathrm{OPT}(\mathcal{J}_2) = c$. In particular we have with probability $\frac{P}{3}$ that $A(\mathcal{J}_2^\sigma) = c^2 = c\mathrm{OPT}(\mathcal{J}_2)$. ◀

We now conclude the main three lower bound results.

▶ **Theorem 27.** *For every online algorithm A, deterministic or randomized, there exists a job set $\mathcal{J}$ such that $\mathbf{P}_{\sigma\sim S_n}\left[A(\mathcal{J}^\sigma) \geq \frac{\sqrt{73}-1}{6}\mathrm{OPT}(\mathcal{J})\right] \geq \frac{1}{6}$. If A is randomized the previous probability also includes its random choices.*

Proof. By the previous lemma we get that

$$\max_{j=1,2}\left(\mathbf{P}_{\sigma\sim S_n}\left[A(\mathcal{J}_j^\sigma) \geq c\mathrm{OPT}(\mathcal{J})\right]\right) = \max\left(\frac{1-P}{3}, \frac{P}{3}\right) \geq \frac{1}{6}.$$ ◀

▶ **Corollary 28.** *If an online algorithm A is nearly c-competitive, then $c \geq \frac{\sqrt{73}-1}{6} \approx 1.257$.*

Proof. This is immediate by the previous theorem. ◀

▶ **Corollary 29.** *The best competitive ratio possible in the secretary model is $\frac{\sqrt{73}+29}{36} \approx 1.043$.*

Proof. Let A be any online algorithm. Pick a job set $\mathcal{J}$ according to Theorem 27. Then

$$A^{\mathrm{rom}}(\mathcal{J}) = \mathbf{E}_{\sigma\sim S_n}[A(\mathcal{J}^\sigma)] \geq \frac{1}{6}\cdot\frac{\sqrt{73}-1}{6}\mathrm{OPT}(\mathcal{J}) + \frac{5}{6}\mathrm{OPT}(\mathcal{J}) = \frac{\sqrt{73}+29}{36}\mathrm{OPT}(\mathcal{J}).$$ ◀

B Full description of the main algorithm

Our new algorithm achieves a competitive ratio of $c = \frac{1+\sqrt{13}}{3} \approx 1.535$. Let $\delta = \delta(m) = \frac{1}{\log(m)}$ be the *margin of error our algorithm allows.* Throughout the analysis it is mostly sensible to treat δ as a constant and forget about its dependency on m. Our algorithm maintains a certain set $\mathcal{M}_{\text{res}}$ of $\lceil \delta m \rceil$ *reserve machines.* Their complement, the *principal machines*, are denoted by $\mathcal{M}$. Let us fix an input sequence $\mathcal{J}^\sigma$. Let $\hat{L} = \hat{L}[\mathcal{J}^\sigma] = L_{\delta^2}[\mathcal{J}^\sigma]$. For simplicity, we hide the dependency on $\mathcal{J}^\sigma$ whenever possible. Our online algorithm uses $B = \max\left(p_{\max}^{\delta^2 n}, \hat{L}\right)$ as an *estimated lower bound for* OPT. This bound B is known after the first $\lfloor \delta^2 n \rfloor$ jobs are treated. Our algorithm uses geometric rounding implicitly. Given a job J_t of size p_t let $f(p_t) = (1+\delta)^{\lfloor \log_{1+\delta} p_t \rfloor}$ be its *rounded size.* We also call J_t an $f(p_t)$-job. Using rounded sizes, we introduce job classes. Let $p_{\text{small}} = c - 1 = \frac{\sqrt{13}-2}{3} \approx 0.535$ and $p_{\text{big}} = \frac{c}{2} = \frac{1+\sqrt{13}}{6} \approx 0.768$. Then we call job J_t

- *small* if $f(p_t) \leq p_{\text{small}} B$ and *critical* else,
- *big* if $f(p_t) > p_{\text{big}} B$,
- *medium* if J is neither small nor big, i.e. $p_{\text{small}} B \leq f(p_t) \leq p_{\text{big}} B$,
- *huge* if its (not-rounded) size exceeds B, i.e. $B < p_t$, and *normal* else.

Consider the sets $\mathcal{P}_{\text{med}} = \{(1+\delta)^i \mid (1+\delta)^{-1} p_{\text{small}} B < (1+\delta)^i \leq p_{\text{big}} B\}$ and $\mathcal{P}_{\text{big}} = \{(1+\delta)^i \mid p_{\text{big}} B < (1+\delta)^i \leq B\}$ corresponding to all possible rounded sizes of medium respectively big jobs, excluding huge jobs. Let $\mathcal{P} = \mathcal{P}_{\text{med}} \cup \mathcal{P}_{\text{big}}$. This subdivision gives rise to a *weight function*, which will be important later. Let $w(p) = 1/2$ for $p \in \mathcal{P}_{\text{med}}$ and $w(p) = 1$ for $p \in \mathcal{P}_{\text{big}}$. The elements $p \in \mathcal{P}$ define job classes $\mathcal{C}_p \subseteq \mathcal{J}$ consisting of all p-jobs, i.e. jobs of rounded size p. By some abuse of notation, we call the elements in $\mathcal{P}$ "job classes", too. We let $n_p = |\mathcal{C}_p|$ and $\hat{n}_p = |\{J_{\sigma(j)} \mid \sigma(j) \leq \delta^2 n \wedge J_{\sigma(j)} \text{ is a } p\text{-job}\}|$. We want to use the values $\hat{n}_p$, which are available to an online algorithm quite early, to estimate the values n_p, which accurately describe the set of critical jobs. First, $\delta^{-2}\hat{n}_p$ comes to mind as an estimate for n_p. Yet, we need a more complicated guess: $c_p = \max\left(\left\lfloor \left(\delta^{-2}\hat{n}_p - m^{3/4}\right) w(p) \right\rfloor, \hat{n}_p\right) w(p)^{-1}$. It has three desirable advantages. First, for every $p \in \mathcal{P}$ the value c_p is close to n_p with high probability, but, opposed to $\delta^{-2}\hat{n}_p$, unlikely to exceed it. Overestimating n_p turns out to be far worse than underestimating it. Second, $w(p)c_p$ is an integer and, third, we have $c_p \geq \hat{n}_p w(p)^{-1}$. A fundamental fact regarding the values $(c_p)_{p \in \mathcal{P}}$ and B is, of course, that they are known to the online algorithm once $\lfloor \delta^2 n \rfloor$ jobs are scheduled.

Algorithm 2 The complete algorithm: How to schedule job J_t.

```
STRAT is initialized to CRITICAL, $J_t$ is the job to be scheduled.
if $n \leq m$ then Schedule $J_t$ on any empty machine;
else if $t \leq \varphi n$ then schedule $J_t$ on a least loaded machine in $\mathcal{M}$;    ▷ Sampling phase
else
    if we have $t = \lfloor \varphi n \rfloor + 1$ then
        if $\sum_{p \in \mathcal{P}} w(p) c_p > m$ then STRAT ← LEAST-LOADED
        else proceed with the Preparation for the Critical-Job-Strategy (Algorithm 4);
    if STRAT = CRITICAL then proceed with the Critical-Job-Strategy (Algorithm 5);
    else proceed with the Least-Loaded-Strategy (Algorithm 3);
```

Statement of the algorithm. If there are less jobs than machines, i.e. $n \leq m$, it is optimal to put each job onto a separate machine. Else, a short sampling phase greedily schedules each of the first $\lfloor \delta^2 n \rfloor$ jobs to the least loaded principal machine $M \in \mathcal{M}$. Now, the values

B and $(c_p)_{p\in\mathcal{P}}$ are known. Our algorithm has to choose between two strategies, the Least-Loaded-Strategy and the Critical-Job-Strategy, which we will both introduce subsequently. It maintains a variable STRAT, initialized to CRITICAL, to remember its choice. If it chooses the Critical-Job-Strategy, some additional preparation is required. It may at any time discover that the Critical-Job-Strategy is not feasible and switch to the Least-Loaded-Strategy but it never switches the other way around.

The **Least-Loaded-Strategy** places any normal job on a least loaded principal machine. Huge jobs are scheduled on any least loaded reserve machine. This machine will be empty, unless we consider rare worst-case orders.

Algorithm 3 The Least-Loaded-Strategy: How to schedule job J_t.

```
if $J_t$ is huge then schedule $J_t$ on any least loaded reserve machine;
else schedule $J_t$ on any least loaded principal machine;
```

For the Critical-Job-Strategy we introduce *p-placeholder-jobs* for every size $p \in \mathcal{P}$. Sensibly, the size of a p-placeholder-job is p. During the Critical-Job-Strategy we treat placeholder-jobs similar to *real* jobs. The *anticipated load* $\tilde{l}_M^t$ of a machine M at time t is the sum of all jobs on it, including placeholder-job, opposed to the common load l_M^t, which does not take the latter into account. Note that $\tilde{l}_M^t$ defines a pseudo-load as introduced in Section 3.

During the **Preparation for the Critical-Job-Strategy** the algorithm maintains a counter c'_p of all p-jobs scheduled so far (including placeholders). A job class $p \in \mathcal{P}$ is called *unsaturated* if $c'_p \leq c_p$. First, we add unsaturated medium placeholder-jobs to any principal machine that already contains a medium real job from the sampling phase. We will see in Lemma 32 that such an unsaturated medium job class always exists. Now, let m_{empty} be the number of principal machines which do not contain critical jobs. We prepare a set $\mathcal{J}_{\text{rep}}$ of cardinality at most m_{empty}, which we will then schedule onto these machines. The set $\mathcal{J}_{\text{rep}}$ may contain single big placeholder-jobs or pairs of medium placeholder-jobs. We greedily pick any unsaturated job class $p \in \mathcal{P}$ and add a p-placeholder-job to $\mathcal{J}_{\text{rep}}$. If p is medium, we pair it with a job belonging to any other, not necessarily different, unsaturated medium job class. Such a job class always exists by Lemma 32. We stop once all job classes are saturated or if $|\mathcal{J}_{\text{rep}}| = m_{\text{empty}}$. We then assign the elements in $\mathcal{J}_{\text{rep}}$ to machines. We iteratively pick the element $e \in \mathcal{J}_{\text{rep}}$ of maximum size and assign the corresponding jobs to the least loaded principal machine, which does not contain critical jobs yet. Sensibly, the size of a pair of jobs in $\mathcal{J}_{\text{rep}}$ is the sum of their individual sizes. We repeat this until all jobs and job pairs in $\mathcal{J}_{\text{rep}}$ are assigned to some principal machine.

Algorithm 4 Preparation for the Critical-Job-Strategy.

```
while there is a machine $M$ containing a single medium job do
    Add a placeholder $p$-job for an unsaturated size class $p \in \mathcal{P}_{\text{med}}$ to $M$; $c'_p \leftarrow c'_p + 1$;
while there is an unsaturated size class $p \in \mathcal{P}$ and $|\mathcal{J}_{\text{rep}}| < m_{\text{empty}}$ do
    Pick an unsaturated size class $e = p \in \mathcal{P}$ with $c'_p$ minimal; $w(e) \leftarrow p$; $c'_p \leftarrow c'_p + 1$;
    if $p$ is medium then pick $q \in \mathcal{P}_{\text{med}}$ unsaturated. $e \leftarrow (p, q)$; $w(e) \leftarrow p+q$; $c'_q \leftarrow c'_q+1$;
    Add $e$ to $\mathcal{J}_{\text{rep}}$;
while $\mathcal{J}_{\text{rep}} \neq \emptyset$ do
    Pick a least loaded machine $M \in \mathcal{M}$, which does not contain a critical job yet;
    Pick $e \in \mathcal{J}_{\text{rep}}$ of maximum size $w(e)$ and add the jobs in $e$ to $M$;
    $\mathcal{J}_{\text{rep}} \leftarrow \mathcal{J}_{\text{rep}} \setminus \{e\}$;
```

▶ **Lemma 32.** *In line 2 and 5 of Algorithm 4 there is always an unsaturated medium size class available. Thus, Algorithm 4, the Preparation for the Critical-Job-Strategy, is well defined.*

Proof. Concerning line 2, there are precisely $\sum_{p\in\mathcal{P}_{\text{med}}}\hat{n}_p$ machines with critical jobs while there are at least $\sum_{p\in\mathcal{P}_{\text{med}}}(c_p-\hat{n}_p)\geq\sum_{p\in\mathcal{P}_{\text{med}}}\hat{n}_p$ placeholder-jobs available to fill them. Here we make use of the fact that for medium jobs $p\in\mathcal{P}_{\text{med}}$ we have $c_p\geq\hat{n}_p w(p)^{-1}=2\hat{n}_p$.

Concerning line 5, observe that so far every machine and every element in $\mathcal{J}_{\text{rep}}$ contains an even number of medium jobs. If the placeholder picked in line 4 was the last medium job remaining, $\sum_{p\in\mathcal{P}_{\text{med}}}c_p$ would be odd. But this is not the case since every c_p for $p\in\mathcal{P}_{\text{med}}$ is even. ◀

Algorithm 5 The Critical-Job-Strategy.

```
if $J_t$ is medium or big then let $p$ denote its rounded size;
    if there is a machine $M$ containing a $p$-placeholder-job $J$ then
        Delete the $p$-placeholder-job $J$ and assign $J_t$ to $M$;
    else if $J_t$ is medium and there exists $M \in \mathcal{M}_{\text{res}}$ containing a single medium job then
        Schedule $J_t$ on $M$;
    else if there exists an empty machine $M \in \mathcal{M}_{\text{res}}$ then schedule $J_t$ on $M$;
    else STAT ← LEAST-LOADED;                                ▷ We say the algorithm fails.
        use the Least-Loaded-Strategy (Algorithm 3) from now on;
else assign $J_t$ to the least loaded machine in $\mathcal{M}$ (take placeholder jobs into account);
```

After the Preparation is done, the **Critical-Job-Strategy** becomes straightforward. Each small job is scheduled on a principal machines with least anticipated load, i.e. taking placeholders into account. Critical jobs of rounded size $p\in\mathcal{P}$ replace p-placeholder-jobs whenever possible. If no such placeholder exists anymore, critical jobs are placed onto the reserve machines. Again, we try pair up medium jobs whenever possible. If no suitable machine can be found among the reserve machines, we have to switch to the Least-Loaded-Strategy. We say that the algorithm *fails* if it ever reaches this point. In this case, it should rather have chosen the Least-Loaded-Strategy to begin with. Since all reserve machines are filled at this point, the Least-Loaded-Strategy is impeded, too. The most difficult part of our analysis shows that, excluding worst-case orders, this is not a problem on job sets that are prone to cause failing.

One-Way Functions and a Conditional Variant of MKTP

Eric Allender
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Mahdi Cheraghchi
Department of EECS, University of Michigan, Ann Arbor, MI, USA

Dimitrios Myrisiotis
Department of Computing, Imperial College London, London, UK

Harsha Tirumala
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Ilya Volkovich
Computer Science Department, Boston College, Chestnut Hill, MA, USA

Abstract

One-way functions (OWFs) are central objects of study in cryptography and computational complexity theory. In a seminal work, Liu and Pass (FOCS 2020) proved that the average-case hardness of computing time-bounded Kolmogorov complexity is *equivalent* to the existence of OWFs. It remained an open problem to establish such an equivalence for the average-case hardness of some natural NP-complete problem. In this paper, we make progress on this question by studying a conditional variant of the Minimum KT-complexity Problem (MKTP), which we call McKTP, as follows.

1. First, we prove that if McKTP is average-case hard on a polynomial fraction of its instances, then there exist OWFs.
2. Then, we observe that McKTP is NP-complete under polynomial-time randomized reductions.
3. Finally, we prove that the existence of OWFs implies the nontrivial average-case hardness of McKTP.

Thus the existence of OWFs is inextricably linked to the average-case hardness of this NP-complete problem. In fact, building on recently-announced results of Ren and Santhanam [28], we show that McKTP is hard-on-average *if and only if* there are logspace-computable OWFs.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Problems, reductions and completeness; Theory of computation → Cryptographic primitives

Keywords and phrases Kolmogorov complexity, KT Complexity, Minimum KT-complexity Problem, MKTP, Conditional KT Complexity, Minimum Conditional KT-complexity Problem, McKTP, one-way functions, OWFs, average-case hardness, pseudorandom generators, PRGs, pseudorandom functions, PRFs, distinguishers, learning algorithms, NP-completeness, reductions

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.7

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2021/009/`

Funding *Eric Allender*: Partially supported by NSF Grants CCF-1909216 & CCF-1909683.
Mahdi Cheraghchi: Mahdi Cheraghchi's research was partially supported by the National Science Foundation under Grant No. CCF-2006455.
Dimitrios Myrisiotis: This work was partly carried out during a visit of Dimitrios Myrisiotis to DIMACS, with support from the Special Focus on Lower Bounds in Computational Complexity funded under NSF Grant CCF-1836666.
Harsha Tirumala: Harsha Tirumala was partially supported by NSF Grant CCF-1909216 and by the Simons Collaboration on Algorithms and Geometry.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 7; pp. 7:1–7:19
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We would like to thank Russell Impagliazzo for explaining his work [18] to us, and Ján Pich and Ninad Rajgopal for illuminating discussions. We thank Ján Pich for bringing his work [27] to our attention. We thank Mikito Nanashima and Hanlin Ren for helpful comments and for spotting bugs in the proofs of earlier versions of Lemma 20 and Lemma 21, respectively. In particular, we thank Hanlin Ren for asking the question of whether KT complexity would be an appropriate complexity measure to consider in the context of our work. We thank Yanyi Liu and Rafael Pass for the excellent correspondence regarding their work [20, 23, 24], and Rahul Santhanam for bringing the work by Impagliazzo and Naor [19] to our attention. Finally, we would like to thank the anonymous reviewers for their helpful feedback.

1 Introduction

One-way functions (OWFs) – that is, functions that are easy to compute but hard to invert – are objects of great importance in cryptography and computational complexity. For example, it is known that OWFs exist if and only if pseudorandom generators exist [12] and, moreover, if OWFs exist, then $\mathsf{P} \neq \mathsf{NP}$.

In this paper, we ask the following question:

> *Can the existence of OWFs be shown to be equivalent to the average-case hardness of some* NP*-complete problem?*

We take concrete steps toward giving an affirmative answer to this question, by presenting a candidate problem. Note that by Impagliazzo and Naor [19] it is known that there exists some NP-complete problem (Subset Sum) whose average-case hardness implies the existence of OWFs. However, what we attempt to do is different: We want to make concrete progress in *characterizing* OWFs by the average-case hardness of an NP-complete problem.

The importance of NP stems mainly from the fact that, for thousands of important naturally-occurring computational problems, their worst-case computational complexity is best explained by knowing that they are NP-complete. However, NP-completeness has not been as relevant for the concerns of cryptographers, who require one-way functions, which in turn require problems in NP that are hard-on-average. Liu and Pass [20] gave what is arguably the first "natural" example of a problem in NP that is hard-on-average if and only if one-way functions exist; but this problem (computing time-bounded Kolmogorov complexity, K^t) is not known to be NP-complete. Although it is not hard to modify their language to obtain an artificial NP-complete problem with the same average-case complexity (see Proposition 24), there had been no "natural" example of an NP-complete problem whose average-case complexity had been connected directly to the existence of one-way functions. Our main contribution is to present such an example.

There are different ways to define time-bounded Kolmogorov complexity; the measure KT (defined in [4]) has the property that $\mathrm{KT}(x)$ is approximately the same as the circuit complexity of the function that has x as its truth table. Thus the problem $\mathrm{MKTP} = \{(x, i) \mid \mathrm{KT}(x) \leq i\}$ has been useful [4] in studying the Minimum Circuit Size Problem $\mathrm{MCSP} = \{(f, i) \mid \mathrm{CC}(f) \leq i\}$, which has been the subject of much recent work. As with most other Kolmogorov complexity measures, $\mathrm{KT}(x)$ is defined in [4] as a special case of the conditional KT-complexity $\mathrm{KT}(x \mid y)$, where y is the empty string. Our results concern the decision problem $\mathrm{McKTP} = \{(x, y, i) \mid \mathrm{KT}(x \mid y) \leq i\}$. We show the following.

(a) If McKTP is hard-on-average, then one-way functions exist (Theorem 1).
(b) McKTP is NP-complete under randomized reductions (Theorem 2).
(c) If one-way functions exist, then McKTP is (somewhat) hard-on-average (Theorem 4).
(d) In fact, McKTP is hard-on-average if and only if logspace-computable one-way functions exist (Theorem 3 and Theorem 5).

There has been a flurry of recent activity on this topic, and it may be helpful to present the following timeline:

1. [20] is posted by Liu and Pass, proving an equivalence between the existence of OWFs and the average-case hardness of K^t complexity.
2. [6] is posted by Allender, Cheraghchi, Myrisiotis, Tirumala, and Volkovich, claiming to characterize the existence of OWFs by the average-case complexity of an NP-complete problem called Sparse Partial MCSP. This paper was retracted.
3. [5] is posted by Allender, Cheraghchi, Myrisiotis, Tirumala, and Volkovich, presenting the proofs of Item a through Item c above.
4. [21] is posted by Liu and Pass, whereby they prove that *subexponentially-hard* OWFs exist if and only if MK^tP (a decision problem based on K^t complexity) is average-case hard for *sublinear-time non-uniform* heuristics.
5. [24] is posted by Liu and Pass, showing that one-way functions exist if and only if the EXP-complete language MKtP is hard-on-average[1] and that logspace-computable one-way functions exist if and only if the PSPACE-complete language MKSP is hard-on-average.
6. [28] is posted by Ren and Santhanam, showing that MKTP is hard-on-average if and only if logspace-computable one-way functions exist. This allows us to prove Item d above.
7. [23] is posted by Liu and Pass (which is inspired by and in part a response to [6]), showing that a conditional variant of K^t complexity is NP-complete, and is hard-on-average if and only if one-way functions exist.
8. [16] is posted by Ilango, Ren, and Santhanam, showing that one-way functions exist if and only if the undecidable problem MKP (i.e., a decision problem based on Kolmogorov complexity) is hard-on-average under a samplable distribution, and if and only if MCSP is hard-on-average under a locally-sampleable distribution.
9. [22] is posted by Liu and Pass, generalizing the results of Ilango, Ren, and Santhanam [16], whereby they show that there exists some sparse language L such that OWFs exist if and only if L is average-case hard with respect to some efficiently sampleable "high-entropy" distribution.

1.1 Prior work

An early goal in cryptographic research was to base the existence of cryptographically secure one-way functions on the worst-case complexity of some NP-complete problem. This goal remains elusive; it was shown in [2] that no black-box argument of this sort can proceed based on non-adaptive reductions. Non-adaptive worst-case-to-average-case reductions were also studied by Bogdanov and Trevisan [8], who showed that such reductions to sets in NP exist only for problems in NP/poly $\cap$ coNP/poly. Recent work by Nanashima [26] holds open the possibility that the security of OWFs can be based on an *adaptive* black-box reduction, by first establishing a non-adaptive black-box reduction basing the existence of *auxiliary input one-way functions* on the worst-case complexity of an NP-complete problem, although this would also require non-relativizing techniques. Instead of worst-case hardness, the focus of our work is on average-case hardness assumptions. A nice survey on this area, that lays out many of the issues about one-way functions and average-case complexity, is the one by Bogdanov and Trevisan [7].

[1] This is also proved in [28], and was posted to ECCC one day later.

Hirahara and Santhanam have discussed zero-error average-case complexity of problems related to MKTP [14]. Santhanam [29] showed that a restricted type of hitting-set generator exists if and only if MCSP is zero-error average-case hard. Hirahara also proved similar results connecting the worst-case and the zero-error average-case complexity of problems related to MCSP and Kolmogorov complexity [13].

More recently, Brzuska and Couteau [9] discuss basing OWFs on average-case hardness, stating that it remains an open question to do this for the general notion of average-case hardness. They present some negative results, indicating the difficulty of establishing the existence of fine-grained one-way functions, based on the existence of average-case hardness, via black-box reductions.

There is also an important line of work (including Ajtai [1] and Micciancio and Regev [25]) basing the existence of OWFs on the *worst-case* complexity of certain problems in NP (including problems that are closely related to NP-complete problems, although they are not themselves known to be NP-complete).

1.2 Our results

In this work, we connect the existence of OWFs to the average-case hardness of computing a conditional (and NP-complete) variant of MKTP, which we term McKTP.

Initially, we prove that the average-case hardness of McKTP implies the existence of OWFs.

▶ **Theorem 1** (Informal). *OWFs exist if* McKTP *is hard-on-average on a polynomial fraction of its instances.*

We also show that McKTP is NP-complete under randomized reductions.

▶ **Theorem 2** (Informal). McKTP *is* NP*-complete under polynomial-time one-sided-error randomized reductions.*

Moreover, Theorem 1 suggests an approach for excluding Impagliazzo's *Pessiland* [17], that is, a version of our world where there are average-case hard problems in NP *and* there are no OWFs. This approach is based on the following observation. If McKTP is NP-hard under average-case reductions, then by Theorem 1 the existence of an average-case hard problem in NP would imply the existence of OWFs. Therefore proving that McKTP is NP-hard under average-case reductions excludes Pessiland.

We are able to prove a stronger version of Theorem 1, building on the work of Ren and Santhanam [28].

▶ **Theorem 3** (Informal). *Logspace-computable OWFs exist if* McKTP *is hard-on-average on a polynomial fraction of its instances.*

Finally, we prove a *weak* converse of Theorem 1, and a *strong* converse of Theorem 3.

▶ **Theorem 4** (Informal). *OWFs exist only if* McKTP *is hard-on-average on an* exponential *fraction of its instances.*

▶ **Theorem 5** (Informal). *Logspace-computable OWFs exist only if* McKTP *is hard-on-average on an* polynomial *fraction of its instances.*

By Theorem 3 and Theorem 5, we get the following corollary.

▶ **Corollary 6.** McKTP *is hard-on-average if and only if logspace-computable OWFs exist.*

1.2.1 How significant are our results?

The reader may wonder whether the hypothesis of Theorem 1 is overly strong. Is there perhaps some trivial heuristic that succeeds well on average for this NP-complete decision problem?

The input to the problem consists of a triple (x, y, θ), where the question is whether $\mathrm{KT}(x \mid y) \leq \theta$, where θ is a number bounded by $|x| + O(\log |x|)$. A simple heuristic is to accept if θ is at the high end of this range, and reject otherwise; one can augment this to accept for slightly lower values of θ if x has certain hallmarks of low complexity (such as starting or ending with a logarithmic number of zeros, or agreeing with y on those substrings). However, when inputs are chosen at random, this heuristic still seems likely to fail with constant probability if θ is close to the boundary between where the heuristic accepts and rejects. In particular, it is far from clear how to design a heuristic that would have failure probability less than, say $1/s^2$, where θ ranges over a domain of size s. In particular, it seems quite plausible that there is a constant k for which no heuristic can achieve failure probability less than $1/s^k$, which is precisely the hypothesis of Theorem 1, and is sufficient for the existence of OWFs.

Moreover, by Theorem 5, this hypothesis is in fact *equivalent* to the existence of logspace-computable OWFs, which is widely believed to hold.

By the same token, the conclusion of Theorem 4 gives a much weaker, but still non-trivial, average-case hardness condition for McKTP.

1.3 Our techniques

Our main results are Theorem 1, Theorem 2, and Theorem 4. Below we provide some intuition regarding their proofs.

1. Theorem 1 is proved by
   a. giving an average-case decision-to-search reduction for McKTP (see Lemma 20) and
   b. observing that a recent result by Liu and Pass [20], whereby they prove that the average-case hardness of a search variant of time-bounded Kolmogorov complexity K^t yields OWFs, can be adjusted to the case of McKTP as well (see Lemma 21).
      The three properties of time-bounded Kolmogorov complexity K^t, for some $t : \mathbb{N} \to \mathbb{N}$ where $t(n) \geq n$ for all $n \in \mathbb{N}$, that are used by Liu and Pass, are as follows.
      i. One can create a string of low time-bounded Kolmogorov complexity in polynomial time. This can be done by running a universal Turing machine U on some string, for polynomially-many steps, and subsequently recording the output of U.
      ii. For any string x, the possible values of its K^t complexity are polynomially-many in $|x|$. In fact, there is a $c > 0$ such that, for any function $t : \mathbb{N} \to \mathbb{N}$ such that $t(n) \geq n$ for all $n \in \mathbb{N}$, and any string x, the possible values of $K^t(x)$ are at most $|x| + c$.
      iii. The following domination property holds. Let $x^* \in \{0,1\}^n$ be a string, and $c > 0$ be as in Item 1(b)ii. Then,

$$\Pr_{\Pi \sim \{0,1\}^{n+c}}\left[U\left(\Pi, 1^{t(n)}\right) = x^*\right] \geq \frac{1}{2^{n+c}} = \frac{2^{-n}}{2^c} \geq \frac{\Pr_{x \sim \{0,1\}^n}[x = x^*]}{\mathrm{poly}(n)}.$$

      As it turns out, all of these properties are satisfied even when one considers McKTP.
2. Theorem 3 is proved by use of the techniques of [28]. In particular, the proof of Theorem 1 shows that the following function is one-way, if McKTP is hard-on-average:

Given (s, t, y, Π), output the string obtained by running U on y and the length-s prefix of Π for t steps.

Ren and Santhanam observe that this function is logspace-computable if we restrict t to be $O(\log n)$. Then, crucially, they show that for most strings in the range of this function, $s + t$ is minimized when $t = O(\log n)$. These insights, combined with the the proof of the preceding theorem, suffice.

3. Theorem 2 is proved by
   a. noting that McKTP is in NP (see Lemma 11) and
   b. showing the NP-hardness of McKTP (see Corollary 34). This is done by giving a polynomial-time randomized reduction from Set Cover, which is NP-hard to approximate (see Corollary 33), to an appropriate gap version of McKTP (see Corollary 32). Note that this step closely mimics the proof of Ilango [15] for the NP-hardness of Minimum Oracle Circuit Size Problem (MOCSP).
4. Theorem 4 is proved by giving a proof of its contrapositive statement, as explained by the items below.
   a. Assume that McKTP is easy on average under the uniform distribution.
   b. By a corollary of Ilango, Loff, and Oliveira, for all $a \geq 1$, there exists a learning algorithm for $\mathsf{SIZE}[n^a]$ that works for infinitely many $n \in \mathbb{N}$.
   c. By a learner-to-distinguisher reduction, for every polynomial-time computable Boolean function family $\{f_y\}_{y \in \{0,1\}^*}$, there is a distinguisher for $\{f_y\}_{y \in \{0,1\}^*}$.
   d. By the correctness of the works by Håstad, Implagliazzo, Levin, and Luby [12], and Goldreich, Goldwasser, and Micali [11], there are no OWFs.
5. Theorem 5 is proved by giving a slight modification to the proof of [28, Lemma 4.7].

1.4 Paper organization

In Section 2 we give some background knowledge and useful facts. We prove Theorem 1 in Section 3, Theorem 3 in Section 4, and Theorem 5 in Section 5. Finally, we prove Theorem 2 in Appendix B. Theorem 4 is proved in the full version of the paper [5].

2 Preliminaries

2.1 Notation

We denote the natural numbers by $\mathbb{N}$ and the positive reals by $\mathbb{R}_{>0}$. For any $n \in \mathbb{N}$, we denote the set $\{1, \ldots, n\}$ by $[n]$. Let $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ be a string of length n; we write $|x| := n$. The empty string is denoted by λ.

We denote by $\mathcal{F}_n$ the class of all Boolean functions on n variables. We identify infinite Boolean functions $f : \{0,1\}^* \to \{0,1\}$ with collections $\{f_n\}_{n \in \mathbb{N}}$, whereby $f_n : \{0,1\}^n \to \{0,1\}$ for all $n \in \mathbb{N}$.

We consider Boolean circuits over the bounded fan-in $\{\wedge_2, \vee_2, \neg\}$ basis. Given a circuit, its *size* is the number of its gates. Let $s : \mathbb{N} \to \mathbb{N}$ be a function. If we use s to upper bound the size of some circuit, then we shall call s a *size function*.

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, the *circuit complexity of* f, denoted $\mathrm{CC}(f)$, is the size of a minimum size circuit that computes f. For a size function $s : \mathbb{N} \to \mathbb{N}$, we denote by $\mathsf{SIZE}[s(n)]$ the class of Boolean functions $f = \{f_n\}_{n \in \mathbb{N}}$, whereby $f_n : \{0,1\}^n \to \{0,1\}$ for all $n \in \mathbb{N}$, such that $\mathrm{CC}(f_n) \leq s(n)$ for all $n \in \mathbb{N}$.

In this work, we do not distinguish between Turing machines and algorithms. We say that an algorithm A is a *PPT algorithm* if A is a probabilistic polynomial-time algorithm. If A is a PPT algorithm that runs in time $p(n)$ for a polynomial p, then we denote by $A(x; r)$

the output of A on input $x \in \{0,1\}^*$ using random bits $r \in \{0,1\}^{p(|x|)}$. We say that an algorithm A is a *PPT oracle algorithm* if A is a PPT algorithm that has access to some oracle. If A is a PPT oracle algorithm that runs in time $p(n)$ for a polynomial p and has access to an oracle for a language $L \subseteq \{0,1\}^*$, then we denote by $A^L(x;r)$ the output of A^L on input $x \in \{0,1\}^*$ using random bits $r \in \{0,1\}^{p(|x|)}$.

2.2 Probability theory

We will use the following useful fact from probability theory.

▶ **Lemma 7** (Markov's inequality). *If X is a non-negative random variable with $\mu := \mathbf{E}[X]$, then for all $k > 0$ it is the case that $\mathbf{Pr}[X \geq k\mu] \leq 1/k$.*

2.3 KT complexity

2.3.1 A universal Turing machine

In what follows, we fix some *efficient* universal (oracle) Turing machine (UTM) U. Let $y, \Pi, z \in \{0,1\}^*$ and $t \in \mathbb{N}$. The notation $U^{\Pi,y}(z, 1^t)$ denotes the output of U when U runs the program Π on input z for at most t steps, given that U has extended oracle access to program Π and standard oracle access to auxiliary string y. These notions are defined as follows.

1. *Standard oracle access to auxiliary string* y means that U has a standard oracle tape T_y of $\log |y|$ cells, and that in order to read a bit y_i of y, whereby $1 \leq i \leq |y|$, the machine U has to write $i \in \{0,1\}^{\log|y|}$ on T_y and then enter a question state. In the next step, the contents of T_y are erased and replaced by a bit b such that $b = y_i$.

   One important aspect of our choice of U is that, for every auxiliary string $y \in \{0,1\}^*$ and $1 \leq i \leq \log |y|$, the oracle query $y_i \overset{?}{=} 1$ is such that it *requires* time $\log |y|$, and can be implemented in time $O(\log |y|)$.
2. *Extended oracle access to program* Π means that U has a tape T_Π of $|\Pi|$ cells that contains Π, and the head of T_Π has *both* the ability to jump to an indexed location $1 \leq i \leq |\Pi|$ of T_Π, namely $T_\Pi[i] = \Pi_i$, *and* to move left and right on T_Π. Note that in the former case the index i is written in a separate tape of $\log |\Pi|$ cells, specifically allocated for that purpose. (So extended oracle access implies the existence of two tapes that help facilitate the oracle query.)

The notation $U^{\Pi,y}(z)$ denotes the output of U when U runs the program Π on input z, until Π halts (if this is the case, otherwise Π runs forever), whereby U has extended oracle access to Π and standard oracle access to y.

In this work, we will assume that whenever U is given oracle access to a program Π, this access will be *extended*, and whenever U is given oracle access to an auxiliary string y, this access will be *standard*. This is mainly to avoid unnecessary complications in the proof of Theorem 2 (where it is convenient to have sequential access to Π, while requiring that each query to y uses logarithmic time) while maintaining the trivial upper bound on KT complexity (see Lemma 8) which requires oracle access to Π.

We will also assume that the output of U will either be 1 or 0, on any input.

2.3.2 Definition of KT complexity, and some properties

Given strings $x, y \in \{0,1\}^*$, we define the *KT complexity of x given y*, denoted $\mathrm{KT}(x \mid y)$, to be the minimum value of $|\Pi| + t$ over programs $\Pi \in \{0,1\}^*$ and run-time bounds $t \in \mathbb{N}$ whereby for all $1 \leq i \leq |x|$ it is the case that $U^{\Pi,y}(i, 1^t) = x_i$.[2] For all strings $x \in \{0,1\}^*$, we define $\mathrm{KT}(x)$ to be equal to $\mathrm{KT}(x \mid \lambda)$.

▶ **Lemma 8** ([4]). *There is a $c > 0$ such that for all $x \in \{0,1\}^*$ it is the case that* $\mathrm{KT}(x)$ *is at most* $|x| + c \log |x|$.

▶ **Corollary 9.** *There is a $c > 0$ such that for all $x, y \in \{0,1\}^*$ it is the case that* $\mathrm{KT}(x \mid y)$ *is at most* $|x| + c \log |x|$.

2.4 Minimum Conditional KT-complexity Problem, and variants

We give here formal definitions of the computational problems that we will consider in this work. These are the decision and search variants of McKTP.

▶ **Definition 10** (Decision variant). *Let $c > 0$ be as in Corollary 9. Let $n \in \mathbb{N}$ and $m : \mathbb{N} \to \mathbb{N}$. The* Minimum m-Conditional KT-complexity Problem of dimension n (McKT^{m}P of dimension n) *is defined as follows.*

- *Input: Strings $x \in \{0,1\}^n$, $y \in \{0,1\}^{m(n)}$, and a parameter $0 \leq \theta \leq n + c \log n$ in binary.*
- *Question: Is there a program $\Pi \in \{0,1\}^*$ and a run-time bound $t \in \mathbb{N}$ such that $U^{\Pi,y}(i, 1^t) = x_i$ for all $1 \leq i \leq n$, and $|\Pi| + t \leq \theta$?*

The following result is a standard observation.

▶ **Lemma 11.** *For all polynomial-time computable functions $m : \mathbb{N} \to \mathbb{N}$, it is the case that* McKT^{m}P *of dimension n is in* NP.

▶ **Definition 12** (Search variant). *Let $n \in \mathbb{N}$ and $m : \mathbb{N} \to \mathbb{N}$. The* search variant of Minimum m-Conditional KT-complexity Problem of dimension n (Search McKT^{m}P of dimension n) *is defined as follows.*

- *Input: Strings $x \in \{0,1\}^n$ and $y \in \{0,1\}^{m(n)}$.*
- *Output: A program $\Pi \in \{0,1\}^*$ and a run-time bound $t \in \mathbb{N}$ in binary such that $U^{\Pi,y}(i, 1^t) = x_i$ for all $1 \leq i \leq n$, and the sum $|\Pi| + t$ is minimized over the choices of Π and t.*

2.5 One-way functions

In the following, a function μ is said to be *negligible* if for every polynomial p there exists a $n_0 \in \mathbb{N}$ such that for all naturals $n > n_0$ it is the case that $\mu(n) \leq 1/p(n)$.

▶ **Definition 13.** *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. We say that f is a* one-way function (OWF) *if for every PPT algorithm A there exists a negligible function μ such that for all $n \in \mathbb{N}$ it is the case that*

$$\Pr_{x \sim \{0,1\}^n, r}\left[A(1^n, f(x)\,; r) \in f^{-1}(f(x))\right] < \mu(n)$$

where the size of r is equal to the running time of A.

[2] Originally [4], $\mathrm{KT}(x \mid y)$ was defined with the additional requirement that, for $i = |x|+1$, $U^{\Pi,y}\left(i, 1^t\right) = *$. We do not need that additional complication here, although our theorems would also hold using that definition.

We will also employ the following weaker notion of OWFs.

▶ **Definition 14.** *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. We say that f is an* α-weak one-way function (α-weak OWF) *if for every PPT algorithm A and all sufficiently large $n \in \mathbb{N}$ it is the case that*

$$\mathop{\mathbf{Pr}}_{x\sim\{0,1\}^n,r}\left[A(1^n, f(x)\,; r) \in f^{-1}(f(x))\right] < 1 - \alpha(n)$$

where the size of r is equal to the running time of A. We say that f is a weak one-way function (weak OWF) *if there exists some polynomial $q > 0$ such that f is a $(1/q)$-weak OWF.*

Yao [30] proved that the existence of weak OWFs implies the existence of OWFs.

▶ **Theorem 15** ([30]). *Assume that there exists a weak one-way function. Then there exists a one-way function. (Also, if there exists a weak-one-way function computable in logspace, then there is a one-way function computable in logspace.)*

2.6 Average-case hardness/easiness

A *heuristic* H is a PPT algorithm that, on input any $x \in \{0,1\}^n$, outputs a value in $\{0,1\}$ along each computation path.

▶ **Definition 16** (Average-case hardness). *Let $\alpha : \mathbb{N} \to [0,1]$ be a failure parameter function. We say that a function $f : \{0,1\}^n \to \{0,1\}$ is* α-hard-on-average (α-HoA) *if for all heuristics H and all sufficiently large $n \in \mathbb{N}$ it is the case that*

$$\mathop{\mathbf{Pr}}_{x\sim\{0,1\}^n,r}[H(x;r) = f(x)] \leq 1 - \alpha(n)$$

where the size of r is equal to the running time of H.

▶ **Definition 17** (Average-case easiness). *Let $\alpha : \mathbb{N} \to [0,1]$ be a success parameter function. We say that a function $f : \{0,1\}^n \to \{0,1\}$ is* α-easy-on-average (α-EoA) *if f is not $(1-\alpha)$-hard-on-average; that is, if there exists some heuristic H such that for infinitely many $n \in \mathbb{N}$ it is the case that*

$$\mathop{\mathbf{Pr}}_{x\sim\{0,1\}^n,r}[H(x;r) = f(x)] > 1 - (1-\alpha(n)) = \alpha(n)$$

where the size of r is equal to the running time of H.

Let $R \subseteq \{0,1\}^n \times \{0,1\}^*$ be a search problem. A *heuristic* H is a PPT algorithm that, on input any $x \in \{0,1\}^n$, outputs a value in $\{0,1\}^*$ along each computation path.

The notions of average-case hardness and easiness for search problems are defined in a fashion similar to that of decision problems; see Definition 16 and Definition 17.

3 OWFs from average-case hardness of McKTP

In this section, we prove the following result.

▶ **Theorem 18.** *Assume that, for some $m : \mathbb{N} \to \mathbb{N}$, $\mathrm{McKT}^m\mathrm{P}$ of dimension n is $(1/p)$-HoA for some polynomial p. Then, there exists some weak OWF.*

By Theorem 18 and Theorem 15, we get the following corollary.

▶ **Corollary 19** (Theorem 1, restated). *Assume that, for some $m : \mathbb{N} \to \mathbb{N}$, $\mathrm{McKT}^m\mathrm{P}$ of dimension n is $(1/p)$-HoA for some polynomial p. Then, there exists some OWF.*

3.1 Proof of Theorem 18

We will first require the following two lemmas.

▶ **Lemma 20.** *For all functions $m : \mathbb{N} \to \mathbb{N}$, if* McKT^mP *is* $(1/p)$*-HoA for some polynomial* p*, then* Search McKT^mP *is* $\left(1/p^2\right)$*-HoA.*

Proof. We will prove the contrapositive. That is, we will prove that if Search McKT^mP is $\left(1 - 1/p^2\right)$-EoA, then McKT^mP is $(1 - 1/p)$-EoA. In what follows, let $c > 0$ be as in Corollary 9.

Let $N' := n + m(n)$ be the size of the instances of Search McKT^mP of dimension n. Assume that Search McKT^mP is $\left(1 - 1/p^2\right)$-EoA. That is, assume that there exists some heuristic H' that on input a random instance $(x, y) \in \{0,1\}^n \times \{0,1\}^{m(n)}$ outputs with probability greater than $1 - 1/p(N')^2$ a program $\Pi \in \{0,1\}^*$ and a run-time bound $t \in \mathbb{N}$ (in binary) such that $U^{\Pi,y}(i, 1^t) = x_i$ for all $1 \le i \le n$, and the sum $|\Pi| + t$ is minimized over the choices of Π and t.

Given H', a heuristic H for McKT^mP of dimension n and input size $N := n + m(n) + \log(n + c \log n)$, works as follows:

> On input strings $x \in \{0,1\}^n$ and $y \in \{0,1\}^{m(n)}$, and a size parameter $0 \le \theta \le n + c \log n$ in binary, run H' on (x, y) to get a program $\Pi \in \{0,1\}^*$ and a run-time bound $t \in \mathbb{N}$ (in binary). If Π and t are such that $U^{\Pi,y}(i, 1^t) = x_i$ for all $1 \le i \le n$ and $|\Pi| + t \le \theta$, then return YES. Else, return NO.

Note that the running time of H is polynomial in N. The success probability of H over a random instance (x, y, θ) and random bits r is

$$\begin{aligned}
&\Pr_{x,y,\theta,r}[H(x, y, \theta; r) \text{ succeeds}] \\
&\quad \ge \Pr_{x,y,\theta,r}[H(x, y, \theta; r) \text{ succeeds} \mid H'(x, y; r) \text{ succeeds}] \cdot \Pr_{x,y,r}[H'(x, y; r) \text{ succeeds}] \\
&\quad > 1 \cdot \left(1 - \frac{1}{p(N')^2}\right) = 1 - \frac{1}{p(N')^2} \ge 1 - \frac{1}{p(N)},
\end{aligned}$$

since $1/p(N')^2 \le 1/p(N)$ for all sufficiently large $n \in \mathbb{N}$, as desired.

Therefore, McKT^mP is $(1 - 1/p)$-EoA as witnessed by H. ◀

The following is an elaboration on the seminal work by Liu and Pass [20].

▶ **Lemma 21** (Following Liu and Pass [20]). *Assume that, for some function $m : \mathbb{N} \to \mathbb{N}$,* Search McKT^mP *is* $(1/p)$*-HoA for some polynomial p. Then, there exists some weak OWF.*

Proof. Fix some UTM U, and let $c > 0$ be as in Corollary 9. Let $n \in \mathbb{N}$ be sufficiently large and such that Search McKT^mP of dimension n is $(1/p)$-HoA. Consider the function $f : \{0,1\}^* \to \{0,1\}^*$ defined by the mapping rule

$$(s, t, y, \Pi') \mapsto \left(s + t, U^{\Pi,y}\left(1, 1^t\right), \ldots, U^{\Pi,y}\left(n, 1^t\right), y\right),$$

where $m := m(n)$, $y \in \{0,1\}^m$, $\Pi' \in \{0,1\}^{n + c \log n}$ is a program, and $\Pi := \Pi'|_{[s]}$ is the s-bit prefix of Π'. Note that without loss of generality, $s + t \le n + c \log n$, by Corollary 9. This also implies that $s \le n + c \log n$ and $t \le n + c \log n$. For that matter, f is a function from $\{0,1\}^M$ to $\{0,1\}^N$, where $M := 2 \log(n + c \log n) + m + n + c \log n$ and $N := \log(n + c \log n) + n + m$, and is computable in polynomial time.

Observe also that f is only defined over infinitely many input lengths. However, by a padding trick, f can be transformed into another function f' that is defined over all input lengths, and such that f' is a weak one-way function, given that f is [20].

We now claim that if Search McKT^{m}P is $(1/p)$-HoA, then f is a $(1/q)$-weak OWF, where q is a polynomial such that $q(n) := 2\left(n + c\log n\right)^2 n^c p(n+m(n)))^3$ for all $n \in \mathbb{N}$. Towards a contradiction, assume that there exists a PPT algorithm A that inverts f with probability at least $1 - 1/q(M) \geq 1 - 1/q(n)$.

First, note that except for a fraction $1/\left(2p(n+m)\right)$ of sequences of random bits r for A, the deterministic machine A_r, given by $A_r(f(z)) := A(f(z)\,;r)$ for all $z \in \{0,1\}^M$, fails to invert f with probability at most $2p(n+m)\,/q(n)$ over a uniformly random input z. This is so, as

$$\begin{aligned}
&\mathop{\mathbf{Pr}}_{r}\left[\mathop{\mathbf{Pr}}_{z}[A_r(f(z)) \text{ fails}] > \frac{2p(n+m)}{q(n)}\right] \\
&\quad \leq \mathop{\mathbf{Pr}}_{r}\left[\mathop{\mathbf{Pr}}_{z}[A_r(f(z)) \text{ fails}] \geq 2p(n+m)\cdot \mathop{\mathbf{Pr}}_{z,r}[A_r(f(z)) \text{ fails}]\right] \\
&\quad = \mathop{\mathbf{Pr}}_{r}\left[\mathop{\mathbf{Pr}}_{z}[A(f(z)\,;r) \text{ fails}] \geq 2p(n+m)\cdot \mathop{\mathbf{E}}_{r}\left[\mathop{\mathbf{Pr}}_{z}[A(f(z)\,;r) \text{ fails}]\right]\right] \leq \frac{1}{2p(n+m)},
\end{aligned}$$

by Lemma 7. Henceforth, we will call such a sequence of random bits *good*; otherwise, we will call a sequence of random bits *bad*. Therefore, we have

$$\mathop{\mathbf{Pr}}_{z,r}[A(f(z)\,;r) \text{ fails} \mid r \text{ is good}] = \mathop{\mathbf{Pr}}_{z,r}[A_r(f(z)) \text{ fails} \mid r \text{ is good}] \leq \frac{2p(n+m)}{q(n)}.$$

We propose the following heuristic H for Search McKT^{m}P:

> On input strings $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$, and using random bits r, the algorithm H runs $A(j,x,y;r)$ for all $j \in [n + c\log n]$. For each $j \in [n+c\log n]$, $A(j,x,y;r)$ returns a tuple $\left(s_j, t_j, y, \Pi'_j\right)$. Then, $H(x,y;r)$ returns a program $\Pi'_k|_{[s_k]}$ from $\left\{\Pi'_j|_{[s_j]}\right\}_{j\in[n+c\log n]}$ such that $U^{\Pi'_k|_{[s_k]},y}(i,1^{t_k}) = x_i$ for all $1 \leq i \leq n$, and $\left|\Pi'_k|_{[s_k]}\right| + t_k = s_k + t_k$ is minimized.

We will now analyze the average-case performance of H. Fix a good sequence of random bits r, as defined above, and recall that, in this case, $\mathbf{Pr}_z[A_r(f(z)) \text{ fails}] \leq 2p(n+m)\,/q(n)$. Let S_r be the set of inputs (x,y) for which $H(x,y;r)$ fails, when given random bits r. Observe that, for any good r,

$$\mathop{\mathbf{Pr}}_{x,y}[H(x,y;r) \text{ fails}] = \frac{|S_r|}{2^{n+m}}.$$

Consider $(x,y) \in S_r$ and let $w_{x,y} := \mathrm{KT}(x \mid y)$ be the conditional KT-complexity of x given y. By Corollary 9, we have $w_{x,y} \leq n + c\log n$. If $H(x,y;r)$ fails, then it means that A fails to invert $(w_{x,y}, x, y)$ when given the good sequence of random bits r.

Recall that $\mathbf{Pr}_z[A_r(f(z)) \text{ fails}] \leq 2p(m\,(n+1))\,/q(n)$. Recall also, from the definition of f, and from the fact that $w_{x,y} \leq n + c\log n$, that

$$\mathop{\mathbf{Pr}}_{z}[f(z) = (w_{x,y}, x, y)] \geq \frac{1}{(n+c\log n)^2 \cdot 2^m \cdot 2^{n+c\log n}}.$$

Thus, for any good sequence r, we have

$$\frac{2p(n+m)}{q(n)} \geq \mathop{\mathbf{Pr}}_{z}[A_r\left(f(z)\right) \text{ fails}]$$

$$
\begin{aligned}
&= \sum_{(w,x,y):A_r(w,x,y)\text{ fails}} \Pr_z[f(z)=(w,x,y)] \\
&\geq \sum_{(x,y):A_r(w_{x,y},x,y)\text{ fails}} \Pr_z[f(z)=(w_{x,y},x,y)] \\
&\geq \sum_{(x,y)\in S_r} \frac{1}{(n+c\log n)^2\cdot 2^m\cdot 2^{n+c\log n}} \\
&= \frac{|S_r|}{2^{n+m}}\cdot\frac{1}{(n+c\log n)^2\, 2^{c\log n}} = \frac{\mathbf{Pr}_{x,y}[H(x,y;r)\text{ fails}]}{(n+c\log n)^2\, n^c}.
\end{aligned}
$$

Since this holds for any good sequence r, we have that

$$
\begin{aligned}
\Pr_{x,y,r}[H(x,y;r)\text{ fails} \mid r\text{ is good}] &\leq \frac{(n+c\log n)^2\, n^c 2p(n+m)}{q(n)} \\
&= \frac{(n+c\log n)^2\, n^c 2p(n+m)}{2\,(n+c\log n)^2\, n^c p(n+m)^3} \\
&= \frac{1}{p(n+m)^2} < \frac{1}{2p(n+m)},
\end{aligned}
$$

since $p(n+m)>2$ for all sufficiently large $n\in\mathbb{N}$. Therefore, H fails with probability at most

$$
\Pr_{x,y,r}[H(x,y;r)\text{ fails} \mid r\text{ is good}] + \Pr_r[r\text{ is bad}] < \frac{1}{2p(n+m)} + \frac{1}{2p(n+m)} = \frac{1}{p(n+m)}.
$$

This yields a contradiction. ◀

We now turn to the proof of Theorem 18.

Proof of Theorem 18. Immediate; by Lemma 20 and Lemma 21, since if p is a polynomial, then p^2 is a polynomial too. ◀

4 Logspace-computable OWFs from average-case hardness of McKTP

Now we show that, applying the insights of Ren and Santhanam [28], we can strengthen the theorems of the preceding section. We show the following.

▶ **Theorem 22.** *Assume that, for some $m:\mathbb{N}\to\mathbb{N}$, McKT^{m}P of dimension n is $(1/p)$-HoA for some polynomial p. Then, there exists some weak OWF computable in logspace.*

Proof sketch. Modify the definition of f from the proof of Lemma 21, so that now f is

$$
(s,t,y,\Pi') \mapsto \left(s+t, U^{\Pi,y}(1,1^t),\ldots,U^{\Pi,y}(n,1^t),y\right),
$$

where $m := m(n)$, $y\in\{0,1\}^m$, $\Pi'\in\{0,1\}^{n+c\log n}$ is a program, $\Pi := \Pi'|_{[s]}$ is the s-bit prefix of Π', and $t\leq d\log n$ for some d. This function f is clearly computable in logspace.

Significantly, Ren and Santhanam [28, Theorem 4.1] show that, if the search version of KT is hard-on-average, then a function very similar to f is a weak one-way function. Essentially identical considerations allow us to conclude that, if Search McKT^{m}P is $(1/p)$-HoA for some polynomial p, then f is a weak one-way function. The main point is that, for every y, most strings x have the property that, when $|\Pi|+t$ is minimized (where U uses description Π and run-time t to compute the bits of x), $t=O(\log n)$. The rest of the analysis is very similar to that of Lemma 21. ◀

By Theorem 22 and Theorem 15, we get the following corollary.

▶ **Corollary 23** (Theorem 3, restated). *Assume that, for some $m : \mathbb{N} \to \mathbb{N}$, $\mathrm{McKT}^m\mathrm{P}$ of dimension n is $(1/p)$-HoA for some polynomial p. Then, there exists some logspace-computable OWF.*

5 Average-case hardness of McKTP from logspace-computable OWFs: Proof of Theorem 5

Again, we appeal to the techniques of Ren and Santhanam. Ren and Santhanam [28, Theorem 4.4] show that, if there is a one-way function computable in logspace, then the problem of computing an approximation to KT complexity is hard-on-average. A nearly-identical proof shows that computing $\mathrm{KT}(x \mid y)$ is HoA. Essentially the only modification that needs to be made to the proof of [28, Theorem 4.4] arises in the proof of their Lemma 4.7, which establishes that computing KT is HoA under a condition that holds if there is a logspace-computable OWF. The proof of [28, Lemma 4.7] relies on the fact that the output of a certain pseudorandom generator has small KT complexity, whereas a random string has high KT complexity. But the output z of this generator also has small $\mathrm{KT}(z \mid y)$ for every y, whereas a random string z has $\mathrm{KT}(z \mid y)$ large for almost every y. Thus a very similar analysis shows that computing $\mathrm{KT}(x \mid y)$ is HoA, which in turn (via Lemma 20) implies that $\mathrm{McKT}^m\mathrm{P}$ is HoA.

References

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

2 Adi Akavia, Oded Goldreich, Shafi Goldwasser, and Dana Moshkovitz. On basing one-way functions on NP-hardness. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 701–710. ACM, 2006. See also [3].

3 Adi Akavia, Oded Goldreich, Shafi Goldwasser, and Dana Moshkovitz. Erratum for: On basing one-way functions on NP-hardness. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 795–796. ACM, 2010.

4 Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM J. Comput.*, 35(6):1467–1493, 2006.

5 Eric Allender, Mahdi Cheraghchi, Dimitrios Myrisiotis, Harsha Tirumala, and Ilya Volkovich. One-way functions and a conditional variant of MKTP. *Electron. Colloquium Comput. Complex.*, 28:9, 2021.

6 Eric Allender, Mahdi Cheraghchi, Dimitrios Myrisiotis, Harsha Tirumala, and Ilya Volkovich. One-way functions and Partial MCSP. *Electron. Colloquium Comput. Complex.*, 28:9, 2021.

7 Andrej Bogdanov and Luca Trevisan. Average-case complexity. *Found. Trends Theor. Comput. Sci.*, 2(1), 2006.

8 Andrej Bogdanov and Luca Trevisan. On worst-case to average-case reductions for NP problems. *SIAM J. Comput.*, 36(4):1119–1159, 2006.

9 Chris Brzuska and Geoffroy Couteau. Towards fine-grained one-way functions from strong average-case hardness. *IACR Cryptol. ePrint Arch.*, 2020:1326, 2020.

10 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing (STOC)*, pages 624–633. ACM, 2014.

11 Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

12 Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

13 Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. In *59th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 247–258. IEEE Computer Society, 2018.

14 Shuichi Hirahara and Rahul Santhanam. On the average-case complexity of MCSP and its variants. In Ryan O'Donnell, editor, *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, volume 79 of *LIPIcs*, pages 7:1–7:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

15 Rahul Ilango. Approaching MCSP from above and below: Hardness for a conditional variant and $\mathsf{AC}^0[p]$. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 34:1–34:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

16 Rahul Ilango, Hanlin Ren, and Rahul Santhanam. Hardness on any samplable distribution suffices: New characterizations of one-way functions by meta-complexity. *Electron. Colloquium Comput. Complex.*, 28:82, 2021.

17 Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*, pages 134–147. IEEE Computer Society, 1995.

18 Russell Impagliazzo and Leonid A. Levin. No better ways to generate hard NP instances than picking uniformly at random. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 812–821. IEEE Computer Society, 1990.

19 Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptol.*, 9(4):199–216, 1996.

20 Yanyi Liu and Rafael Pass. On one-way functions and Kolmogorov complexity. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1243–1254. IEEE, 2020.

21 Yanyi Liu and Rafael Pass. Cryptography from sublinear-time average-case hardness of time-bounded Kolmogorov complexity. In *Proceedings of the 53rd ACM Symposium on Theory of Computing (STOC)*. ACM, 2021.

22 Yanyi Liu and Rafael Pass. A note on one-way functions and sparse languages. *IACR Cryptol. ePrint Arch.*, 2021:890, 2021.

23 Yanyi Liu and Rafael Pass. On one-way functions from NP-complete problems. *Electron. Colloquium Comput. Complex.*, 28:59, 2021.

24 Yanyi Liu and Rafael Pass. On the possibility of basing cryptography on $\mathsf{EXP} \neq \mathsf{BPP}$. *Electron. Colloquium Comput. Complex.*, 28:56, 2021.

25 Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007. doi:10.1137/S0097539705447360.

26 Mikito Nanashima. On basing auxiliary-input cryptography on NP-hardness via nonadaptive black-box reductions. In *12th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 185 of *LIPIcs*, pages 29:1–29:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

27 Ján Pich. Learning algorithms from circuit lower bounds. *CoRR*, abs/2012.14095, 2020. arXiv:2012.14095.

28 Hanlin Ren and Rahul Santhanam. Hardness of KT characterizes parallel cryptography. *Electron. Colloquium Comput. Complex.*, 28:57, 2021.

29 Rahul Santhanam. Pseudorandomness and the Minimum Circuit Size Problem. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151 of *LIPIcs*, pages 68:1–68:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

30 Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 80–91. IEEE Computer Society, 1982.

A Hard-on-average problems in NP

We first introduce some useful notation. For a language $L \subseteq \{0,1\}^*$ we define its *characteristic function*, namely $f_L : \{0,1\}^* \to \{0,1\}$, to be a function given by

$$f_L(x) := \begin{cases} 1, & \text{if } x \in L, \\ 0, & \text{otherwise} \end{cases}$$

for all $x \in \{0,1\}^*$.

For sets $K, L \subseteq \{0,1\}^*$, the *disjoint union of* K *and* L, denoted $K \uplus L$, is the set $\{0x \mid x \in K\} \cup \{1x \mid x \in L\}$.

For a failure parameter function $\alpha : \mathbb{N} \to [0,1]$, we say that a language L is *α-hard-on-average (α-HoA)* if its characteristic function f_L is α-HoA. Similarly we define average-case easiness for languages.

We prove the following.

▶ **Proposition 24.** *Let L be a language in* NP *that is α-HoA for some failure parameter function $\alpha : \mathbb{N} \to [0,1]$. Then, the language $L^* := L \uplus \mathrm{SAT}$ is* NP*-complete and α^*-HoA, where $\alpha^* : \mathbb{N} \to [0,1]$ is a failure parameter function such that $\alpha^*(n) := \alpha(n-1) - 1/2$ for all naturals $n \geq 2$.*

Before we prove Proposition 24, we recount the following basic observation.

▶ **Lemma 25.** NP *is closed under disjoint union.*

We now turn to the proof of Proposition 24.

Proof of Proposition 24. By Lemma 25, the language L^* is in NP since L^* is the disjoint union of $L \in \mathsf{NP}$ and $\mathrm{SAT} \in \mathsf{NP}$.

We will now show that L^* is NP-hard, by giving a polynomial-time reduction R from SAT to L^*. For all $x \in \{0,1\}^*$, let $R(x) := 1x \in \{0,1\}^*$. We see that R is polynomial-time computable. Moreover, if $x \in \mathrm{SAT}$, then $R(x) = 1x \in L^*$, and if $R(x) \in L^*$, then $1x \in L^*$ and so $x \in \mathrm{SAT}$.

What is left is to prove that L^* is α^*-HoA, where $\alpha^* : \mathbb{N} \to [0,1]$ is such that $\alpha^*(n) := \alpha(n-1) - 1/2$ for all naturals $n \geq 2$. Towards a contradiction, assume that L^* is $(1-\alpha^*)$-EoA and let H^* be a heuristic that witnesses this phenomenon. We will give a heuristic H that witnesses the fact that L is $(1-\alpha)$-EoA, whereby establishing the desired contradiction. To this end, let

$$H(x) := H^*(0x)$$

for all $x \in \{0,1\}^*$. We will show that H has the desired average-case performance. Indeed,

$$\begin{aligned}
\Pr_{x \sim \{0,1\}^n}[H(x) = f_L(x)] &= \Pr_{x \sim \{0,1\}^n}[H^*(0x) = f_{L^*}(0x)] \\
&= \Pr_{y \sim \{0,1\}^{n+1}}[H^*(y) = f_{L^*}(y) \mid y_1 = 0] \\
&\geq \Pr_{y \sim \{0,1\}^{n+1}}[H^*(y) = f_{L^*}(y)] - \Pr_{y \sim \{0,1\}^{n+1}}[y_1 = 1] \\
&\geq 1 - \alpha^*(n+1) - \frac{1}{2} \\
&= 1 - \left(\alpha((n+1)-1) - \frac{1}{2}\right) - \frac{1}{2} \\
&= 1 - \alpha(n)\,.
\end{aligned}$$

◀

B McKTP is NP-complete under randomized reductions

In this section, we prove Theorem 2 by adapting Ilango's work [15].

B.1 Set Cover

We first fix some notation about Set Cover.

▶ **Definition 26.** *The* Set Cover *problem is defined as follows.*

- *Input: A tuple $(n, S_1, \ldots, S_t)$ in binary, where $n \in \mathbb{N}$ and $S_1, \ldots, S_t \subseteq [n]$ are sets such that $[n] \subseteq \bigcup_{i=1}^{t} S_i$.*
- *Output: The value of*

$$\min_{I \subseteq [t]} \left\{ |I| \mid [n] \subseteq \bigcup_{i \in I} S_i \right\}.$$

Dinur and Steurer [10] show that it is NP-hard to approximate Set Cover.

▶ **Theorem 27** ([10])**.** *It is* NP*-hard to approximate Set Cover by a factor of at most $(1 - o(1)) \ln n$.*

B.2 Approximation algorithms

In the following, we will adopt the following notion of an approximation algorithm.

▶ **Definition 28.** *Let Π be an optimization problem. For all instances $I \in \{0,1\}^*$ of Π, let the optimal solution of I be denoted by $\mathrm{OPT}(I) \in \mathbb{R}$. Let $\alpha > 0$. We say that a probabilistic algorithm A* approximates Π by a factor of α *if, for all instances I of Π, it is the case that*

$$\mathrm{OPT}(I) < A(I) \leq \alpha \cdot \mathrm{OPT}(I)$$

with probability at least $1 - o(1)$ over the internal randomness of A.

B.3 Proof of Theorem 2

For a string b of length m and a set $R \subseteq [m]$, let $b_{\langle R \rangle}$ be the string of length m where

$$b_{\langle R \rangle}(j) := \begin{cases} b(j)\,, & \text{if } j \in R, \\ 0, & \text{otherwise} \end{cases}$$

for all $1 \leq j \leq m$. Equivalently,

$$b_{\langle R \rangle}(j) := b(j) \wedge \mathbb{1}_{j \in R}$$

for all $j \in [m]$.

Next, we define a uniformly random partition $\mathcal{P} = (P_1, \ldots, P_n)$ of $[m]$ into n parts to be such that each element $i \in [m]$ is put into P_j where $j \in [n]$ is chosen uniformly at random. It will be also useful to think of $\mathcal{P}$ as a uniformly random function $P : [m] \to [n]$.

For a partition $\mathcal{P} = (P_1, \ldots, P_n)$ of $[m]$ and any set $S \subseteq [n]$, we define the *$\mathcal{P}$-lift of S*, denoted $S^{\mathcal{P}}$, to be the set

$$S^{\mathcal{P}} := \bigcup_{i \in S} P_i.$$

Following Ilango [15], we show that McKTP can be used to approximate Set Cover.

▶ **Lemma 29** (Following Ilango [15]). *Let $S_1, \ldots, S_t \subseteq [n]$ be sets that cover $[n]$. Let b be a string of length $m \geq (nt)^5$ and let $\mathcal{P} = (P_1, \ldots, P_n)$ be a uniformly random partition of $[m]$ into n parts. Define the oracle $O : \{0,1\}^{\log t} \times \{0,1\}^{\log m} \to \{0,1\}$ to be such that*

$$O(i,z) := \begin{cases} b_{\langle S_i^{\mathcal{P}} \rangle}(z)\,, & \text{if } i \in [t], \\ 0, & \text{otherwise}, \end{cases}$$

for all $i \in [t]$ and $z \in [m]$. Let y be the truth table of O, and note that $|y| = mt$. Let ℓ be the size of an optimal cover of $[n]$ by $S_1, \ldots, S_t$. Then, we have that

1. $\mathrm{KT}(b \mid y) \leq 200\ell\,(\log t + \log m)$ *and*
2. $\mathrm{KT}(b \mid y) > \ell\,(\log t + \log m)\,/2$ *with high probability over the choice of b.*

Proof. We prove each item of Lemma 29 separately.

▷ Claim 30. It is the case that $\mathrm{KT}(b \mid y) \leq 200\ell\,(\log t + \log m)$.

Proof. Assume that an optimal set cover of size ℓ is realized by the sets $S_{i_1}, \ldots, S_{i_\ell}$. Fix some UTM U that has oracle access to y. Let $\Pi \in \{0,1\}^*$ be a program that contains in its description encodings of $i_1, \ldots, i_\ell \in \{0,1\}^t$ and operates as follows:

On input $x \in \{0,1\}^{\log m}$, compute and output $y_{(i_1,x)} \vee \cdots \vee y_{(i_\ell,x)}$.

Note that $|\Pi| \leq (\ell+2)\log t + O(1) \leq 100\ell \log t$. In what follows, let $T \in \mathbb{N}$ be a sufficiently large run-time bound such that

$$\begin{aligned} U^{\Pi,y}\left(x,1^T\right) &:= y_{(i_1,x)} \vee \cdots \vee y_{(i_\ell,x)} \\ &= O(i_1,x) \vee \cdots \vee O(i_\ell,x) = \bigvee_{i\in[\ell]} \bigvee_{j \in S_i} b_{\langle P_j \rangle}(x) = \bigvee_{j\in[n]} b_{P_j}(x) = b(x)\,, \end{aligned}$$

for all $x \in \{0,1\}^{\log m}$. Note that $T \leq 100\ell\,(\log t + \log m)$. Therefore, we have that $\mathrm{KT}(b \mid y) \leq 200\ell\,(\log t + \log m)$. ◁

We now turn to the lower bound. We do this by a union bound argument. Fix some oracle program $M^y(\cdot) := U^{\Pi,y}\left(\cdot,1^T\right)$ of program Π that uses oracle y and runs in time T such that $|\Pi| + T \leq \ell\,(\log t + \log m)\,/2$. Then, as each oracle query requires time $\log t + \log m$, we can deduce that M makes at most $\ell/2 \leq n/2 \leq n$ oracle queries to y.

We will show that

$$\Pr_{b,\mathcal{P}}[M^y \text{ computes } b \text{ in time } T, \text{ and } |\Pi| + T \leq \ell\,(\log t + \log m)\,/2]$$

is exponentially small. We do this by finding a long sequence of inputs $x_1, \ldots, x_d$ on which M has not too large a chance of computing b.

We construct this list recursively, as follows. Let $x_1 := 0^{\log m}$, and let

$$Q_1 := \left\{x \in \{0,1\}^{\log m} \mid M^y(x_1) \text{ makes a query } (i,x) \text{ to } y, \text{ for some } i \in [t]\right\}.$$

Now, for $j \geq 1$, if $\{0,1\}^{\log m} \setminus Q_j$ is non-empty, then let x_{j+1} be an element of $\{0,1\}^{\log m} \setminus Q_j$, and let

$$Q_{j+1} := Q_j \cup \left\{x \in \{0,1\}^{\log m} \mid M^y(x_{j+1}) \text{ makes a query } (i,x) \text{ to } y, \text{ for some } i \in [t]\right\}.$$

If $\{0,1\}^{\log m} = Q_j$, then terminate the sequence. Since M makes at most n queries to y, we know that $|Q_j| \leq jn$. Thus, since $|Q_d| = \left|\{0,1\}^{\log m}\right| = m$ the length of this sequence is at least m/n. That is, $d \geq m/n$.

It remains to bound the probability

$$\mathbf{Pr}[\text{for all } j \in [d],\ M^y(x_j) = b(x_j)] = \prod_{j=1}^{d} \mathbf{Pr}\left[M^y(x_j) = b(x_j) \mid \bigwedge_{k \in [j-1]} M^y(x_k) = b(x_k)\right].$$

Fix some $j \in [d]$. We will bound

$$\mathbf{Pr}\left[M^y(x_j) = b(x_j) \mid \bigwedge_{k \in [j-1]} M^y(x_k) = b(x_k)\right].$$

Let $E := \bigwedge_{k \in [j-1]} M^y(x_k) = b(x_k)$ be the event that we are conditioning on.

▷ Claim 31. It is the case that

$$\mathbf{Pr}[M^y(x_j) = b(x_j) \mid E] \leq 1 - \frac{1}{2n}.$$

Proof. By construction of the sequence $x_1, \ldots, x_d$, we know that on all the inputs $x_1, \ldots, x_{j-1}$, the program M^y does not make an oracle call of the form (i, x_j) for any i. Thus, the only time the value of O depends on $b(x_j)$ and $P(x_j)$ is on inputs of the form (i, x_j) for some i, and since $b(x_j)$ and $P(x_j)$ are chosen independently at random, we know that $b(x_j)$ and $P(x_j)$ are still uniform random variables conditioned on E. That is,

$$\mathbf{Pr}[b(x_j) = 1 \mid E] = \frac{1}{2}$$

and

$$\mathbf{Pr}[P(x_j) = r \mid E] = \frac{1}{n}$$

for all $r \in [n]$.

Now, define O' as

$$O'(i,x) := \begin{cases} 0, & \text{if } x = x_j, \\ O(i,x), & \text{otherwise,} \end{cases}$$

and let y' be the truth table of O'. Let also $i_1, \ldots, i_v$ with $v \leq \ell/2$ be such that, using the modified oracle O', they are the only oracle queries $M^{y'}(x_j)$ makes that have x_j as the 2nd component of the query, so the queries are $(i_1, x_j), \ldots, (i_v, x_j)$. Since $v < \ell$ there exists an element r^* that is not in $S_{i_1} \cup \cdots \cup S_{i_v}$.

Moreover, observe that if $P(x_j) = r^*$, then $M^y(x_j)$ will actually make the same oracle queries (and get the same zero responses) as the modified oracle program $M^{y'}$. In this case, since $P(x_j) = r^*$ is not in $S_{i_1} \cup \cdots \cup S_{i_v}$, it follows that

$$O(i_1, x_j) = \cdots = O(i_v, x_j) = 0$$

regardless of the value of $b(x_j)$. Thus, the output of M^y on input x does not depend at all on the value of $b(x)$ if $P(x_j) = r^*$. Hence, the probability it correctly guesses $M^y(x) = b(x)$ is at most half when $P(x_j) = r^*$.

Since $P(x_j)$ is chosen uniformly at random, we have that $P(x_j) = r^*$ with probability $1/n$. Therefore,

$$\mathbf{Pr}[M^y(x_j) = b(x_j) \mid E] \le 1 - \frac{1}{2n}$$

and the proof os complete. ◁

Using Claim 31, we have

$$\prod_{j=1}^{d} \mathbf{Pr}\left[M^y(x_j) = b(x_j) \mid \bigwedge_{k \in [j-1]} M^y(x_k) = b(x_k) \right] \le \left(1 - \frac{1}{2n}\right)^d$$
$$\le e^{-d/(2n)} \le e^{-m/\left(2n^2\right)} \le e^{-n^3 t^5/2}.$$

On the other hand the number of oracle programs of size at most $\ell\left(\log t + \log m\right)/2 \le O(nt\log n)$ is at most $2^{O\left(n^2 t\right)}$. Thus, by a union bound, the probability that there exists an oracle program Π that computes any bit of b in time T, whereby $|\Pi| + T \le \ell\left(\log t + \log m\right)/2$, is $o(1)$ as desired. ◀

Lemma 29 implies the following corollary.

▶ **Corollary 32.** *There is a polynomial-time computable function $M : \mathbb{N} \to \mathbb{N}$ such that the following hold. Given a Set Cover instance $I := (n, S_1, \ldots, S_t)$, a random b of length $N \ge (nt)^5$ and a random partition P of $[N]$ into n parts, if one constructs a string y as in Lemma 29, whereby $|y| \le M(N)$, then* $\mathrm{KT}(b \mid y)$ *approximates Set Cover by a factor of* 400 *according to Definition 28. That is, if ℓ is the size of an optimal set cover of I and $c := \log N + \log t$, then it is the case that with probability* 1

$$\frac{2}{c} \cdot \mathrm{KT}(b \mid y) \le 400\ell,$$

and with probability $1 - o(1)$

$$\frac{2}{c} \cdot \mathrm{KT}(b \mid y) > \ell.$$

Proof. Let $y \in \{0,1\}^*$, $n \in \mathbb{N}$, and $t \in \mathbb{N}$ be as in Lemma 29. Let $\gamma := 1/2$. Then, $\mathrm{McKT}^M\mathrm{P}$ of dimension $N := |b| \ge (nt)^5$ and $M := N^{1+\gamma} = N^{1+1/2} = N \cdot N^{1/2} \ge Nt = |y|$ is such that Lemma 29 immediately implies that

$$\ell < \frac{2}{c} \cdot \mathrm{KT}(b \mid y) \le 400\ell,$$

where the first inequality holds with probability $1 - o(1)$ and the second one holds with probability 1. ◀

Theorem 27 and Corollary 32 yield the following corollary.

▶ **Corollary 33.** *There exists a polynomial-time computable function $m : \mathbb{N} \to \mathbb{N}$ such that* $\mathrm{McKT}^m\mathrm{P}$ *is* NP*-hard under polynomial-time randomized reductions.*

Finally, by combining Lemma 11 and Corollary 33 we get a proof of Theorem 2.

▶ **Corollary 34** (Theorem 2, restated). *There exists a polynomial-time computable function $m : \mathbb{N} \to \mathbb{N}$ such that* $\mathrm{McKT}^m\mathrm{P}$ *is* NP*-complete under polynomial-time randomized reductions.*

Generalizations of Length Limited Huffman Coding for Hierarchical Memory Settings

Shashwat Banchhor
Department of Computer Science, Indian Institute of Technology, New Delhi, India

Rishikesh Gajjala
Indian Institute of Science, Bangalore, India
Department of Computer Science, Indian Institute of Technology, New Delhi, India

Yogish Sabharwal
IBM Research, New Delhi, India

Sandeep Sen[1]
Department of Computer Science, Shiv Nadar University, Uttar Pradesh, India

Abstract

In this paper, we study the problem of designing prefix-free encoding schemes having minimum average code length that can be decoded efficiently under a decode cost model that captures memory hierarchy induced cost functions. We also study a special case of this problem that is closely related to the length limited Huffman coding (LLHC) problem; we call this the *soft-length limited Huffman coding* problem. In this version, there is a penalty associated with each of the n characters of the alphabet whose encodings exceed a specified bound $D(\leq n)$ where the penalty increases linearly with the length of the encoding beyond D. The goal of the problem is to find a prefix-free encoding having minimum average code length and total penalty within a pre-specified bound $\mathcal{P}$. This generalizes the LLHC problem. We present an algorithm to solve this problem that runs in time $O(nD)$. We study a further generalization in which the penalty function and the objective function can both be arbitrary monotonically non-decreasing functions of the codeword length. We provide dynamic programming based exact and PTAS algorithms for this setting.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases Approximation algorithms, Hierarchical memory, Prefix free codes

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.8

Related Version Preliminary versions of this work appeared as posters in Data compression conference 2020 [5] and Data compression conference 2021 [6].
Previous Version: `https://ieeexplore.ieee.org/document/9105878`
Previous Version: `https://ieeexplore.ieee.org/document/9418722`

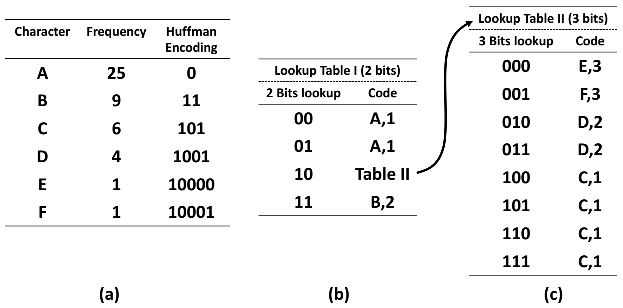
Acknowledgements We are grateful to an anonymous reviewer for suggesting the use of Monge property and simplifying the proofs of Theorem 1 and Theorem 2 in a previous version of this manuscript.

1 Introduction

Data compression algorithms aim to reduce the number of bits required to represent data in order to save storage capacity, speed up file transfer, and decrease costs for storage hardware and network bandwidth. Compression techniques are primarily divided into two categories: lossless and lossy. Lossless compression enables data to be restored to its original state, without the loss of a single bit of data, when it is uncompressed (decoded). Huffman encoding is a basic and popular approach for lossless data compression based on variable length prefix-free encoding [20], where the characters of the alphabet are encoded with variable

[1] Currently on leave from Dept. of Comp. Science, Indian Institute of Technology, New Delhi, India

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 8; pp. 8:1–8:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Character	Frequency	Huffman Encoding
A	25	0
B	9	11
C	6	101
D	4	1001
E	1	10000
F	1	10001

(a)

Lookup Table I (2 bits)	
2 Bits lookup	Code
00	A,1
01	A,1
10	Table II
11	B,2

(b)

Lookup Table II (3 bits)	
3 Bits lookup	Code
000	E,3
001	F,3
010	D,2
011	D,2
100	C,1
101	C,1
110	C,1
111	C,1

(c)

Figure 1 Consider an alphabet with frequencies and Huffman encoding as shown in Table (a). Tables (b) and (c) illustrate the 1^{st} and 2^{nd} level lookup tables of width 2 and 3 bits respectively.

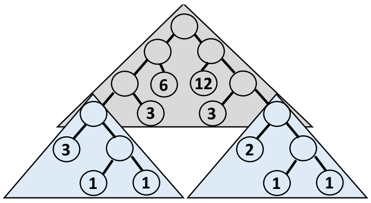


Figure 2 Illustration of blocking scheme: $<(3,1),(2,1)>$.

length codewords and no character encoding is a prefix of another. Huffman encoding is widely used in many applications including file compression (e.g. GZIP [12], PKZIP [12], BZIP2 [10], etc.) and image and video storage formats (JPEG [28], PNG [7], MP3 [8], etc.).

Traversal of a Huffman tree to decode compressed data has an inherent cost proportional to the path length that can be prohibitively slow for many real time applications. One such application is inference task in deep learning. As the sizes of deep learning models are quite large, smaller models are obtained by using Huffman coding in conjunction with other techniques to reduce the memory consumption [18]. The model is decoded in real-time when inference has to be performed. In such settings, it is acceptable to trade-off the compression ratio for improved decode time as this is a critical aspect for a good user experience. Since the data is encoded only once, it may be beneficial to spend the extra time in suitably encoding data to expedite decoding.

To avoid repeated sequential path traversals of the Huffman tree, we can exploit the indirect addressing capabilities of the RAM model by using lookup tables; code trees are employed where small tables are used to represent subtrees [26]. If w bits are used (called the width of the table), then the size of the table is 2^w. So we partition the code into prefixes of smaller lengths when the tree is not balanced, to economize space. If a prefix of the lookup bits forms a valid code word, then the table entry points to the corresponding code word and the input slides ahead by the number of bits used in the encoding of the code word. Otherwise, the table entry points to another table where a lookup is performed with the next fixed number of bits (possibly different than w) of the input; this is repeated until a valid word is decoded. This is illustrated in Figure 1.

This scheme is further complicated by the memory hierarchy that limits the storage at the faster levels of memory and has increasing latencies as we access deeper tables. The prefix tree can be viewed as multiple levels of blocks where each block corresponds to a lookup table used during decoding. Figure 2 demonstrates the concept of blocking where we assume that the blocks that require the same number of indirections have similar latencies. This problem can be formulated as follows:

Consider an alphabet C such that the size of alphabet, $|C| = n$. For each character c in C, let the attribute $freq(c)$ denote the frequency of c in the input data to be encoded. Given a prefix tree T corresponding to a prefix-free code for C, let $d_T(c)$ denote the depth of the leaf corresponding to the encoding of c in the tree. Note that $d_T(c)$ is also the length of the codeword for character c. The average code length of the encoding represented by the tree T is given by

$$len(T) = \sum_{c \in C} freq(c) \cdot d_T(c) \tag{1}$$

Define a *blocking scheme* of m *block levels* as a sequence of m block parameters, $<(w_1, q_1)$, (w_2, q_2), $\ldots$,$(w_m, q_m)>$, where w_j and q_j specify the width and the access cost of a block, respectively, at *block level* j in the tree. For a blocking scheme, the number of memory hierarchies is the number of times the access cost changes when traversing the blocks in order. For a character c having depth $d_T(c)$ in a prefix tree T, the cost of looking up (decoding) the character under the scheme BS, $\delta_T(c)$, is given by the total sum of the cost of accessing the blocks starting from the first *block level* up to the *block level* to which the character belongs, i.e., $\delta_T(c) = \sum_{i \leq W(c)} q_i$ where $W(c) = \arg\min_h \left\{ \sum_{j=1}^{h} w_j \geq d_T(c) \right\}$. The total decode time of the encoding for a prefix tree T is given by: $\delta(T) = \sum_{c \in C} freq(c) \cdot \delta_T(c)$.

> **Problem Definition (COPT):** Given a blocking scheme BS and parameter Δ, called the *permitted cost*, the goal of our problem is to determine a prefix tree, T, that minimizes the average code length, $len(T)$, subject to $\delta(T) \leq \Delta$. We call this the *code optimal prefix tree problem* and denote it by COPT(Δ). With slight abuse of notation, we shall also refer to the decode time of the associated solution as COPT(Δ).

We present an exact and a PTAS algorithm for the $COPT(\Delta)$ problem:

▶ **Theorem 1.**

(a) *There exists a dynamic programming based algorithm to solve the $COPT(\Delta)$ problem that runs in time $O(n^{2+m})$ for m block levels.*

(b) *For the case where the number of block levels, m, is a constant, there exists an algorithm that returns a prefix tree having code-length $\leq (1+\epsilon)COPT(\Delta)$. The running time of the algorithm is $O\left(\frac{n^2}{\epsilon} \max\left(\frac{1}{\epsilon^2}, \log^2(n)\right)\right)$.*

Another technique for optimizing the decode time that is popular in practice was proposed by Moffat and Turpin [26]. Their algorithm looks up one entry of an *offset* array (sequentially) for every bit of the compressed data read from the input. To speed up their algorithm, they use a lookup table using a fixed number of bits from the input. This is then followed by looking up an entry of the offset array for every subsequent bit of the input. The lookup table is often kept in fast memory as compared to the offset array. The overall decode time can be optimized by accommodating more words in the lookup table. This can be modeled as a special case of the COPT problem where the memory hierarchy comprises of only two levels. The first level corresponds to a cache or scratchpad having constant memory access cost. The second level corresponds to the main memory for which every access incurs a cost of q. This corresponds to a blocking scheme of $<(w_1, z)$,(w_2, q),(w_2, q),$\ldots>$. Any entry of the prefix tree residing in the cache can be accessed with constant cost z and thereafter every entry in the main memory is accessed with cost q. Intuitively, if the codes cannot fit into the topmost block, we need a design that will minimize the number of higher level (deep) blocks. Having a hard-bound on the code word length has been previously dealt under Length Limited Huffman Code (LLHC) problem[21]; we define a variation to deal with the current problem using a notion of penalties.

LLHC is a well studied variant of Huffman coding motivated by the construction of optimal prefix-free codes under certain practical conditions [14] such as computer file searching and text retrieval systems [30]. The LLHC(C, D) problem outputs a prefix-free encoding over alphabet C, whose lengths are bounded by D such that the average code length is minimized. The encoding length bound, D, is a hard bound in the LLHC problem and is naturally bounded by the size of the alphabet n. Consider a soft version of the LLHC problem, where there is a penalty associated with the character encodings exceeding bound D that increases linearly with the length of the encoding. Given a bound on the admissible penalty, the goal of the problem is to find a prefix-free encoding having minimum average code length and penalty within the specified admissible bound. We note that this problem also allows us to consider settings where the desired encoding length D is smaller than $\log n$; this is impossible in the LLHC setting because of the information theoretic bottleneck.

We next define this generalized version of the LLHC problem more formally. For a character having depth λ in a prefix tree T, we associate a penalty, $p(.)$ as follows:

$$p(\lambda) = \begin{cases} z & \text{if } \lambda \leq D \\ z + q \cdot (\lambda - D) & \text{if } \lambda > D \end{cases}$$

for some constants z and q. Here, z is a constant cost for encodings having length no more than D and q is the penalty for every extra encoding bit used beyond D. The reader may note that this is a simplification from the natural blocking model where the number of bits may be more than 1. However, this assumption allows us to exploit certain properties leading to very fast solutions that are likely to work well in practice. The penalty of the prefix tree is the sum of the penalties of all the characters weighted by their frequencies, i.e.,

$$P(T) = \sum_{c \in C} freq(c) \cdot p(d_T(c)). \tag{2}$$

Problem Definition (Soft-LLHC)**:** Given parameters z, q & D, which define the penalty function $p(.)$ and a penalty bound $\mathcal{P}$, the goal of the *Soft length limited Huffman coding problem*, denoted Soft-LLHC$(\mathcal{P}, z, q, D)$, is to determine a prefix tree, T, that minimizes the average code length $len(T)$ subject to $P(T) \leq \mathcal{P}$.

Figure 3 illustrates the Huffman coding for an alphabet C, the corresponding LLHC and Soft-LLHC when $D = 3$. We note that LLHC is a special case of this problem wherein $z = 0$, $q = 1$ and $\mathcal{P} = 0$. This setting does not allow for any penalty, and constrains codewords to have length $\leq D$. Thus Soft-LLHC is a generalization of the LLHC problem. We present a fast algorithm for the Soft-LLHC problem:

▶ **Theorem 2.** *There exists an algorithm to solve the* Soft-LLHC*$(\mathcal{P}, z, q, D)$ problem with running time $O(nD)$ when the characters of C are given in sorted order of frequencies. For the case when $D = o(\log n)$, the running time of the algorithm can be bounded by $O(n + D2^D)$.*

Note that a special case of our COPT problem with two levels of memory hierarchy for $BS =< (w_1, z), (1, q), (1, q) \cdots >$ can be mapped to the Soft-LLHC problem by taking $D = w_1$ and $\mathcal{P} = \Delta$.

Lastly, we study a more generalized version of the Soft-LLHC problem that also generalizes the COPT problem. In this problem, the penalty and cost functions can be any monotonically non-decreasing function of the code length.

We next define this problem formally.

Problem Definition (Gen-LLHC)**:** Given parameters $\mathcal{P}$, called the *penalty bound*, a penalty function $p(\cdot)$ and an objective function $f(\cdot)$ that are both monotonically non-decreasing functions, the goal of the *Generalized length limited Huffman coding problem*, denoted Gen-LLHC$(\mathcal{P}, p(\cdot), f(\cdot))$, is to determine a prefix tree, T, that minimizes

$$F(T) = \sum_{c \in C} freq(c) \cdot f(d_T(c))$$

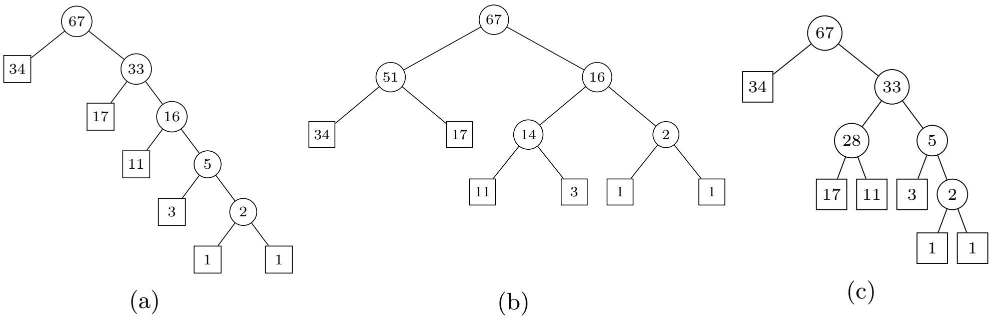


Figure 3 Consider an alphabet with 6 characters with frequencies $1, 1, 3, 11, 17, 34$ and $D = 3$. Any character with depth $w \leq 3$ bits has a penalty of z unit whereas characters with depth $w > 3$ bits have penalty $z + q \cdot (w - D)$. (a) illustrates the corresponding Huffman tree that has code length of $5 \cdot 1 + 5 \cdot 1 + 4 \cdot 3 + 3 \cdot 11 + 2 \cdot 17 + 1 \cdot 34 = 123$ and penalty of $(z + 2q) \cdot 1 + (z + 2q) \cdot 1 + (z + q) \cdot 3 + z \cdot 11 + z \cdot 17 + z \cdot 34 = 67z + 7q$. (b) Illustrates the LLHC prefix tree with higher code length of $3 \cdot 1 + 3 \cdot 1 + 3 \cdot 3 + 3 \cdot 11 + 2 \cdot 34 + 2 \cdot 17 = 150$ but a penalty of $z \cdot 1 + z \cdot 1 + z \cdot 3 + z \cdot 11 + z \cdot 17 + z \cdot 34 = 67z$. (c) Illustrates the soft-LLHC prefix tree with code length of $4 \cdot 1 + 4 \cdot 1 + 3 \cdot 3 + 3 \cdot 11 + 2 \cdot 17 + 1 \cdot 34 = 128$ and penalty of $(z + q) \cdot 1 + (z + q) \cdot 1 + z \cdot 3 + z \cdot 11 + z \cdot 17 + z \cdot 34 = 67z + 2q$.

subject to the penalty being bounded by the specified penalty bound, i.e.,

$$P(T) = \sum_{c \in C} freq(c) \cdot p(d_T(c)) \leq \mathcal{P}.$$

Note that in GEN-LLHC the penalty function is not necessarily linear, as it was in SOFT-LLHC.

Also note that the COPT problem can be modeled as the GEN-LLHC problem by taking the penalty function as $p(d_T(c)) = \delta_T(c)$, $\mathcal{P}$ as Δ and the function f mapping to the code length, i.e., $f(d_T(c)) = d_T(c)$. Note that the effect of BS is handled in the way $p(d_T(c))$ is defined. We present the following results for the GEN-LLHC problem:

▶ **Theorem 3.**

(a) *There exists a dynamic programming algorithm to solve the* GEN-LLHC$(\mathcal{P}, p(\cdot), f(\cdot))$ *problem that runs in* $O(n^3 \cdot \mathcal{P})$ *time.*

(b) *There exists a dynamic programming algorithm that returns a prefix-tree having objective value at most* $(1 + \epsilon)$ *times that of the optimal solution to* GEN-LLHC$(\mathcal{P}, p(\cdot), f(\cdot))$ *and penalty no more than* $\mathcal{P}$*. The running time of this algorithm is* $O(n^4/\epsilon)$*.*

▶ Remark 4. Note that while the running time in Theorem 2 has no dependence on $\mathcal{P}$, Theorem 3(a) is not a strictly polynomial time algorithm for super polynomial values of $\mathcal{P}$.

▶ Remark 5. Theorem 3(a),(b) assume the functions $p(\cdot), f(\cdot)$ can be computed in $O(1)$ time.

Hardness. Note that it follows from Theorem 2 that the SOFT-LLHC problem is in P as D can be at most n. We do not have a hardness result for the GEN-LLHC problem though we present a PTAS for the problem in Theorem 3. The COPT problem is a special case of the GEN-LLHC problem for which we give an algorithm which runs in polynomial time when the number of block levels is constant.

1.1 Related Work

The first algorithm for LLHC was due to Karp[21] and was based on an integer linear programming formulation. Gilbert[15] then gave an enumeration based algorithm for LLHC. Both these algorithms had exponential running time. Later Hu and Tan[19] gave an $O(nD2^D)$ time Dynamic Programming algorithm. Note that D is bounded by n in the worst case. In 1974, Garey[14] presented the first polynomial time algorithm, running in time $O(n^2D)$ for the case of binary encoded alphabets. Larmore[23] combined techniques of [19] and [14] to give an algorithm with running time $O(n^{3/2}D\log^{1/2} n)$ for the binary case. Larmore and Hirschberg [24] then designed a completely new algorithm with running time $O(nD)$; this algorithm was based on a reduction to the coin collector's problem, which was then solved using a technique they called the Package-Merge algorithm. There have been several subsequent works that have improved the running time further for the special case when $D = \omega(\log n)$ to $O(n\sqrt{D\log n} + n\log n)$ by Aggarwal, Schieber and Tokuyama [2] and to $n2^{O(\sqrt{\log D \log\log n})}$ by Schieber [27]. Baer [3] studied a variant of the Huffman coding problem wherein there is a continuous (strictly) monotonic increasing cost (penalty) function, called Campbell penalties [11], associated with the length of a character encoding; the goal of the problem is to minimize the "mean" length of the cost function over all the characters of the alphabet. We note that this problem seeks to minimize an objective different from the average code length, thereby addressing a different setting compared to Huffman coding, LLHC and our Soft-LLHC problems. In particular, the Soft-LLHC problem seeks to minimize the average code length constrained by a budget on the admissible penalty.

Generalized cost functions for building Huffman trees have been studied before. Fujiwara and Jacobs [13] studied the Generalized Huffman Tree (GHT) problem in which the cost of each encoded character depends on its depth in the tree by an arbitrary function. Here the goal is to determine a prefix tree, T, that minimizes $\sum_{i=1}^{|C|} f_i(d_T(c_i))$ for the GHT problem and minimizes $\max_{i=1}^{|C|} f_i(d_T(c_i))$ for the Max-GHT problem. This is a further generalization of our cost function, where a separate function is associated with each character.

On the other hand, the Soft-LLHC problem corresponds to optimizing the objective function allowing deviations from the individual code lengths for which we provide bi-criterion results that are novel to the best of our knowledge. We do note however that the LLHC problem is a special case of both the Soft-LLHC problem (as specified earlier) as well as the GHT problem (by taking the cost function to be ∞ when depth exceeds D and equal to frequency times depth otherwise).

Fujiwara and Jacobs [13] further prove that the Max-GHT problem is NP-hard when the cost functions are allowed to be arbitrary and provide a polynomial time algorithm when the cost functions are non-decreasing. We observe that the hardness result crucially depends on the the prefix tree being a complete binary tree. However, we show that for certain functions, the optimal prefix tree for Max-GHT need not necessarily be a complete binary tree (see Appendix A). As a matter of fact, we present a simple polynomial time construction for the relaxed version of Max-GHT by reducing the Max-GHT problem with arbitrary functions into Max-GHT problem with non-decreasing functions in $O(n^2)$ time. Using the polynomial time algorithm of Fujiwara and Jacobs [13] for the case when the cost functions are non-decreasing, this actually yields a polynomial time algorithm for the case of arbitrary functions as well. This result is captured in the following theorem and it's proof is presented in Appendix A.

▶ **Theorem 6.** *There is an $O(n^2 \log n)$ algorithm for Max-GHT with arbitrary functions.*

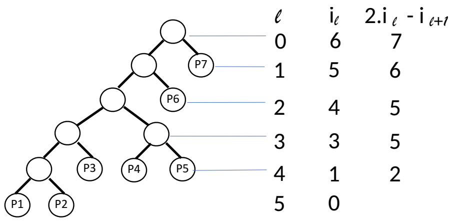


Figure 4 Illustration of the calculation of the number of characters below level ℓ ($2i_\ell - i_{\ell+1}$). This figure is taken from [16].

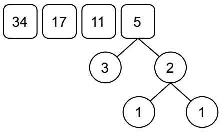


Figure 5 The **3-level forest** to the tree T, shown in Figure 3(a).

Organization of the paper. Our algorithms build on the dynamic program for Huffman codes proposed by Larmore and Przytycka[22] and extended in Golin[16]. This algorithm is discussed in Section 2. In Section 3, we first present our algorithm for the simplest of the problems, Soft-LLHC; this provides the proof for Theorem 2. In Section 4, we discuss the algorithmic approach for the generalized version of the Gen-LLHC problem; this corresponds to Theorem 3. In Section 5, we present the algorithms for the COPT problem. This is presented last as the proofs reuse results from the algorithm for Gen-LLHC. We present the proof for the PTAS corresponding to Theorem 1(b) and defer the proof of Theorem 1(a) to the Appendix. We end with concluding remarks in Section 6.

2 Preliminaries: a DP for Huffman codes

Consider a prefix tree, T. The nodes of T can be classified as either leaf nodes (i.e., nodes with no child nodes), or internal nodes (i.e., nodes with exactly 2 child nodes). Leaf nodes represent characters of the alphabet. Let $d_T(u)$ denote the depth of any node in the tree, T (with the root being at depth 0). The depth (or height) of the tree, denoted $h(T)$ is the maximum depth of any node in the tree, i.e., $h(T) = \max_{u \in T}\{d_T(u)\}$. We use the variable ℓ to refer to the level starting from the top of the tree ($\ell = 0$ for the root). Further, let i_ℓ denote the number of internal nodes at or deeper than level ℓ.

This is illustrated in Figure 4. The following proposition relates the number of characters below some level with the number of internal nodes at different levels.

▶ **Proposition 7.** *The number of characters below (deeper than) level ℓ is $2i_\ell - i_{\ell+1}$.*

The formal proof of the proposition can be found in [16]. The intuitive idea is as follows: to form each internal node we need two child nodes (can be either internal or leaf). Hence, for i_ℓ internal nodes we would require $2i_\ell$ nodes at or deeper than level $(\ell + 1)$. Since of these $2i_\ell$ nodes $i_{\ell+1}$ are internal nodes at or deeper than level $(\ell + 1)$, the number of characters or leaf nodes below level ℓ must be $2i_\ell - i_{\ell+1}$.

The following Theorem is an adaptation of a result from Golin and Zhang[16] that specifies a condition for us to be able to construct a valid prefix tree. Note that Golin and Zhang [16] did not require the condition that $\forall \ell \leq h-2, n \geq (2i_\ell - i_{\ell+1}) \geq (2i_{\ell+1} - i_{\ell+2})$. They instead proved that any sequence that is an optimal solution to the LLHC problem corresponds to a valid prefix tree (Lemma 2 and 8 in [16]). We instead show that this extra condition is necessary and sufficient, for any $\mathcal{I}$ to correspond to a valid full binary prefix tree.

▶ **Theorem 8.** *Given a decreasing sequence of integers, $\mathcal{I} = \langle i_k, i_{k+1}, \ldots, i_h = 0\rangle$, such that $\forall \ell \leq h-2, \quad n \geq (2i_\ell - i_{\ell+1}) \geq (2i_{\ell+1} - i_{\ell+2})$ and $i_k \leq n-1$ we can construct a forest, rooted at level k, such that the number of internal nodes at or below level ℓ is i_ℓ.*

We defer the proof of Theorem 8 to Appendix B. Corollary 9 follows from Theorem 8 when $k = 0$ and $i_0 = n - 1$.

▶ **Corollary 9.** *Given a decreasing sequence of integers, $\mathcal{I} = \langle i_0 = n-1, i_1, \ldots, i_h = 0\rangle$, such that $\forall \ell \leq h-2:\ n \geq (2i_\ell - i_{\ell+1}) \geq (2i_{\ell+1} - i_{\ell+2})$, we can construct a prefix tree of height h such that the number of internal nodes at or below level ℓ is i_ℓ.*

Observe that in any optimal prefix-tree, a character with higher frequency cannot appear lower than a character having lower frequency (otherwise we could swap them leading to an improved codelength). Using this fact, the following result from Golin and Zhang[16] helps us to rewrite the code length of the code represented by a prefix tree as the sum of contributions of prefix sums at each level.

▶ **Theorem 10.** *Let $S = [S_1, S_2 \cdots S_n]$ be prefix sum array of frequencies, where $S_i = \sum_{j=1}^{i} freq(j)$ and frequencies are sorted in increasing order of the depths of the characters in the tree T. Then the code length of the tree, T, can be written as a sum of h prefix sums, where each sum represents the code length contribution by each level of the tree, i.e., $len(T) = \sum_{\ell=0}^{h-1} S_{2i_\ell - i_{\ell+1}}$.*

The formal proof can be found in [16]. The intuitive idea is as follows:

by the definitions of $len(T)$ and $d_T(c)$ (Eqn 1 and depth of character c in tree T), we have

$$len(T) = \sum_{c\in C} freq(c)\cdot d_T(c) = \sum_{c\in C}\sum_{\ell=1}^{d_T(c)} freq(c)$$

By rearranging the summation over each level and using Proposition 7, we get $len(T) = \sum_{\ell=0}^{h-1}\sum_{j=1}^{2i_\ell - i_{\ell+1}} freq(j)$ and viewing the inner sum as prefix sum, we get $len(T) = \sum_{\ell=0}^{h-1} S_{2i_\ell - i_{\ell+1}}$.

The goal of Golin and Zhang[16] is to determine a prefix tree, T, for which the code length, i.e., $len(T)$ is minimum. The idea of their dynamic program is as follows. Let $H(i)$ denote the minimum code length amongst all forests having exactly i internal nodes. Then $H(n-1)$ yields the optimal code length. As mentioned in Theorem 9, it suffices to obtain a sequence of i_ℓ's to determine the prefix tree. Suppose that $i_\ell = i$ and $i_{\ell+1} = j$, then due to Larmore and Przytycka [22] we have, $H(i) = H(j) + S_{2i-j}$. This allows us to determine the optimal values of i_ℓ's as follows. We initialize H to ∞ for all entries and then use the following recurrence:

$$H(i) = \min_{\substack{j\in[max(0,2i-n),i-1]\\ \&\ 2i-j\ \geq\ 2j-k}} H(j) + S_{2i-j}$$

where k is the recursive index used in populating $H(j)$, i.e., $H(j)$ was minimized for $H(k) + S_{2j-k}$ (this can be recorded in a separate table); the condition $(2i - j\ \geq\ 2j - k)$ ensures that the number of leaves below the root level in the structure with i internal nodes is greater than or equal to that in the structure with j internal nodes. Here, $H(0)$ and S_0 are initialized to 0.

Time complexity. As there are n entries of H and each entry requires $O(n)$ computations to compare the recurrences, the algorithm takes $O(n^2)$ time. Using the concavity of S_i, this was improved to $O(n)$ time in [25] by filling the cells using Concave Least weight Subsequence (CLWS), which can be solved using SMAWK algorithm as a subroutine in $O(n)$ time [29].

▶ Remark. We use a slightly different notion of level than [16]. While [16] considers levels starting with the bottom most level as 0 and increasing up to the root, we consider levels to start with 0 from the root and increasing down the tree. The above theorems and lemmas have been rephrased accordingly.

3 Algorithm for the Soft-LLHC (Soft-LLHC) Problem

Note that Soft-LLHC$(\mathcal{P}, z, q, D)$ can be reformulated as Soft-LLHC$(\mathcal{P}', 0, 1, D)$ by taking $\mathcal{P}' = \frac{1}{q} \cdot (\mathcal{P} - z \sum_{c \in C} freq(c))$. Here on we work with this reformulation of the problem.

Consider the structure of the prefix tree, T, in a solution to the Soft-LLHC problem. Recall that a character with higher frequency cannot appear below a character with lower frequency. We can view the tree as comprising of levels starting with level 0 at the root. We define the **d-level forest of T, denoted** $F_d(T)$, to be the forest induced on T, obtained by removing all the internal nodes having depth less than d (along with their incident edges). Note that the leaf nodes having depth less than or equal to d become singleton trees in $F_d(T)$. See Figure 5 for an illustration. Let $d_{F_d(T)}(c)$ denote depth of character c in this forest. Note that in the reformulated version of our problem, the penalty of the entire tree T is equal to the codelength of the forest rooted at level D for any tree. Hence for $d \leq D$, the penalty of the tree T can be written as

$$\sum_{c \in C : d_{F_d(T)}(c) > D-d} \big(d_{F_d(T)}(c) - (D-d)\big) \cdot freq(c)).$$

We maintain a table $\overline{H}$ of size $\mathcal{C} \times D$. Intuitively, for $0 \leq i < |\mathcal{C}|$ and $1 \leq d \leq D$, an entry $\overline{H}(i, d)$ of this table tries to capture the structure of the d-level forest, $F_d(T')$, corresponding to the best prefix tree, T', for which $F_d(T')$ comprises i internal nodes. More precisely, an entry $\overline{H}(i, d)$ of this table represents the minimum amongst the code lengths of all forests (over the alphabet $\mathcal{C}$) comprising of exactly i internal nodes and additionally satisfying the condition that the penalty condition is not violated, i.e.,

$$\sum_{c \in C : d_{F_d(T)}(c) > D-d} \big(d_{F_d(T)}(c) - (D-d)\big) \cdot freq(c)) \leq \mathcal{P}'.$$

Note that for $d < D$, the d level forest $F_d(T^*)$ in the optimal tree T^* is formed by introducing new internal nodes that combine some of the trees of the forest $F_{d+1}(T^*)$ (by merging their roots pairwise to form new internal nodes). From the previous section, the code length of a prefix-tree of height h can be represented as sum of h prefix-sums. This is applicable for the Soft-LLHC problem as well. Thus, the values of the table $\overline{H}$ can be computed as follows. Initialize all entries of $\overline{H}$ to ∞ and then use the following recurrence for $d < D$:

$$\overline{H}(i, d) = \min_{\substack{j \in [max(0, 2i-n), i-1] \\ \& \ 2i-j \ \geq \ 2j-k}} \overline{H}(j, d+1) + S_{2i-j}.$$

Here k corresponds to the recursive index used in populating $\overline{H}(j, d+1)$, i.e., $\overline{H}(j, d+1)$ was minimized for $\overline{H}(k, d+2) + S_{2j-k}$ (this can be recorded in a separate table).

Note that at level D, an entry $\overline{H}(i, D)$ corresponds to the minimum code length amongst all forests having exactly i internal nodes and penalty no more than $\mathcal{P}'$. Since the penalty of this tree corresponds exactly to its codelength (as $z = 0$ and $q = 1$ for the reformulated Soft-LLHC), this actually corresponds exactly to the entry $H(i)$, provided $H(i) \leq \mathcal{P}'$ and we can thus initialize $\overline{H}(i, D) = H(i)$. Note that if $H(i) > \mathcal{P}$, then there does not exist a tree with penalty less than $\mathcal{P}$ and having i internal nodes with depth at least D; thus we can set $\overline{H}(i, D) = \infty$ in this case. The final solution is then obtained from the entry $\overline{H}(n-1, 0)$. The prefix tree can be constructed by alluding to Theorem 9.

Time complexity. The entries of H can be computed in time $O(n)$ as discussed previously. For computing $\overline{H}$, there are nd cells and each cell takes $O(n)$ time to fill using the recurrence above. Hence the running time is $O(n^2 D)$. This time can be improved by employing properties of Monge matrices. This is discussed next.

Improving the running time using Monge property. Monge property is a discrete extension of concavity which allows for the speeding up of several algorithms[9]. SMAWK is one such classical algorithm, using which row-minima can be found. It was used by Golin and Zhang[16] to solve the LLHC problem. We follow a similar approach. Consider the recurrence

$$\widehat{H}(i, d) = \min_{j \in [max(0, 2i-n), i-1]} \widehat{H}(j, d+1) + S_{2i-j}.$$

Note that we drop the condition $2i - j \geq 2j - k$ from the recurrence for $\overline{H}$. This is because we can argue that the optimal sequence will correspond to a valid prefix-tree. As all the solutions of $\widehat{H}$ satisfy the penalty constraint, we only minimize the code length. If the optimal sequence doesn't correspond to a valid prefix-tree, it is possible to construct a sequence using Lemma 8 in [16], with a smaller value of $\sum_{\ell=0}^{D-1} S_{2i_\ell - i_{\ell+1}}$, leading to a contradiction.

Now consider a new implicit matrix $M^{(d)}$ such that for all $0 \leq i, j \leq n$: $M^{(d)}_{i,j} = \widehat{H}(j, d+1) + S_{2i-j}$ when $0 \leq 2i - j \leq n$ and ∞ when $2i - j > n$ or $2i - j < 0$. We show that $M^{(d)}$ is a Monge matrix. This follows from the following Lemma.

▶ **Lemma 11.** $M^{(d)}_{i,j} + M^{(d)}_{i+1,j+1} \leq M^{(d)}_{i+1,j} + M^{(d)}_{i,j+1}$.
(Note that SMAWK allows for this condition to be satisfied when both sides evaluate to ∞).

Proof. We consider the following three (exhaustive) cases:
(I) *When* $2i - j < 1$*:* $M^{(d)}_{i,j+1}$ is ∞ and thus the result holds by definition (as $2i - (j+1) < 0$).
(II) *When* $2i - j > n - 2$*:* $M^{(d)}_{i+1,j}$ is ∞ and thus the result holds by definition (as $2(i+1) - j > n$).
(III) *When* $1 \leq 2i - j \leq n - 2$*:* entries $M^{(d)}_{i,j}, M^{(d)}_{i+1,j+1}, M^{(d)}_{i+1,j}, M^{(d)}_{i,j+1}$ are defined and we have:

$$\begin{aligned}(M^{(d)}_{i,j} + M^{(d)}_{i+1,j+1}) \;&-\; (M^{(d)}_{i+1,j} + M^{(d)}_{i,j+1}) \\ &\leq\; (\widehat{H}(j, d+1) + S_{2i-j} + \widehat{H}(j+1, d+1) + S_{2i-j+1}) \\ &\qquad -\; (\widehat{H}(j, d+1) + S_{2i-j+2} + \widehat{H}(j+1, d+1) + S_{2i-j-1}) \\ &=\; S_{2i-j} + S_{2i-j+1} - S_{2i-j+2} - S_{2i-j-1} \\ &=\; freq(c_{2i-j}) - freq(c_{2i-j+2}) \quad \leq \quad 0\end{aligned}$$

where c_i is the ith least frequent character and thus the result holds. ◀

Observe that, by definition, $\widehat{H}(i, d) = \min_{0 \leq j \leq i} M^{(d)}_{i,j} = \min_{0 \leq j \leq n} M^{(d)}_{i,j}$. The Monge property on $M^{(d)}$ implies that the SMAWK algorithm[1] can solve for the row minima of each $M^{(d)}$ matrix in $O(n)$ time. Thus our algorithm repeats the process of finding row minima of each $M^{(d)}$ matrix for $d = D - 1$ to 0 to obtain the minima corresponding to $\widehat{H}(., d)$. Thus solving for D such matrices takes $O(nD)$ time(See Algorithm 1). The number of internal nodes with depth at most D is bounded by 2^{D+1} and hence the computation of $H(i)$ takes $O(n)$ time. Thus, the run time can be bounded by $O(n + D2^D)$ when $D = o(\log n)$.

Algorithm 1 (for **Theorem 2**).

```
Input: Weighted Alphabet $C = \{c_1, c_2, \ldots, c_n\}$; Penalty bound $\mathcal{P}$;
Output: Minimum code length prefix tree having penalty less than $\mathcal{P}$
$S \leftarrow$ Prefix sum array of sorted frequencies
$H(i) \leftarrow$ From CLWS for all $i \in [0, n-1]$
for $i \leftarrow 0$ to $n-1$ do
    if $H(i) \leq \mathcal{P}$ then
        $\widehat{H}(i, D) = H(i)$
    if $H(i) > \mathcal{P}$ then
        $\widehat{H}(i, D) = \infty$
for $d \leftarrow D-1$ to $0$ do
    $SMAWK(M^{(d)})$ uses $\widehat{H}(i, d+1)$ and computes $\widehat{H}(i, d)$
$C^* \leftarrow \widehat{H}(n-1, 0)$
$T^* \leftarrow$ Obtain the prefix tree by following the parent pointers of $C^*$
return $T^*$;
```

4 Algorithms for the Generalized LLHC (Gen-LLHC) Problem

We build on the ideas of Golin and Zhang[16] (see Section 2 for details). By extending their construction, we show that for Gen-LLHC($\mathcal{P}, p(\cdot), f(\cdot)$), the objective value $F(T)$, for any tree T, can also be written as a sum of h terms. The i^{th} term representing the product of the sum of the frequencies of all the leaf nodes with depth less than or equal to i and the difference in the objective values at depth i and $i-1$, that is $f(i) - f(i-1)$ ($f(0) = 0$). However our goal is to minimize the objective value, $F(T)$, of the tree. We handle the penalty bound involved by maintaining an extra parameter in our proposed dynamic program. We store structures with the minimum objective value having penalty less than the new parameter (corresponding to the admissible values of penalty bound) introduced. Using this formulation we obtain an exact algorithm referred to in Theorem 3(a).

The running time of the exact algorithm is $O(n^3 \cdot \mathcal{P})$ which may be super polynomial in n for large values of $\mathcal{P}$. Subsequently, we are able to bound the number of feasible penalty values using standard rounding techniques and get an approximate algorithm which runs in $O(n^4/\epsilon)$ and has code length no more than $(1+\epsilon)$ time the optimal value. We prove a slightly generalized variant of the problem, denoted Gen-LLHC*($\mathcal{P}, p(\cdot), f(\cdot), h$) that takes an additional parameter h representing a height bound and determines a prefix tree T of height at most h that minimizes $F(T)$ subject to the penalty bound as before. We show that

▶ **Theorem 12.** *There exists a dynamic programming algorithm that returns a prefix-tree having height at most h and objective value at most $(1+\epsilon)$ times that of the optimal solution to* Gen-LLHC*($\mathcal{P}, p(\cdot), f(\cdot), h$) *and penalty $\leq \mathcal{P}$ with running time of $O(n^2h^2/\epsilon)$.*

Theorem 3(b) follows by taking the parameter h as n as that is the maximum height possible.

The details of the algorithms and proofs are presented in following subsections.

Note that unlike the Gen-LLHC problem, the Soft-LLHC has strictly polynomial running time as we use $\mathcal{P}$ only to filter and remove the infeasible solutions.

As mentioned before, we do not have a hardness result for the Gen-LLHC problem. We note that proving hardness is challenging for several problems related to Huffman coding. For instance, hardness results are not known for Huffman coding with unequal letter costs[17] that admit a PTAS. As another instance, we have shown that the hardness result for a closely related problem, MAX-GHT, due to Fujiwara and Jacobs[13] in prior literature is not correct (See Theorem 6 and the associated discussion in Section 1.1).

4.1 Exact DP for Gen-LLHC: Proof of Theorem 3(a)

We start with a simple proposition.

▶ **Proposition 13.** *A character having higher frequency will appear at the same or lower level (that is closer to the root) than a character having lower frequency.*

The proposition is easy to verify - if this was not true, one could simply swap the two characters thereby improving the objective value as well as the penalty.

Note that for the Gen-LLHC problem also, the code length can be represented as sum of h prefix-sums. We will now show that for Gen-LLHC$(\mathcal{P}, p(\cdot), f(\cdot))$, the objective value $F(T)$, for any tree T, can also be written as a sum of h terms, where each term corresponds to the contribution by the corresponding level, to the objective value, of the tree. Recall that $f(\cdot)$ was a monotonically non-decreasing function. This result is captured in Lemma 14. We define two new function $\hat{f}(.), \hat{p}(.)$:

$$\hat{f}(i) = \begin{cases} f(1) & \text{if } i = 1 \\ f(i) - f(i-1) & \text{if } i > 1 \end{cases}$$

$$\hat{p}(i) = \begin{cases} p(1) & \text{if } i = 1 \\ p(i) - p(i-1) & \text{if } i > 1 \end{cases}$$

Now if h represents the total height of a tree, T, then we have the following lemma.

▶ **Lemma 14.**

$$F(T) = \sum_{\ell=0}^{h-1} \hat{f}(\ell+1) \cdot (S_{2i_\ell - i_{\ell+1}})$$

The proof of Lemma 14 is deferred to Appendix C. Using Lemma 14, it remains to find a sequence of i_l's as before except that now the goal is to minimize the objective value, $F(T)$, of the tree. The prefix tree can be constructed by alluding to Theorem 9. We describe a recurrence to obtain such a sequence. Let $D(i, \ell, P)$ denote the minimum objective value amongst all forests rooted at level ℓ, having i internal nodes with penalty at most P (here, the objective value of the forest is the sum of the objective values of the trees in the forest).

A dynamic program using the above recurrence can be designed as follows. Let h be some upper bound on the height of the optimal prefix tree.

Base Case. For all forests with no internal nodes, we initialize the objective value to 0, i.e.,

$$\forall\ \ell \in [0, h]\ and\ P \in [0, \mathcal{P}] : \quad D(0, \ell, P) = 0$$

Inductive Case. To compute $D(i, \ell, P)$, we iterate over the number of internal nodes at depths greater than ℓ. If j internal nodes are at depths strictly greater than ℓ, then there are $(2i - j)$ characters at depths strictly greater than ℓ and $\hat{f}(\ell+1) \cdot S_{2i-j}$ is the contribution, to the objective value $F(T)$, of all the characters having level(depth) $> \ell$, due to the access at level $(\ell + 1)$. Furthermore, $D(j, \ell + 1, P')$ denotes the objective value contributed by all accesses made at levels(depths) greater than $\ell + 1$. This yields the following recurrence:

$$D(i, \ell, P) = \min_{\substack{j \in [max(0, 2i-n), i-1] \\ \&\ 2i-j\ \geq\ 2j-k}} \Big\{ D(j, \ell+1, P') + \hat{f}(\ell+1) \cdot S_{2i-j} \Big\} \tag{3}$$

where $P' = P - \hat{p}(\ell+1) \cdot S_{2i-j}$ and k is the recursive index using which $D(j, \ell+1, P')$ was populated. We only need to recurse if $P' > 0$. The tree with the optimal objective value can be obtained by maintaining the parent pointers of each update and backtracking (similar to as shown in pseudo-code of Theorem 3(b) in Appendix D).

Time complexity. There are $O(n)$ characters, h levels and $O(\mathcal{P})$ values for penalty; hence there are $O(nh\mathcal{P})$ cells in the table. As each cell can be filled in $O(n)$ time, the time complexity is $O(n^2h\mathcal{P})$. As height, h is at most n, we get the time complexity to be $O(n^3\mathcal{P})$. As $\mathcal{P}$ may not be polynomial in n, this is a pseudo-polynomial time algorithm.

As the above algorithm is symmetric in terms of penalty and objective value, we can find the tree having the minimum penalty and objective value at most $\mathcal{C}$ in $O(n^3\mathcal{C})$ time using the recurrence

$$D(i, \ell, C) = \min_{\substack{j \in [max(0, 2i-n), i-1] \\ \&\ 2i-j\ \geq\ 2j-k}} \{D(j, \ell+1, C - S_{2i-j}) + \hat{p}(\ell+1) \cdot S_{2i-j}\} \tag{4}$$

Let the penalty of the solution to the above DP be $\mathcal{P}_{dual}$, we will use it to give a $PTAS$ algorithm for $Gen - LLHC$ in the next section.

4.2 PTAS for Gen-LLHC: Proof of Theorem 3(b) and Theorem 12

We first prove Theorem 12. Theorem 3(b) follows as h takes value at most n.

The algorithm presented in the previous section has linear running time dependency on the parameter $\mathcal{P}$. In this section, we propose a polynomial time approximation algorithm that runs in time $O(n^4/\epsilon)$ and returns a prefix tree having penalty at most $\mathcal{P}$ and objective value with in $(1+\epsilon)$ times the optimal value. We first give an algorithm which returns a tree with penalty at most the value of the minimum penalty possible for tree with objective value at most $\mathcal{C}$, and objective value at most $(1+\epsilon)\mathcal{C}$.

For this we restrict the parameter C to only take on values that are multiples of $\lambda = \lfloor(\epsilon \cdot \mathcal{C})/2h\rfloor$ ranging from $0 \cdot \lambda$ upto $((2h/\epsilon) + h) \cdot \lambda$ where h is some upper bound on the height of the optimal prefix tree. We denote the dynamic program table maintained by this algorithm with $\overline{D}$. Let $\overline{D}(i, \ell, C)$ denote the minimum penalty amongst all forests rooted at level ℓ, having i internal nodes with objective value at most C (here, the penalty of the forest is the sum of the penalties of the trees in the forest). Note, here each DP cell stores a structure having minimum penalty as compared to the exact algorithm of Gen-LLHC, where each DP cell stores a structure having minimum objective value.

We define a rounding function $\mathbf{r}$ as follows:

$$\mathbf{r}(x) = \left\lceil \frac{x}{\lambda} \right\rceil \cdot \lambda.$$

We change the recurrence from the previous section as follows: The base case becomes: for all forests with no internal nodes, we initialize the objective value to 0, i.e., $\forall\ \ell \in [0, h]$ and C a multiple of λ and $C \in [0, \mathbf{r}(\mathcal{C}) + h\lambda]$: $D(0, \ell, C) = 0$.. The inductive step is modified to:

$$\overline{D}(i, \ell, C) = \min_{\substack{j \in [max(0, 2i-n), i-1] \\ \&\ 2i-j\ \geq\ 2j-k}} \left\{\overline{D}(j, \ell+1, C') + \hat{p}_{\ell+1} \cdot S_{2i-j}\right\} \tag{5}$$

where $C' = C - \mathbf{r}(\hat{f}_{\ell+1} \cdot S_{2i-j})$ and k is the recursive index using which $\overline{D}(j, \ell+1, C')$ was populated. Note that we only update entries of $\overline{D}$ for which the C parameter is itself a multiple of λ. It is easy to see that the C parameter will take on only $O(h/\epsilon)$ values. The table can be compressed accordingly and maintained only for these entries, however we omit these implementation details in the interest of better readability.

The following Lemma shows we can get a prefix-tree with penalty $\leq \mathcal{P}_{dual}$ by sacrificing an additive λ factor for every level in the objective value.

▶ **Lemma 15.** *For any valid values of i, ℓ and P:* $D(i, \ell, C) \;\geq\; \overline{D}(i, \ell, \mathbf{r}(C) + (h - \ell) \cdot \lambda)$.

From the previous section, we know that the optimal solution is captured by $D(h-1, 0, \mathcal{C})$. Hence the above Lemma (proof in Appendix D) implies that the optimal solution is also captured by $\overline{D}(h-1, 0, \mathbf{r}(\mathcal{C}) + h \cdot \lambda)$. We now look at all the entries $\overline{D}(h-1, 0, \mathbf{r}(C) + h \cdot \lambda) \leq P_{dual}$ and pick the entry with the minimum value of $\mathbf{r}(C) + h \cdot \lambda$.

Hence using $\overline{D}$, given a objective function threshold $\mathcal{C}$, we can find a prefix tree with penalty $\leq P_{dual} \leq \mathcal{P}$ and objective value $\leq (1+\epsilon) \cdot \mathcal{C}$. Now, if substitute $\mathcal{C} = C^*$, where C^* is the objective value of the solution to the Gen-LLHC problem, we will have the solution to the *PTAS* of Gen-LLHC problem. Now, instead of calculating C^* directly we use binary search in the range $[0, \mathcal{F} \cdot f(n)]$, where $\mathcal{F}$ is the cumulative frequency of all the characters in the prefix tree and $f(n)$ is the value of objective function at level n. As the depth is at most n for all characters and $f(.)$ is an increasing function, the objective value is at most $\mathcal{F} \cdot f(n)$. Thus, we have a *PTAS* algorithm for Gen-LLHC.

Time complexity. There are at most $O(h)$ levels and $O(n)$ characters. C can have at most $O(h/\epsilon)$ possible values. Hence, there are $O(nh^2/\epsilon)$ cells in the table. Each cell can be filled in at most $O(n)$ time. So the time complexity is $O(n^2h^2/\epsilon)$. Since there are a total of h recursive calls, the error in objective function value is bounded by $h\lambda \leq \epsilon \cdot \mathcal{C}$. Thus, we can find a prefix tree having objective value less than $(1+\epsilon) \cdot C^*$ and penalty at most $\mathcal{P}$ in $O(n^2h^2/\epsilon)$ time. This proves Theorem 12. Taking the upper bound for the height, h, as n, we get the time complexity to be $O(n^4/\epsilon)$, which proves Theorem 3(b).

5 Algorithms for the Code Optimal Prefix Tree (COPT) problem

For a fixed number of block levels, m, the possible number of values corresponding to the decode time for the forests in the dynamic program is n^{m-1} . We use this to give an $O(n^{m+2})$ algorithm for Theorem 1(a) in Appendix F. We now present the proof of Theorem 1(b).

5.1 Proof of Theorem 1(b)

Consider the blocking scheme in the definition of the *COPT* problem. As mentioned in the introduction, the number of block levels, m, is typically a small constant in practice. We now present a more efficient dynamic program based pseudo-approximation algorithm for the case when the number of block levels is constant.

We first prove some results (c.f. Propositions 16, 17 and Lemma 18) required in the formulation of our new dynamic program. The following proposition shows that given a set of characters, we can construct a (nearly complete) prefix tree of depth $\lceil \log n \rceil$.

▶ **Proposition 16.** *There exists a prefix tree for a set of characters, C, having depth* $\lceil \log |C| \rceil$.

Proof. It is easy to verify that we can place $2^{\lceil \log |C| \rceil} - |C|$ characters at depth $\lceil \log |C| \rceil - 1$ in the subtree and the remaining characters at depth $\lceil \log |C| \rceil$ to form a valid prefix tree (additional nodes are added to serve as internal nodes). ◀

Consider a set of characters, C. The following proposition shows that given an arbitrary tree, T, with characters of C appearing as leaf nodes in T, we can always construct a valid prefix tree over C that has height no more than that of T and in which each character appears at a depth no more than its depth in T.

▶ **Proposition 17.** *Let C denote a set of characters. Given a tree, T, in which the characters of C appear as leaf nodes, there exists a valid prefix tree, T', over C that has no greater height than T and in which $d_{T'}(c) \leq d_T(c)\ \forall c \in C$.*

Proof. We start with the tree T and iteratively modify it until we obtain a valid prefix tree.

We find a node(say u) that violates any of these conditions and modify the tree as follows:

- Is a leaf node but does not correspond to a character: We simply delete u.
- Has only one child(say node v): We remove u and directly attach v to the parent of u.

It is easy to see that when no more violating nodes are left, we get a valid prefix tree. It is also straightforward to observe that we never increase the depth of any node in this process. ◀

The following Lemma shows that there cannot be too many levels in the optimal prefix tree between two consecutive characters of the alphabet when sorted in order of frequencies.

▶ **Lemma 18.** *In a complete binary tree, if c_i is a character at level ℓ and c_{i+1} is at level ℓ' then $\ell' - \ell < \lceil \log(n) \rceil$.*

Proof. We prove this by contradiction. Let us assume that $\ell' - \ell \geq \lceil \log(n) \rceil$. Since there is a leaf c_{i+1} at depth greater than ℓ, there must be at least one internal node at the level ℓ. By our assumption there are no leaves in the tree rooted at this internal node, till the next $\lceil \log(n) \rceil$ levels. Hence there are at least n internal nodes above level ℓ'. But the tree we started with has exactly $n - 1$ internal nodes as it has n leaves. Contradiction. ◀

The following lemma shows that there exists a tree having bounded height that has almost the same code length and decode time as the optimal prefix tree of $COPT(\mathcal{P})$.

▶ **Lemma 19.** *Given $\delta > 0$, there exists a prefix tree, T', for which the code length is at most $(1 + \delta)$ times the code length of $COPT(\mathcal{P})$ and the height of T' is no more than $2m(\lceil 1/\delta \rceil + \lceil \log n \rceil)$, where m is the number of block levels.*

Proof. Let h^* denote the height (total number of tree levels) of the optimal prefix tree (solution to $COPT(\mathcal{P})$). If $h^* \leq 2m(\lceil 1/\delta \rceil + \lceil \log n \rceil)$, then the claim is trivially satisfied. We therefore focus on the case when $h^* > 2m(\lceil 1/\delta \rceil + \lceil \log n \rceil)$. As there are at most m block levels, at least one of these has more than $2(\lceil 1/\delta \rceil + \lceil \log n \rceil)$ tree levels.

Let us focus on one such block level, and let the starting tree level for the block level be ℓ'. There must be at least one node c_i between the tree levels $\ell' + \lceil 1/\delta \rceil + \lceil \log n \rceil)$ and $\ell' + \lceil 1/\delta \rceil + 2\lceil \log n \rceil)$ due to Lemma 18.

▶ **Proposition 20.** *In the optimal prefix tree, the kth highest frequency is at a level at most $k + \lceil \log(n) \rceil$.*

The proof of Proposition 20 is deferred to Appendix G. From the proposition, there are at least $\lceil 1/\delta \rceil$ nodes with frequency higher than c_i.

Let T^* be the prefix tree corresponding to the optimal solution $COPT(\mathcal{P})$ and $\ell = \ell' + \lceil 1/\delta \rceil + \lceil \log n \rceil$. We modify T^* to construct another prefix tree, T' as follows:

- all characters up to $\ell + \lceil \log n \rceil$ retain the same level as in T^*, except for c_i
- c_i is replaced with a new internal node, say u, and made a child of u (c_i is at level $\ell + 1$).
- We call a character of T^* *deep* if it has depth more than $2m(\lceil 1/\delta \rceil + \lceil \log(n) \rceil)$. Let γ be the number of deep characters in T^*. Using Proposition 16, there exists a subtree comprising of all the deep characters of T^*, having depth at most $\lceil \log \gamma \rceil \leq \lceil \log n \rceil$. We attach this subtree as the second child of u. The level of any of the characters in this subtree is no more than $\ell + \lceil \log n \rceil + 1$.
- We finally invoke Proposition 17, to get a valid prefix tree.

We now show that the codelength and decode time of T' are no more than $(1+\delta)$ times the corresponding parameters of T^*. Note that both the code length and decode time of the deep characters of T^* only reduces as their depth reduces in T'. Therefore the code length and decode time can only increase due to the character c_i moving one level (tree level) down.

We first analyze the increase in code length due to c_i moving one (tree) level down. Recall that the block level to which c_i belonged was divided into $2\lceil 1/\delta \rceil$ partitions and c_i belongs to the $\lceil 1/\delta \rceil^{th}$ partition. Moreover each partition contains a character. Also, there are at least $\lceil 1/\delta \rceil$ nodes with frequency higher than c_i and hence have tree level same or above that of c_i. Thus $len(T^*) \geq \lceil 1/\delta \rceil \cdot f_i$. The increase in code length incurred by moving c_i down one tree level is f_i. Thus $f_i \leq \delta \cdot (\lceil 1/\delta \rceil \cdot f_i) \leq \delta \cdot len(T^*)$. Therefore $len(T') \leq (1+\delta) \cdot len(T^*)$.

As c_i lies between the tree levels $\ell' + (\lceil 1/\delta \rceil + \lceil \log n \rceil)$ and $\ell' + (\lceil 1/\delta \rceil + 2\lceil \log n \rceil)$, the next tree level to c_i must also be in the same block level. Therefore $\Delta(T') \leq \Delta(T^*) \leq \mathcal{P}$. ◀

Given the above Lemma, the algorithm is quite straightforward - we simply invoke Theorem 12 by bounding the h parameter by $2m(\lceil \log(n) \rceil + \lceil 1/\delta \rceil)$.

Time Complexity. The analysis is same as that of the algorithm for Theorem 12; we know that the time taken is $O(h^2 n^2/\epsilon)$.Taking the bound on the height h to be $2m(\lceil \log(n) \rceil + \lceil 1/\delta \rceil)$ and setting δ to be ϵ, the running time becomes $O\left(\frac{n^2 \cdot m^2}{\epsilon}\left(\log^2(n) + \frac{1}{\epsilon^2}\right)\right)$. For constant m, this yields a complexity of $O\left(\frac{n^2}{\epsilon} max\left(\frac{1}{\epsilon^2}, \log^2(n)\right)\right)$.

6 Conclusion and open problems

Motivated by many practical challenges in implementing compression, we introduce and study a novel variation of finding optimal prefix trees where one is allowed to deviate from the optimal code length within a specified bound. This allows us to capture more generalized decoding costs for which we develop a bi-criterion framework and present efficient algorithms. An important application of this framework is to a natural class of memory access cost functions that use blocking and to the best of our knowledge, this is the first work that lays the theoretical foundations and present a family of algorithms with provable guarantees. An open problem is to proving NP-hardness for the Gen-LLHC problem that could be quite challenging as exemplified by Theorem 6. Another interesting future direction is to study the empirical performance of our algorithms with real world data sets on practical systems with hierarchical memory; we anticipate promising results, similar to those obtained for a closely related variant in the hierarchical memory setting where the goal is to minimize the decode time and the average code length is bound by a threshold parameter[4].

References

1 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.

2 Alok Aggarwal, Baruch Schieber, and Takeshi Tokuyama. Finding a minimum-weightk-link path in graphs with the concave monge property and applications. *Discrete & Computational Geometry*, 12(3):263–280, 1994.

3 M.B. Baer. Source coding for quasiarithmetic penalties. *IEEE Transactions on Information Theory*, 52(10):4380–4393, 2006. doi:10.1109/TIT.2006.881728.

4 Shashwat Banchhor, Rishikesh R. Gajjala, Yogish Sabharwal, and Sandeep Sen. Decode efficient prefix codes. *CoRR*, abs/2010.05005v2, 2020. arXiv:2010.05005v2.

5 Shashwat Banchhor, Rishikesh R. Gajjala, Yogish Sabharwal, and Sandeep Sen. Decode-efficient prefix codes for hierarchical memory models. In *Data Compression Conference, DCC 2020*, page 360. IEEE, 2020. doi:10.1109/DCC47342.2020.00077.

6 Shashwat Banchhor, Rishikesh R. Gajjala, Yogish Sabharwal, and Sandeep Sen. Efficient algorithms for decode efficient prefix codes. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *31st Data Compression Conference, DCC 2021, Snowbird, UT, USA, March 23-26, 2021*, page 338. IEEE, 2021. doi:10.1109/DCC50243.2021.00080.

7 Thomas Boutell. PNG (portable network graphics) specification ver 1.0. *RFC*, 2083:1–102, 1997. doi:10.17487/RFC2083.

8 Vladimir Britanak. A survey of efficient MDCT implementations in MP3 audio coding standard: Retrospective and state-of-the-art. *Signal Process.*, 91(4):624–672, 2011. doi:10.1016/j.sigpro.2010.09.009.

9 Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996. doi:10.1016/0166-218X(95)00103-X.

10 M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Systems Research Centre, 1994.

11 LL Campbell. Definition of entropy by means of a coding problem. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 6(2):113–118, 1966.

12 Peter Deutsch. DEFLATE compressed data format specification ver 1.3. *RFC*, 1951:1–17, 1996. doi:10.17487/RFC1951.

13 Hiroshi Fujiwara and Tobias Jacobs. On the huffman and alphabetic tree problem with general cost functions. *Algorithmica*, 69(3):582–604, 2014. doi:10.1007/s00453-013-9755-6.

14 M. R. Garey. Optimal binary search trees with restricted maximal depth. *SIAM J. Comput.*, 3(2):101–110, 1974. doi:10.1137/0203008.

15 Edgar N. Gilbert. Codes based on inaccurate source probabilities. *IEEE Trans. Inf. Theory*, 17(3):304–314, 1971. doi:10.1109/TIT.1971.1054638.

16 M. Golin and Y. Zhang. A dynamic programming approach to length-limited huffman coding: Space reduction with the monge property. *IEEE Trans. on Information Theory*, 56(8):3918–3929, 2010. doi:10.1109/TIT.2010.2050947.

17 Mordecai J. Golin, Claire Kenyon, and Neal E. Young. Huffman coding with unequal letter costs. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 785–791. ACM, 2002. doi:10.1145/509907.510020.

18 Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR 2016*, 2015. arXiv:1510.00149.

19 TC Hu and KC Tan. Path length of binary search trees. *SIAM Journal on Applied Mathematics*, 22(2):225–234, 1972.

20 D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. doi:10.1109/JRPROC.1952.273898.

21 Richard M. Karp. Minimum-redundancy coding for the discrete noiseless channel. *IRE Trans. Inf. Theory*, 7(1):27–38, 1961. doi:10.1109/TIT.1961.1057615.

22 Lawrence Larmore and Teresa Przytycka. Constructing huffman trees in parallel. *SIAM Journal on Computing*, 24, July 1998. doi:10.1137/S0097539792233245.

23 Lawrence L. Larmore. Height restricted optimal binary trees. *SIAM J. Comput.*, 16(6):1115–1123, 1987. doi:10.1137/0216070.

24 Lawrence L. Larmore and Daniel S. Hirschberg. A fast algorithm for optimal length-limited huffman codes. *J. ACM*, 37(3):464–473, 1990. doi:10.1145/79147.79150.

25 Lawrence L. Larmore and Teresa M. Przytycka. A parallel algorithm for optimum height-limited alphabetic binary trees. *J. Parallel Distributed Comput.*, 35(1):49–56, 1996. doi:10.1006/jpdc.1996.0067.

26 A. Moffat and A. Turpin. On the implementation of minimum redundancy prefix codes. *IEEE Transactions on Communications*, 45(10):1200–1207, 1997.

27 Baruch Schieber. Computing a minimum weightk-link path in graphs with the concave monge property. *J. Algorithms*, 29(2):204–222, 1998. doi:10.1006/jagm.1998.0955.

28 G. K. Wallace. The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992.

29 Robert Wilber. The concave least-weight subsequence problem revisited. *J. Algorithms*, 9(3):418–425, September 1988. doi:10.1016/0196-6774(88)90032-6.

30 Justin Zobel and Alistair Moffat. Adding compression to a full-text retrieval system. *Softw. Pract. Exp.*, 25(8):891–903, 1995. doi:10.1002/spe.4380250804.

A Algorithm for Max-GHT: Proof of Theorem 6

A.1 Introduction

The problems GHT(Generalized Huffman Tree) and Max-GHT(Max Generalized Huffman tree) were formulated by Fujiwara and Jacobs[13]. We first state their problem definitions.

▶ **Definition 21** (GHT). *Given n arbitrary functions $f_1, f_2 \cdots f_n$ corresponding to n leaves, the objective of GHT is to determine a binary tree T with these n leaves, such that $\sum_{i=1}^{i=n} f_i(d_i)$ is minimized, where the ith leaf is at depth d_i in T.*

▶ **Definition 22** (Max-GHT). *Given n arbitrary functions $f_1, f_2 \cdots f_n$ corresponding to n leaves, the objective of Max-GHT is to determine a binary tree T with these n leaves, such that $\max_{i=1}^{i=n} f_i(d_i)$ is minimized, where the ith leaf is at depth d_i in T.*

Fujiwara et al. proved that Max-GHT and GHT are NP-hard for general functions $f_1, f_2 \cdots f_n$. However, they also proved that if each f_i is non-decreasing, then Max-GHT can be solved in $O(n^2 \log n)$ time. The complexity of GHT was unresolved, if f_i is non-decreasing.

However, there is an implicit assumption in their hardness proof. They assume that there exists a solution which is a full binary tree(all internal nodes have exactly two children) for both GHT and Max-GHT. While this has to be true when the functions are non-decreasing, it need not be true when the functions are arbitrary. Consider the following simple counter example - when there are two leaves and for $i = 1, 2$ we have function values $f_i(1) = 1$ and $f_i(2) = 0$. The optimal solution(with zero cost for both GHT and Max-GHT) will have both leaves at level 2 and hence such tree cannot be full binary. This re-opens the problems they posed and we present a simple $O(n^2)$ algorithm to convert Max-GHT and GHT with general functions to problems where Max-GHT and GHT have non-decreasing functions. As a direct consequence of this, we have an $O(n^2 \log n)$ algorithm to solve Max-GHT with general functions. Due to this reduction, we conclude that if GHT with non-decreasing functions can be solved in polynomial time, then GHT with general functions can also be solved in polynomial time. We also note that there is a solution with full binary tree for both GHT and Max-GHT with non-decreasing functions.

A.2 Reduction

▶ **Lemma 23.** *The GHT and Max-GHT problem with n arbitrary functions $f_1, f_2 \cdots f_n$, can be reduced to a problem with n non-decreasing functions $g_1, g_2 \cdots g_n$ in $O(n^2)$ time, where $g_i(j) = \min_{l=j}^{n} f_i(l)$*

Proof. We update $g_i's$ in a bottom to top manner with l^{th} entry as $\min(f_i(l), g_i(l+1))$ for $l < n$ and $g_i(n) = f_i(n)$. Hence the total time taken is $O(n)$ per function and $O(n^2)$ in total. It's easy to see that these functions evaluate to $g_i(j) = \min_{l=j}^{n} f_i(l)$ using induction.

For correctness, the key property we use is that there is an optimal tree which is a solution to GHT/Max-GHT, such that for any pair of depths (d_1, d_2), if $d_1 < d_2$ and $f_i(d_1) \geq f_i(d_2)$, then the ith leaf can not be at d_1 for any i. This is due to a simple exchange argument as we switch a node from d_1 to d_2, the Kraft sum decreases (hence the tree is feasible) and cost will not increase.(Note that if the Kraft sum for a given set of depths is less than 1, we can always construct a binary tree with those function values) Therefore the ith leaf can not be at level l if $g_i(l) \neq f_i(l)$. Hence, the structure of the optimal solution remains unchanged by changing the values for such l. ◀

We note that as the tree need not be full binary, the maximum height need not be n like in [13]. The above algorithm's correctness remains valid even when maximum height exceeds n and the run time would be $O(m)$, where m is the input size(previously n^2).

B Proof of Theorem 8

We prove this constructively by induction. For the sequence $\mathcal{I}' = \langle i_{h-1}, i_h = 0\rangle$, we can construct a forest with i_{h-1} trees, each containing one internal node and two leaves. Since this forest has no internal nodes at or below level h, we have $i_h = 0$. Also, since the only internal nodes are the roots of the trees at level $h-1$, we have i_{h-1} internal nodes at or below level $h-1$. Further, as $2i_{h-1} \leq n$ we have sufficient characters to construct this forest).
Now, let us assume there is a valid forest corresponding to the sequence $\mathcal{I}' = \langle i_{k+1}, \cdots i_h = 0\rangle$. Note that this forest has $i_{k+1} - i_{k+2} > 0$ trees. We now add another $(2i_k - i_{k+1}) - (2i_{k+1} - i_{k+2})$ leaves (characters) at level $k+1$ and construct a forest with $i_k - i_{k-1}$ trees, having a total of i_k internal nodes. Note that $(2i_k - i_{k+1}) - (2i_{k+1} - i_{k+2}) \geq 0$ and $(2i_k - i_{k+1}) \leq n$, hence, we have sufficient characters to create such a forest. This proves the theorem.

C Proof of Lemma 14 pertaining to Exact DP for Gen-LLHC

Proof.

$$F(T) = \sum_{\ell=0}^{h-1} \hat{f}(\ell+1) \cdot (S_{2i_\ell - i_{\ell+1}})$$

By definition of $F(T)$ we have

$$F(T) \quad = \quad \sum_{c \in C} (freq(c) \cdot f(d_T(c)))$$

By using the definition of $\hat{f}(i)$ we get

$$F(T) = \sum_{c \in C} \left(freq(c) \cdot \left(\sum_{i \leq d_T(c)} \hat{f}(i) \right) \right)$$

By rearranging the summation over each level and using proposition 7 we get

$$F(T) = \sum_{\ell=0}^{h-1} \left(\hat{f}(\ell+1) \cdot \sum_{j=1}^{2i_\ell - i_{\ell+1}} freq(j) \right)$$

and using Theorem 10 we get

$$F(T) = \sum_{\ell=0}^{h-1} \hat{f}(\ell+1) \cdot S_{2i_\ell - i_{\ell+1}}$$

This completes the proof of the Lemma. ◀

D Pseudo-code for PTAS for Gen-LLHC: Theorem 3(b)

We present the Pseudo-code for PTAS for Gen-LLHC in Algorithm 2.

Algorithm 2 (for **Theorem 3(b)**).

```
Input: Weighted Alphabet $C = \{c_1, c_2, \ldots, c_n\}$; Penalty bound $\mathcal{P}$; penalty function
       p(.); objective function $f(.)$; function $\hat{f}(.)$ defined over $f(.)$; Approximation
       constant $\epsilon$
Output: Prefix tree having penalty $\leq \mathcal{P}$ and objective value less than $\leq (1+\epsilon) \cdot C^*$
$C_{PTAS} \leftarrow \infty$
for $val \leftarrow 0$ to $\log_2(\mathcal{F} \cdot f(n))$ do
    $\mathcal{C} \leftarrow 2^{val}$
    $\lambda = \lfloor (\epsilon \cdot \mathcal{C})/2h \rfloor$
    for $\ell \leftarrow 0$ to $n$ do
        for $b \leftarrow 0$ to $((2h/\epsilon) + h)$ do
            $C = b \cdot \lambda$
            $\overline{D}(0, \ell, C) := 0$
    for $i \leftarrow 1$ to $(n-1)$ do
        for $\ell \leftarrow (h-1)$ downto $0$ do
            for $b \leftarrow 0$ to $((2h/\epsilon) + h)$ do
                $C = b \cdot \lambda$
                $bestPenalty := \infty$
                for $j \leftarrow \max(0, 2i-n)$ to $i-1$ do
                    $Penalty := \infty$
                    $C' = C - \mathbf{r}(p(\ell+1) \cdot S_{2i-j})$
                    $k :=$ recursive index where $\overline{D}(j, \ell+1, C')$ was minimized
                    if $2i - j < 2j - k$ then
                        continue;
                    if $0 \leq C'$ then
                        $Penalty := \overline{D}(j, \ell+1, C') + \hat{f}_{(l+1)} \cdot S_{2i-j}$
                    if $Penalty < bestPenalty$ and $Penalty < \mathcal{P}$ then
                        $bestPenalty := Penalty$
                $\overline{D}(i, \ell, C) := bestPenalty$
    $C_{PTAS} := \min_C(\overline{D}(n-1, 0, C)$ corresponds to a valid prefix tree$)$
    if $C_{PTAS} \leq \mathcal{C}$ then
        break
return $C_{PTAS}$;
```

E Proof of Lemma 15 pertaining to PTAS for Gen-LLHC

Proof. For any valid values of i, ℓ and P:

$$D(i,\ell,C) \;\geq\; \overline{D}(i,\ell,\mathbf{r}(C)+(h-\ell)\cdot\lambda)$$

In our proof we will be using the following fact:

▶ **Proposition 24.** *Let C' and C'' be multiples of λ. Then, $\overline{D}(z,\ell+1,C') \geq \overline{D}(z,\ell+1,C'')$ whenever $C' \leq C''$.*

This proposition holds because the best solution having objective value at most C' is also a candidate solution having objective value at most C'' (other parameters remaining same).

We now prove the lemma by induction on the value of ℓ decreasing from h to 0.
For $\ell = h$: From our initialization, the entries of D and $\overline{D}$ are all initialized to 0 for $\ell = h$ and hence the claim trivially holds.
For $\ell < h$: Consider $D(i,\ell,C)$. From recurrence (4), there must be some choice of j for which $D(i,\ell,C)$ is minimized. Let z be that choice of j, i.e.,

$$D(i,\ell,C) = D(z,\ell+1,C-\hat{f}_{\ell+1}\cdot S_{2i-z}) + \hat{p}_{\ell+1}\cdot S_{2i-z}$$

Now we obtain the following relations:

$$\begin{aligned}
&D(i,\ell,C)\\
&\;= D(z,\ell+1,C-\hat{f}_{\ell+1}\cdot S_{2i-z}) + \hat{p}_{\ell+1}\cdot S_{2i-z}\\
&\;\geq \overline{D}(z,\ell+1,\mathbf{r}(C-\hat{f}_{\ell+1}\cdot S_{2i-z})+(h-(\ell+1))\lambda) + \hat{p}_{\ell+1}\cdot S_{2i-z}\\
&\;\geq \overline{D}(z,\ell+1,\mathbf{r}(C)-(\mathbf{r}(\hat{f}_{\ell+1}\cdot S_{2i-z})-\lambda)+(h-(\ell+1))\lambda) + \hat{p}_{\ell+1}\cdot S_{2i-z}\\
&\;= \overline{D}(z,\ell+1,\mathbf{r}(C)-\mathbf{r}(\hat{f}_{\ell+1}\cdot S_{2i-z})+(h-\ell)\lambda) + \hat{p}_{\ell+1}\cdot S_{2i-z}\\
&\;\geq \overline{D}(i,\ell,\mathbf{r}(C)+(h-\ell)\lambda)
\end{aligned}$$

where the first inequality follows by induction, the second inequality follows from Proposition 24 and the last inequality follows from the fact that $\overline{D}(z,\ell+1,\mathbf{r}(C)-\mathbf{r}(\hat{f}_{\ell+1}\cdot S_{2i-z})+(h-\ell)\lambda)+\hat{p}_{\ell+1}\cdot S_{2i-z}$ is also a candidate for consideration in recurrence (5) for $\overline{D}(i,\ell,\mathbf{r}(C)+(h-\ell)\lambda)$.

This completes the proof of the Lemma. ◀

F Exact DP for COPT: Proof of Theorem 1(a)

There exists a dynamic program algorithm to solve the COPT($\mathcal{P}$) problem that runs in time $O(n^{2+m})$ for m block levels.

The dynamic programming algorithm is similar to that for the exact algorithm with the main difference being that instead of iterating over lengths we iterate over the decode times of the tree.

Let $\tilde{D}(i,\ell,T)$ denote the minimum codelength amongst all forests rooted at level ℓ, having i internal nodes with decode time at most T. Also, define $\mathcal{T} = \sum_{c=1}^{n} \cdot \sum_{i=1}^{m} q_i$ (here, the decode time of the forest is the sum of the decode times of the trees in the forest)

For a fixed number of block levels, m, the following lemma holds:

▶ **Lemma 25.** *The number of possible values of decode time for the forests rooted at some level is n^{m-1}.*

Proof. Let there be x_i characters in the ith block level $\forall i \in [1, m]$ and x_0 be the number of characters which are not present in the forest corresponding to $\tilde{D}(i, \ell, T)$. These characters corresponding to x_0 will not have any decode time contribution for T. We have $\sum_{i=0}^{m} x_i = n$. For $l > w_1$, the width of first block, we know that x_1 is zero. When $l \leq w_1$, we know that x_0 is zero. That is not both of x_0, x_1 can be non-zero. Hence, there are $O(n^{m-1})$ possible sequences of x_i's satisfying this. For each sequence of x_i's, we can uniquely determine the decode time value. Hence there are $O(n^{m-1})$ possible decode time values. ◀

A dynamic program using the above recurrence can be designed as follows:

Base Case. For all forests with no merges, i.e. no internal nodes, we initialize the decode time to 0, i.e.,

$$\forall\ \ell \in [0, n]\ \textit{and}\ T \in [0, \mathcal{T}]: \quad \tilde{D}(0, \ell, T) = 0$$

Inductive Case. To compute $\tilde{D}(i, \ell, T)$, we iterate over the number of internal nodes that are at depth strictly greater than ℓ. If j internal nodes are at depth strictly greater than ℓ, then there are $(2i - j)$ characters at depth strictly greater than ℓ, then $q_{\ell+1} \cdot P_{2i-j}$ is the decode-time contribution of all the characters having level $> \ell$, due to the access at level $(\ell + 1)$. Furthermore, $\tilde{D}(j, \ell + 1, T')$ denotes the decode time contributed by all accesses made at depths greater than $\ell + 1$. This yields the following recurrence:

$$\tilde{D}(i, \ell, T) = \min_{\substack{j \in [max(0, 2i-n), i-1] \\ \&\ 2i-j\ \geq\ 2j-k}} \left\{ \tilde{D}(j, \ell + 1, T') + P_{2 \cdot i - j} \right\} \tag{6}$$

where $T' = T - \hat{q}_{lvl+1} \cdot P_{2 \cdot i - j}$ and k is the recursive index using which $\tilde{D}(j, \ell + 1, T')$ was populated. We only need to recursively check if $T' > 0$. The tree with the optimal decode time can be obtained by maintaining the parent pointers of each update and then backtracking.

Note that we only update entries of $\tilde{D}$ for which the T parameter corresponds to a sequence of $\langle\ x_0, x_1, x_2, \ldots, x_m\ \rangle$, from lemma 25. The decode time for a sequence $\langle x_0, x_1, x_2, \ldots, x_m \rangle$ is $Dec(\langle x_i \rangle) = \sum_{i=1}^{m} x_i \cdot \hat{q}_i$

From Lemma 25, we know that the T parameter will take on only $O(n^{m-1})$ values. The table can accordingly be compressed and maintained only for these entries, however we omit these implementation details in the interest of better exposition.

After the DP is filled, we check all the entries of the form $\tilde{D}(n - 1, 0, t)$ which have code length parameter $t \leq \mathcal{P}$ and find the the optimal code length corresponding to it.

Time complexity. We note that i and ℓ can take n possible values each and T takes n^{m-1} possible values. So, there are n^{m+1} cells in the DP. Each cell can be filled in at most $O(n)$ time. So, the time complexity of the DP is $O(n^{m+2})$. Checking the DP table to find the optimal decode time will take $O(n^{m+1})$ time. Hence, we can solve the $COPT$ problem in $O(n^{m+2})$ time when the number of block levels is a constant m.

G Proof of Proposition 20 pertaining to Theorem 1(b)

Proof. We prove this using induction. The base case holds from Lemma 18. Consider the two nodes at level one.

We first consider the case where not all k highest frequencies are in the same sub-tree rooted at one of the nodes. By induction assumption, in the sub-tree in which kth highest frequency is present, the kth highest frequency is at level at most $k - 1 + \lceil \log(n) \rceil$. Therefore given holds.

We now consider the case where all k highest frequencies are in the same sub-tree rooted at one of the nodes. Let the highest frequency in the other sub tree be $k + r'$th frequency for some $r' > 0$. If $k + r'$th frequency is at level at most $k + \lceil \log(n) \rceil$, since higher frequencies are at a lower level, kth highest frequency is at level at most $k + \lceil \log(n) \rceil$. If not, the subtree has more than $2^{k+\lceil \log(n) \rceil - 1} > n - 1$ nodes. Contradiction. ◀

Approximation Algorithms for Flexible Graph Connectivity

Sylvia Boyd
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

Joseph Cheriyan
Department of Combinatorics and Optimization, University of Waterloo, Canada

Arash Haddadan
Warner Music Group, New York, NY, USA

Sharat Ibrahimpur
Department of Combinatorics and Optimization, University of Waterloo, Canada

Abstract

We present approximation algorithms for several network design problems in the model of Flexible Graph Connectivity (Adjiashvili, Hommelsheim and Mühlenthaler, "Flexible Graph Connectivity", *Math. Program.* pp. 1–33 (2021), *IPCO* 2020: pp. 13–26). In an instance of the Flexible Graph Connectivity (FGC) problem, we have an undirected connected graph $G = (V, E)$, a partition of E into a set of safe edges $\mathcal{S}$ and a set of unsafe edges $\mathcal{U}$, and nonnegative costs $\{c_e\}_{e \in E}$ on the edges. A subset $F \subseteq E$ of edges is feasible for FGC if for any unsafe edge $e \in F \cap \mathcal{U}$, the subgraph $(V, F \setminus \{e\})$ is connected. The algorithmic goal is to find a (feasible) solution F that minimizes $c(F) = \sum_{e \in F} c_e$. We present a simple 2-approximation algorithm for FGC via a reduction to the minimum-cost r-out 2-arborescence problem. This improves upon the 2.527-approximation algorithm of Adjiashvili et al.

For integers $p \geq 1$ and $q \geq 0$, the (p, q)-FGC problem is a generalization of FGC where we seek a minimum-cost subgraph $H = (V, F)$ that remains p-edge connected against the failure of any set of at most q unsafe edges; that is, for any set $F' \subseteq \mathcal{U}$ with $|F'| \leq q$, $H - F' = (V, F \setminus F')$ should be p-edge connected. Note that FGC corresponds to the $(1, 1)$-FGC problem. We give approximation algorithms for two important special cases of (p, q)-FGC: (a) Our 2-approximation algorithm for FGC extends to a $(k + 1)$-approximation algorithm for the $(1, k)$-FGC problem. (b) We present a 4-approximation algorithm for the $(k, 1)$-FGC problem.

For the unweighted FGC problem, where each edge has unit cost, we give a 16/11-approximation algorithm. This improves on the result of Adjiashvili et al. for this problem.

The (p, q)-FGC model with $p = 1$ or $q \leq 1$ can be cast as the *Capacitated k-Connected Subgraph* problem which is a special case of the well-known Capacitated Network Design problem. We denote the former problem by Cap-k-ECSS. An instance of this problem consists of an undirected graph $G = (V, E)$, nonnegative integer edge-capacities $\{u_e\}_{e \in E}$, nonnegative edge-costs $\{c_e\}_{e \in E}$, and a positive integer k. The goal is to find a minimum-cost edge-set $F \subseteq E$ such that every (non-trivial) cut of the capacitated subgraph $H(V, F, u)$ has capacity at least k. We give a $\min(k, 2 \max_{e \in E} u_e)$-approximation algorithm for this problem.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation Algorithms, Combinatorial Optimization, Network Design, Edge-Connectivity of Graphs, Reliability of Networks

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.9

Funding *Joseph Cheriyan*: Supported in part by NSERC, RGPIN-2019-04197.
Arash Haddadan: This work was mostly done when this author was a postdoctoral researcher at the Biocomplexity Institute and Initiative at the University of Virginia, Charlottesville, and supported by NSF Expeditions in Computing Grant with award number CCF-1918656.
Sharat Ibrahimpur: Supported in part by NSERC grant 327620-09.

Acknowledgements We thank the anonymous reviewers and PC members for their comments.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 9; pp. 9:1–9:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Network design and graph connectivity are core topics in Theoretical Computer Science and Operations Research. A basic problem in network design is to find a minimum-cost subnetwork H of a given network G such that H satisfies some specified connectivity requirements. Most of these problems are NP-hard. Several important algorithmic paradigms were developed in the context of these topics, ranging from exact algorithms for the shortest (s,t)-path problem and the minimum spanning tree (MST) problem to linear programming-based approximation algorithms for the survivable network design problem and the generalized Steiner network problem. Network design problems are often motivated from practical considerations such as the design of fault-tolerant supply chains, congestion control for urban road traffic, and the modeling of epidemics (see [11, 12, 15]).

Recently, Adjiashvili, Hommelsheim and Mühlenthaler [1, 2] introduced a new model called *Flexible Graph Connectivity* (FGC), that is motivated by research in robust optimization. In an instance of FGC, we have an undirected connected graph $G = (V, E)$ on n vertices, a partition of E into safe edges $\mathcal{S}$ and unsafe edges $\mathcal{U}$, and nonnegative costs $\{c_e\}_{e\in E}$ on the edges. The graph G may have multiedges, but no self-loops. A subset $F \subseteq E$ of edges is feasible for FGC if for any unsafe edge $e \in F \cap \mathcal{U}$, the subgraph $(V, F \setminus \{e\})$ is connected. The problem is to find a (feasible) solution F minimizing $c(F) = \sum_{e\in F} c_e$. The motivation for studying FGC is two-fold. First, FGC generalizes many well-studied survivable network design problems. Notably, the problem of finding a minimum-cost 2-edge connected spanning subgraph (abbreviated as 2ECSS) corresponds to an instance of FGC where all edges are unsafe, and the MST problem corresponds to an instance of FGC where all edges are safe. Second, FGC captures a non-uniform model of survivable network design problems where a subset of edges never fail, i.e., they are always safe.

The notion of (p, q)-FGC is an extension of the basic FGC model where we have two additional integer parameters p and q satisfying $p \geq 1$ and $q \geq 0$. A subset $F \subseteq E$ of edges is feasible for (p, q)-FGC if the spanning subgraph $H = (V, F)$ is p-edge connected, and moreover, the deletion of any set of at most q unsafe edges of F preserves p-edge connectivity. In other words, each nontrivial cut $(S, V \setminus S)$ of H either contains p safe edges or contains $p+q$ (safe or unsafe) edges. Note that the FGC problem is the same as the $(1, 1)$-FGC problem. The algorithmic goal is to find a feasible edge-set F of minimum cost. The (p, q)-FGC problem is a natural and fundamental question in robust network design. It can be seen as a way of interpolating between p-edge connectivity (when all edges are safe) and $(p + q)$-edge connectivity (when all edges are unsafe). We remark that for all problems considered in this work, we are only allowed to use at most one copy of an edge; multiedges may arise in F due to multiedges in G.

One of our goals is to give approximation algorithms for important special cases of (p, q)-FGC. Since FGC generalizes the 2ECSS problem, it is already APX-hard (see [5]), so a polynomial-time approximation scheme is ruled out unless P=NP. In the following we sketch a simple randomized $O(q \log n)$-approximation algorithm for (p, q)-FGC under some assumptions. For simplicity, assume that p and q are such that $q/p \leq \alpha$ for an absolute constant $\alpha \geq 0$. Let F^* denote an optimal solution to the given (p, q)-FGC instance. To start with, let $H = (V, F)$ denote a 2-approximate p-edge connected spanning subgraph (abbreviated p-ECSS) of the graph G with edge-costs $\{c_e\}_{e\in E}$, where we make no distinction between safe and unsafe edges; such an H can be found in polynomial-time by using (say) Jain's iterative rounding algorithm [8]. Note that $c(F) \leq 2c(F^*)$, since (V, F^*) is a p-edge connected spanning subgraph of G.

We say that a nonempty set $R \subsetneq V$ is "deficient" if $|\delta(R) \cap F| < p+q$ and $|\delta(R) \cap F \cap \mathcal{S}| < p$; thus, $R \subsetneq V$ is deficient if the cut $\delta(R)$ has less than $(p+q)$ F-edges and has less than p safe F-edges. Let $\mathcal{C}$ denote the family of all deficient sets. Note that deficient sets are the only obstructions to the (p,q)-FGC-feasibility of F. We fix deficient sets by performing a sequence of at most q augmentation iterations, where in each iteration we augment F with an edge-set $F' \subseteq E \setminus F$ such that $\delta(R) \cap F' \neq \emptyset$ for each $R \in \mathcal{C}$. We compute the desired F' via a reduction to the weighted set cover problem. First, let us state an upper bound on $|\mathcal{C}|$. For every $R \in \mathcal{C}$, observe that $\delta(R)$ is a $(1+\alpha)$-approximate min-cut in H, because H is p-edge connected (i.e., the size of a min-cut of H is $\geq p$). By Karger's bound [9], we have $|\mathcal{C}| \leq n^{2\alpha+2}$. In fact, with probability at least $1 - 1/n$, we can explicitly compute $\mathcal{C}$ in time polynomial in n^α. For simplicity, assume that we have explicit access to $\mathcal{C}$. Consider an instance of the weighted set cover problem where we want to cover elements of $\mathcal{C}$ by using sets of the form $\{\mathcal{R}_e\}_{e \in E \setminus F}$, where $\mathcal{R}_e := \{R \in \mathcal{C} : e \in \delta(R)\}$, and the weight of $\mathcal{R}_e$ is c_e. (Informally speaking, we have a ground-set of "points" that correspond to elements of $\mathcal{C}$, i.e., the deficient sets, we have a weighted set $\mathcal{R}_e$ corresponding to each edge $e \in E \setminus F$, and the goal is to pick a min-weight family of sets $\mathcal{R}_e$ whose union contains all the "points".) Since $F \cup F^*$ is feasible for the given (p,q)-FGC-instance, $\{\mathcal{R}_e\}_{e \in F^* \setminus F}$ is a feasible solution to the set-cover instance with cost at most $c(F^*)$. The well-known greedy algorithm for weighted set cover (see Theorem 13.3 in [16]) finds an $F' \subseteq E \setminus F$ satisfying $\delta(R) \cap F' \neq \emptyset$ for all $R \in \mathcal{C}$ and $c(F') \leq O(\alpha \log n) c(F^*)$. We augment F to $F \cup F'$ and discard the sets $R \in \mathcal{C}$ that are no longer deficient w.r.t. the augmented F. We repeatedly apply such augmenting iterations until $\mathcal{C}$ is empty. There are at most q such iterations, because each iteration increases the cardinality of $\delta(R) \cap F$ by one or more for each $R \in \mathcal{C}$. We summarize this discussion by the next claim.

▷ Claim. There is a randomized polynomial-time $O(q \log n)$-approximation algorithm for the special case of (p,q)-FGC where $q/p \leq O(1)$.

The (p,q)-FGC model is related to the model of Capacitated Network Design. There are several results pertaining to approximation algorithms for various problems in Capacitated Network Design, for example, see Goemans et al. [6] and Chakrabarty et al. [3]. A well-studied problem in this area that is relevant to us is the Capacitated k-Connected Subgraph problem, see [3]. We denote this problem by Cap-k-ECSS. Formally, in an instance of this problem, we have an undirected multigraph $G = (V, E)$, nonnegative integer edge-capacities $\{u_e\}_{e \in E}$, nonnegative edge-costs $\{c_e\}_{e \in E}$, and a positive integer k. The goal is to find an edge-set $F \subseteq E$ such that for any nonempty $R \subsetneq V$ we have $\sum_{e \in \delta(R) \cap F} u_e \geq k$, and $c(F)$ is minimized. Let n and m denote the number of vertices and edges of G, respectively. For this problem, Goemans et al. [6] give a $\min(2k, m)$-approximation algorithm, and Chakrabarty et al. [3] give a randomized $O(\log n)$-approximation algorithm.

In general, (p,q)-FGC and Cap-k-ECSS models are incomparable (see below for more details), however, when $p = 1$ or $q \leq 1$ holds, then (p,q)-FGC can be cast as an instance of the Cap-k-ECSS problem. The usual k-ECSS problem corresponds to the Cap-k-ECSS problem with unit edge capacities. The FGC problem corresponds to the Cap-2-ECSS problem, where safe edges have capacity 2 and unsafe edges have capacity 1. More generally, $(1,k)$-FGC corresponds to the Cap-$(k+1)$-ECSS problem, where safe edges have capacity $k+1$ and unsafe edges have capacity 1, and $(k,1)$-FGC corresponds to the Cap-$(k(k+1))$-ECSS problem where safe edges have capacity $k+1$ and unsafe edges have capacity k. We remark that the most general models of (p,q)-FGC and Cap-k-ECSS are incomparable. In particular, it is easy to see that the (p,q)-FGC problem is not the same as the Cap-$(p(p+q))$-ECSS problem where safe edges have a capacity of $p+q$ and unsafe edges have a capacity of p. For instance, take $p = 2$ and $q = 3$: a cut with one safe edge (of capacity 5) and three unsafe edges (each with capacity 2) has total capacity $11 \geq p(p+q)$, but such a cut is deficient in the $(2,3)$-FGC model.

Our Contributions. We mention the main contributions of this work along with a brief overview of our results and techniques.

Our first result is a simple reduction from FGC to the well-known minimum-cost 2-arborescence problem that achieves an approximation guarantee of two. This result matches the current best approximation guarantee known for the 2ECSS problem, and improves on the 2.527-approximation algorithm of [2]. At a high level, our result is based on a straightforward extension of the 2-approximation algorithm of Khuller and Vishkin [10] for the 2ECSS problem. (In fact, Khuller and Vishkin [10] give a simple reduction from the k-ECSS problem to the problem of computing a minimum-cost k-arborescence in a digraph that achieves an approximation guarantee of two.)

▶ **Theorem 1.** *There is a* 2*-approximation algorithm for* FGC*.*

The following result generalizes Theorem 1 to the $(1,k)$-FGC problem, where we want to find a min-cost spanning subgraph that remains connected against the failure of any set of at most k unsafe edges.

▶ **Theorem 2.** *There is a* $(k+1)$*-approximation algorithm for* $(1,k)$-FGC*.*

Our proof of Theorem 2 is based on a reduction from $(1,k)$-FGC to the minimum-cost $(k+1)$-arborescence problem (see [13], Chapters 52 and 53). We lose a factor of $k+1$ in this reduction.

In Section 3, we consider the unweighted version of FGC, where each edge has unit cost. We design improved approximation algorithms for this special case.

▶ **Theorem 3.** *There is a* $\frac{16}{11}$*-approximation algorithm for unweighted* FGC*.*

In Section 4, we consider the $(k,1)$-FGC problem, where we seek a min-cost spanning subgraph that is k-edge connected against failure of at most one unsafe edge. Our main contribution here is the following.

▶ **Theorem 4.** *There is a* 4*-approximation algorithm for* $(k,1)$-FGC*.*

Our algorithm in Theorem 4 runs in two stages. In the first stage we pretend that all edges are safe. Under this assumption, $(k,1)$-FGC simplifies to the k-ECSS problem, for which several 2-approximation algorithms are known. Let $H = (V,F)$ be the k-edge connected spanning subgraph found in Stage 1. In the second stage, our goal is to preserve k-edge connectivity against the failure of any one unsafe edge. In the graph H, consider a cut that has (exactly) k edges and that contains at least one unsafe edge. Such a cut, that we call *deficient*, certifies that F is not feasible for $(k,1)$-FGC, so it needs to be augmented. The residual problem is that of finding a cheapest augmentation of F on all deficient cuts. It turns out that this cut-augmentation problem can be formulated as the f-connectivity problem for an uncrossable function f (to be defined in Section 4). Williamson, Goemans, Mihail and Vazirani [17] present a 2-approximation algorithm for the latter problem.

Lastly, in Section 5, we consider the Capacitated k-Connected Subgraph problem that we denote by Cap-k-ECSS. For notational convenience, let $u_{max} := \max\{u_e : e \in E\}$ denote the maximum capacity of an edge in the given instance of Cap-k-ECSS; similarly, let $u_{min} := \min\{u_e : e \in E\}$. Our main result in Section 5 is the following.

▶ **Theorem 5.** *There is a* $\min(k, 2u_{max})$*-approximation algorithm for the* Cap-k-ECSS *problem.*

Similar to Theorems 1 and 2, our proof of Theorem 5 is based on a reduction from the Cap-k-ECSS problem to the minimum-cost k-arborescence problem. The factor m in the $\min(2k, m)$ approximation guarantee of Goemans et al. comes from the fact that a simple greedy strategy yields an m-approximation for the Cap-k-ECSS problem. Assuming $\min(k, 2u_{max}) \leq m$, our result has a better dependence on k, and, in fact, for the standard case of $u_{min} = 1$, $u_{max} = k \ll m$, no previous result achieves an approximation guarantee of k (to the best of our knowledge). Our result above is incomparable to the result in [3]: our approximation guarantee is independent of the graph size, whereas their result is independent of k. The algorithm in [3] is probabilistic and its analysis is based on Chernoff tail bounds.

Theorem 5 provides the following approximation guarantees for special cases of (p, q)-FGC:

(i) for $(1, 1)$-FGC, $k = u_{max} = 2$, so Theorem 5 gives a 2-approximation (same as Theorem 1);

(ii) for $(1, q)$-FGC, $k = u_{max} = q + 1$, so Theorem 5 gives a $(q + 1)$-approximation (same as Theorem 2); and

(iii) for $(p, 1)$-FGC with $p > 1$, $k = p(p + 1)$ and $u_{max} = p + 1$, so Theorem 5 gives a $2(p + 1)$-approximation (this is weaker than the 4-approximation given by Theorem 4).

2 A $(k + 1)$-Approximation Algorithm for $(1, k)$-FGC

We give a $(k + 1)$-approximation for $(1, k)$-FGC, where k is a positive integer. The 2-approximation for FGC (Theorem 1) follows as a special case. Recall that in an instance of $(1, k)$-FGC we have an undirected multigraph $G = (V, E)$ (with no self loops), a partition of $E = \mathcal{S} \sqcup \mathcal{U}$ into safe and unsafe edges, and nonnegative edge-costs $\{c_e\}_{e \in E}$. Our objective is to find a minimum-cost edge-set $F \subseteq E$ such that the subgraph (V, F) remains connected against failure of any k unsafe edges.

For a subgraph H of G and a nonempty vertex-set $S \subsetneq V$, we use $\delta_H(S)$ to denote the set of edges in H with exactly one endpoint in S, i.e., $\delta_H(S) := \{e = uv \in E(H) : |\{u, v\} \cap S| = 1\}$. We drop the subscript H when the underlying graph is clear from the context. The following characterization of $(1, k)$-FGC solutions is straightforward.

▶ **Proposition 6.** *F is feasible for $(1, k)$-FGC if and only if for all nonempty $S \subsetneq V$, the edge-set $F \cap \delta(S)$ contains a safe edge or $k + 1$ unsafe edges.*

For the rest of the paper, we assume that the given instance of $(1, k)$-FGC is feasible: this can be checked by computing a (global) minimum cut in G where we assign a capacity of $k+1$ to safe edges and a capacity of 1 to unsafe edges. As mentioned before, our algorithm for $(1, k)$-FGC is based on a reduction to the minimum-cost r-out $(k + 1)$-arborescence problem. We state a few standard results on arborescences. Let $D = (W, A)$ be a digraph and $\{c'_a\}_{a \in A}$ be nonnegative costs on the arcs. We remark that D may have parallel arcs but it has no self-loops. Let $r \in W$ be a designated root vertex. For a subgraph H of D and a nonempty vertex-set $S \subsetneq W$, we use $\delta^{\text{in}}_H(S)$ to denote the set of arcs in H such that the head of the arc is in S and the tail of the arc is in $W \setminus S$, i.e., $\delta^{\text{in}}_H(S) := \{a = (u, v) \in A(H) : u \notin S, v \in S\}$.

▶ **Definition 7** (r-out arborescence)**.** *An r-out arborescence (W, T) is a subgraph of D satisfying: (i) the undirected version of T is acyclic; and (ii) for every $v \in W \setminus \{r\}$, there is a directed path from r to v in the subgraph (W, T).*

In other words, an r-out arborescence is a directed spanning tree rooted out of r. More generally, an r-out k-arborescence is a union of k arc-disjoint r-out arborescences.

▶ **Definition 8** (r-out k-arborescence). *For a positive integer k, a subgraph (W, T) is an r-out k-arborescence if T can be partitioned into k arc-disjoint r-out arborescences.*

The following results on existence of arborescences and the corresponding optimization problem will be useful to us.

▶ **Theorem 9** ([13], Chapter 53.8). *Let $D = (W, A)$ be a digraph, $r \in W$ be a root vertex, and k be a positive integer. Then, D contains an r-out k-arborescence if and only if $|\delta_D^{\text{in}}(S)| \geq k$ for any nonempty vertex-set $S \subseteq V \setminus \{r\}$.*

▶ **Theorem 10** ([13], Theorem 53.10). *In strongly polynomial time, we can obtain an optimal solution to the minimum c'-cost r-out k-arborescence problem on D, or conclude that there is no r-out k-arborescence in D.*

The following claim is useful in our analysis.

▷ Claim 11. Let (W, T) be an r-out k-arborescence for an integer $k \geq 1$. Let $u, v \in W$ be two distinct vertices. Then, the number of arcs in T that have one endpoint at u and the other endpoint at v (counting multiplicities) is at most k.

Proof. Since an r-out k-arborescence is a union of k arc-disjoint r-out 1-arborescences, it suffices to prove the result for $k = 1$. The claim holds for $k = 1$ because the undirected version of T is acyclic, by definition. ◁

In our proofs we move from undirected graphs to their directed counterparts by bidirecting edges. We formalize this notion.

▶ **Definition 12** (Bidirected pair). *For an undirected edge $e = uv$, we call the arc-set $\{(u, v), (v, u)\}$ a bidirected pair arising from e.*

The following lemma shows how a $(1, k)$-FGC solution F can be used to obtain an r-out $(k + 1)$-arborescence (in an appropriate digraph) of cost at most $(k + 1)c(F)$.

▶ **Lemma 13.** *Let F be a $(1, k)$-FGC solution. Consider the digraph $D = (V, A)$ where the arc-set A is defined as follows: for each unsafe edge $e \in F \cap \mathcal{U}$, we include a bidirected pair of arcs arising from e, and for each safe edge $e \in F \cap \mathcal{S}$, we include $k + 1$ bidirected pairs arising from e. Consider the natural extension of the cost vector c to D where the cost of an arc $(u, v) \in A$ is equal to the cost of the edge in G that gives rise to it. Then, there is an r-out $(k + 1)$-arborescence in D with cost at most $(k + 1)c(F)$.*

Proof. Let (V, T) be a minimum-cost r-out $(k + 1)$-arborescence in D. First, we argue that T is well-defined. By Theorem 9, it suffices to show that for any nonempty $S \subseteq V \setminus \{r\}$, we have $|\delta_D^{\text{in}}(S)| \geq k + 1$. Fix some nonempty $S \subseteq V \setminus \{r\}$. By feasibility of F, $F \cap \delta(S)$ contains a safe edge or $k + 1$ unsafe edges (see Proposition 6). If $F \cap \delta(S)$ contains a safe edge $e = uv$ with $v \in S$, then by our choice of A, $\delta_D^{\text{in}}(S)$ contains $k + 1$ (u, v)-arcs. Otherwise, $F \cap \delta(S)$ contains $k + 1$ unsafe edges, and for each such unsafe edge uv with $v \in S$, $\delta_D^{\text{in}}(S)$ contains the arc (u, v). In both cases we have $|\delta_D^{\text{in}}(S)| \geq k + 1$, so T is well-defined.

We use Claim 11 to show that T satisfies the required bound on the cost. For each unsafe edge $e \in F$, T contains at most 2 arcs from the bidirected pair arising from e, and for each safe edge $e \in F$, T contains at most $k + 1$ arcs from the (disjoint) union of $k + 1$ bidirected pairs arising from e. Thus, $c(T) \leq 2\,c(F \cap \mathcal{U}) + (k + 1)\,c(F \cap \mathcal{S}) \leq (k + 1)\,c(F)$. ◀

Lemma 13 naturally suggests a reduction from $(1, k)$-FGC to the minimum-cost r-out $(k+1)$-arborescence problem. We prove the main theorem of this section.

Proof of Theorem 2. Fix some vertex $r \in V$ as the root vertex. Consider the digraph $D = (V, A)$ obtained from G as follows: for each unsafe edge $e \in \mathcal{U}$, we include a bidirected pair arising from e, and for each safe edge $e \in \mathcal{S}$, we include $k+1$ bidirected pairs arising from e. For each edge $e \in E$, let $R(e)$ denote the multi-set of all arcs in D that arise from $e \in E$. For any edge $e \in E$ (that could be one of the copies of a multiedge) and each of the corresponding arcs $\vec{e} \in R(e)$, we define $c_{\vec{e}} := c_e$. Let (V, T) denote a minimum c-cost r-out $(k+1)$-arborescence in D. By Lemma 13, $c(T) \leq (k+1)c(F^*)$, where F^* denotes an optimal $(1, k)$-FGC solution to the given instance.

We finish the proof by arguing that T induces a $(1, k)$-FGC solution F with cost at most $c(T)$. Let $F := \{e \in E : R(e) \cap T \neq \emptyset\}$. By definition of F and our choice of arc-costs in D, we have $c(F) \leq c(T)$. It remains to show that F is feasible for $(1, k)$-FGC. Consider a nonempty set $S \subseteq V \setminus \{r\}$. Since T is an r-out $(k+1)$-arborescence, by Theorem 9 we have $|\delta^{\text{in}}_T(S)| \geq k+1$. If $\delta^{\text{in}}_T(S)$ contains a safe arc (i.e., an arc that arises from a safe edge), then that safe edge belongs to $F \cap \delta(S)$. Otherwise, $\delta^{\text{in}}_T(S)$ contains some $k+1$ unsafe arcs (that arise from unsafe edges). Since both orientations of an edge cannot appear in $\delta^{\text{in}}_D(S)$, we get that $|F \cap \mathcal{U} \cap \delta(S)| \geq k+1$. By Proposition 6, F is a feasible solution for the given instance of $(1, k)$-FGC with $c(F) \leq (k+1)\text{OPT}$. ◀

3 Unweighted FGC

Consider the unweighted version of FGC where each edge has unit cost, i.e., $c_e = 1$ for all $e \in E$. We present a $\frac{16}{11}$-approximation algorithm (see Theorem 3); to the best of our knowledge, this is the first result that provides a better than $\frac{3}{2}$ approximation for unweighted FGC. Adjiashvili et al. [2] gave an $\left(\frac{\alpha}{2}+1\right)$-approximation algorithm for unweighted FGC, assuming the existence of an α-approximation algorithm for the unweighted 2ECSS problem: this implies a $\frac{5}{3}$-approximation algorithm for unweighted FGC by using the result of Sebö and Vygen [14]. The algorithm in [2] starts with a maximal forest of safe edges in the graph. At the end of this section, we give an example showing that no such algorithm can obtain an approximation factor better than $\frac{3}{2}$. Our main result in this section is the following.

▶ **Theorem 14.** *Suppose that there is an α-approximation algorithm for the unweighted* 2ECSS *problem. Then, there is a $\frac{4\alpha}{2\alpha+1}$-approximation algorithm for unweighted* FGC*.*

Theorem 3 follows from the above theorem by using the $\frac{4}{3}$-approximation algorithm of Sebö and Vygen [14] for the unweighted 2ECSS problem. Before delving into the proof of Theorem 14, we introduce some basic results on W-joins, which will be useful in our algorithm and its analysis. Let $G' = (V', E')$ be an undirected multigraph with no self-loops and let $\{c'_e\}_{e \in E'}$ be nonnegative costs on the edges.

▶ **Definition 15** (W-join)**.** *Let $W \subseteq V'$ be a subset of vertices with $|W|$ even. A subset $J \subseteq E'$ of edges is called a W-join if W is equal to the set of vertices of odd degree in the subgraph (V', J).*

The following classical result on finding a minimum-cost W-join is due to Edmonds.

▶ **Theorem 16** ([13], Theorem 29.1)**.** *In strongly polynomial time, we can obtain a minimum c'-cost W-join, or conclude that there is no W-join in G'.*

The W-join polytope is the convex hull of the incidence vectors of W-joins. Its dominant has a simple linear description.

▶ **Theorem 17** ([13], Corollary 29.2b). *The dominant of the W-join polytope is given by $\{x \in \mathbb{R}^{E'}_{\geq 0} : x(\delta_{G'}(S)) \geq 1 \,\forall\, S \subsetneq V'$ s.t. $|S \cap W|$ odd$\}$.*

Consider an instance of unweighted FGC consisting of a multigraph $G = (V, E = \mathcal{S} \cup \mathcal{U})$ with a specified partition of E into safe and unsafe edges. We will assume that G is connected and has no unsafe bridges, since otherwise the instance is infeasible. Let F^* denote an optimal solution. Suppose that we have access to an α-approximation algorithm for the 2ECSS problem. We give two algorithms for obtaining two candidate solutions to the given instance. We then argue that the cheaper of these two solutions is a $\frac{4\alpha}{2\alpha+1}$-approximate solution.

Join-based Algorithm for Unweighted FGC. Let T be a spanning tree in G that maximizes the number of safe edges. If $|T \cap \mathcal{S}| = |V| - 1$, then T is an optimal FGC solution for the given instance, and we are done. Otherwise, let $T' := T \cap \mathcal{U}$ be the (nonempty) collection of unsafe edges in T. Let $G' = (V', E')$ denote the graph obtained from G by contracting (safe) edges in $T \setminus T'$. We remove all self-loops from G', but retain parallel edges that arise due to edge contractions. Note that all edges in E' are unsafe and T' is a spanning tree of G'. Let W' denote the (nonempty) set of odd degree vertices in the subgraph (V', T'). Let $J' \subseteq E'$ be a minimum-cardinality W'-join in G', which we can compute in polynomial time by using Theorem 16. By our choice, the subgraph $(V', T' \sqcup J')$ is connected and Eulerian, so it is 2-edge connected in G'. Consider the multiset $F_1 = T \sqcup J'$ consisting of edges in E; if an edge e appears in both T' and J', then we include two copies of e in F_1.
If F_1 contains at most one copy of each edge in E, then F_1 is FGC-feasible. Otherwise, we modify F_1 to get rid of all duplicates without increasing $|F_1|$. Consider an unsafe edge $e \in E'$ that appears twice in F_1, i.e., e belongs to both T' and J'. We remove a copy of e from F_1. If this does not violate FGC-feasibility, then we take no further action. Otherwise, the second copy of e in F_1 is an unsafe bridge in (V, F_1) that induces a cut S in G. By our assumption that G has no unsafe bridges, there is another edge $e' \in E$ that is in $\delta(S)$ but not in F_1. We include e' in F_1. This finishes the description of our first algorithm.

At the end of the de-duplication step, F_1 is FGC-feasible and it contains at most one copy of any edge $e \in E$. It is also clear that $|F_1| \leq |T| + |J'|$. The following claim gives a bound on the quality of our first solution.

▷ Claim 18. We have $|J'| \leq \frac{1}{2}|F^* \cap \mathcal{U}|$. Hence, $|F_1| \leq |F^* \cap \mathcal{S}| + \frac{3}{2}|F^* \cap \mathcal{U}|$.

Proof. We prove the claim by constructing a fractional W'-join of small size. Recall that we chose T so that $T \setminus T'$ is a maximal safe forest in G, and we obtained G' by contracting connected components in $(V, T \setminus T')$. By our assumption that G has no unsafe bridges, we have that G' is 2-edge connected and consists of only unsafe edges. Let $B := F^* \cap E'$ denote the set of unsafe edges in the optimal solution F^* that also belong to G'. Consider the vector $z := \frac{1}{2}\chi^B$ where $\chi^B \in [0,1]^{E'}$ is the incidence vector of B in G'. Let S' be an arbitrary cut in G' and let S be the unique cut in G that gives rise to S' when we contract (safe) edges in $T \setminus T'$. Since F^* is FGC-feasible and there are no safe edges in $\delta_G(S)$, we must have $|B \cap \delta_{G'}(S')| \geq 2$. Consequently, $z(\delta_{G'}(S')) = \frac{1}{2}|B \cap \delta_{G'}(S')| \geq 1$. By Theorem 17, z lies in the dominant of the W'-join polytope, i.e., z dominates a fractional W'-join. Since J' is a min-cardinality W'-join, $|J'| \leq \mathbf{1}^T z \leq \frac{1}{2}|F^* \cap \mathcal{U}|$. We bound the size of F_1 by using the trivial bound $|T| \leq |F^*|$:

$$|F_1| \leq |F^*| + |J'| \leq |F^* \cap \mathcal{S}| + \frac{3}{2}|F^* \cap \mathcal{U}|. \qquad \triangleleft$$

The above claim shows that the size of F_1 can be charged to a certain combination of the number of safe and unsafe edges in F^*. Our second algorithm uses the α-approximation for the 2ECSS problem as a subroutine. The solution returned by this algorithm has the property that its size complements that of F_1.

2ECSS-based Algorithm for Unweighted FGC. Consider the multigraph G'' obtained from G by duplicating every safe edge in E. Similarly, let F'' be the multiedge-set obtained from F^* by duplicating every safe edge in F^*. Clearly, (V, F'') is a 2-edge connected subgraph of G'' consisting of $2|F^* \cap \mathcal{S}| + |F^* \cap \mathcal{U}|$ edges. Let F_2 be the output of running the α-approximation algorithm for the unweighted 2ECSS problem on G''. Since F_2 is 2-edge connected and only safe edges can appear more than once in F_2 (because G'' only has duplicates of safe edges), we can drop the extra copy of all safe edges while maintaining FGC-feasibility in G. This finishes the description of our second algorithm.

The following claim is immediate.

▷ Claim 19. We have $|F_2| \leq 2\alpha|F^* \cap \mathcal{S}| + \alpha|F^* \cap \mathcal{U}|$.

We end this section with the proof of our main result on unweighted FGC.

Proof of Theorem 14. Given an instance of unweighted FGC, we compute two candidate solutions F_1 and F_2 as given by the two algorithms described above. The solution F_1 can be computed using algorithms for the MST problem and the minimum-weight W'-join problem, followed by basic graph operations. The solution F_2 can be computed using the given α-approximation algorithm for the 2ECSS problem. We show that the smaller of F_1 and F_2 is a $\frac{4\alpha}{2\alpha+1}$-approximate solution for the given unweighted FGC-instance. By Claims 18 and 19:

$$\min(|F_1|, |F_2|) \leq \frac{2\alpha}{2\alpha+1}|F_1| + \frac{1}{2\alpha+1}|F_2| = \frac{4\alpha}{2\alpha+1}|F^*| \qquad ◀$$

As mentioned earlier, we have an example (see Figure 1 below) such that any algorithm for unweighted FGC that starts with a maximal forest on safe edges achieves an approximation guarantee of $\frac{3}{2}$ or more.

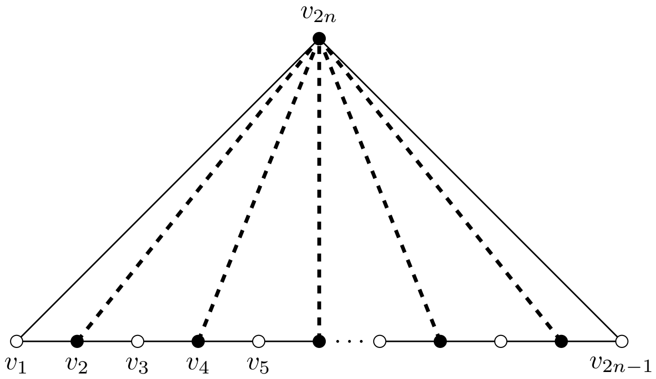


Figure 1 In this instance we have a graph on $2n$ vertices. The set of unsafe edges, shown using solid lines, forms a Hamiltonian cycle. For each $i = 1, \ldots, n-1$, there is a safe edge, shown using a thick dashed line, between v_{2i} and v_{2n}. The solution consisting of all unsafe edges is feasible, and any feasible solution must contain all unsafe edges, so OPT $= 2n$. Any feasible solution that contains a maximal forest on safe edges has size at least $3n-1$.

4 A 4-Approximation Algorithm for $(k,1)$-FGC

Our main result in this section is a 4-approximation algorithm for $(k,1)$-FGC (Theorem 4). Recall that in an instance of $(k,1)$-FGC, we have a multigraph $G=(V,E=\mathcal{S}\cup\mathcal{U})$ with a partition of the edge-set into safe and unsafe edges, nonnegative edge-costs $\{c_e\}_{e\in E}$, and a positive integer k. The objective is to find a minimum-cost subgraph that remains k-edge connected against the failure of any one unsafe edge. We remark that for the $k=1$ case, Theorem 1 yields a better approximation guarantee than Theorem 4. Let F^* denote an optimal solution to the given instance. The following characterization of $(k,1)$-FGC solutions is straightforward.

▶ **Proposition 20.** *F is feasible for $(k,1)$-FGC if and only if for all nonempty $S\subsetneq V$, the edge-set $F\cap\delta(S)$ contains k safe edges or $k+1$ edges.*

The above proposition suggests a two-stage strategy for $(k,1)$-FGC. Suppose that in the first stage we compute a cheap k-edge connected spanning subgraph $H_1=(V,F_1)$ of G without making any distinction between safe and unsafe edges. For any nonempty cut $S\subsetneq V$, we have $\delta_{H_1}(S)\geq k$, so by Proposition 20, the only hindrance to the $(k,1)$-FGC feasibility of F_1 are k-cuts in H_1 that contain at least one unsafe edge. We call such cuts *deficient*. The subproblem remaining for the second stage is an augmentation problem for these *deficient* cuts, which is special case of the (minimum-cost) f-connectivity problem.

In the *f-connectivity* problem we have an undirected multigraph $G'=(V',E')$, nonnegative edge-costs $\{c'_e\}_{e\in E'}$, and a cut-requirement function $f:2^{V'}\rightarrow\{0,1\}$ satisfying $f(\emptyset)=f(V)=0$. We assume access to f via a value oracle that takes as input a vertex-set $S\subseteq V$ and outputs $f(S)$. An edge-set $F\subseteq E'$ is feasible for the f-connectivity problem if $|F\cap\delta_{G'}(S)|\geq f(S)$ for every $S\subseteq V'$. In other words, F is feasible if and only if for every cut S with $f(S)=1$ there is at least one F-edge in this cut. The objective is to find a feasible $F\subseteq E'$ that minimizes $c(F)$. The f-connectivity problem can be modeled as an integer program whose linear relaxation (P) is stated below. For each edge $e\in E'$ the LP has a nonnegative variable x_e that models the extent to which the edge e is picked by the solution.

$$\begin{aligned} &\min \sum_{e\in E'} c'_e x_e && && \text{(P)}\\ &\text{subject to } x(\delta_{G'}(S))\geq 1 && \forall\, S\subseteq V' \text{ s.t. } f(S)=1\\ &\qquad x_e\geq 0 && \forall\, e\in E'. \end{aligned}$$

The f-connectivity problem has received a lot of attention in Combinatorial Optimization since it captures many well-known network design problems. In particular, it captures the generalized Steiner network problem. Williamson et al. [17] gave a primal-dual framework to obtain approximation algorithms for the f-connectivity problems when f is a proper function, and more generally, when f is an uncrossable function (also see the book chapter by Geomans and Williamson [7] for an excellent survey on primal-dual algorithms for network design problems).

▶ **Definition 21** (Uncrossable function). *A function $f:2^{V'}\rightarrow\{0,1\}$ is called uncrossable if $f(V')=0$ and f satisfies the following two conditions:*

(i) *f is symmetric, i.e., $f(S)=f(V'\setminus S)$ for all $S\subseteq V'$;*

(ii) *for any two sets $A,B\subseteq V'$ with $f(A)=f(B)=1$, either $f(A\cap B)=f(A\cup B)=1$ or $f(A\setminus B)=f(B\setminus A)=1$ holds.*

Under the assumption that *minimal violated sets* can be computed efficiently throughout the algorithm, the primal-dual algorithm of [17] gives a 2-approximation for the f-connectivity problem with an uncrossable function f. There is no explicit result in [17] that can be quoted verbatim and applied for our purposes, so we reference the most relevant lemma from their work.

▶ **Definition 22** (Minimal violated sets). *Let $f : 2^{V'} \to \{0,1\}$ be a cut-requirement function and $F \subseteq E'$ be an edge-set. A vertex-set $S \subseteq V'$ is said to be violated, w.r.t. f and F, if $f(S) = 1$ and $F \cap \delta_{G'}(S) = \emptyset$. We say that S is a minimal violated set if S is inclusion-wise minimal among all violated sets.*

▶ **Theorem 23** ([17], Lemma 2.1). *Let $f : 2^{V'} \to \{0,1\}$ be an uncrossable function that is given via a value oracle. Suppose that for any $F \subseteq E'$ we can compute all minimal violated sets (w.r.t. f and F) in polynomial time. We can compute a 2-approximate solution to the f-connectivity problem in polynomial time.*

We now describe a two-stage algorithm that produces a 4-approximate $(k,1)$-FGC solution in polynomial time, thereby proving Theorem 4.

Description of Our 4-Approximation Algorithm for $(k,1)$-FGC. Our algorithm runs in two stages. In the first stage, we compute a 2-approximate k-edge connected spanning subgraph $H_1 = (V, F_1)$ of G without making any distinction between safe and unsafe edges; since F^* is k-edge connected, the k-ECSS instance is feasible. This can be done using Jain's iterative rounding algorithm [8]. Next, we compute the collection $\mathcal{C} = \{S \subsetneq V : |\delta(S) \cap F_1| = k\}$ of all (minimum) k-cuts in H_1. Consider the cut-requirement function $f : 2^V \to \{0,1\}$ where $f(S)$ is 1 if and only if $S \in \mathcal{C}$ and $F_1 \cap \delta(S) \cap \mathcal{U} \neq \emptyset$. Consider an instance of the f-connectivity problem for the graph $G' := G - F_1$ with edge-costs $\{c_e\}_{e \in E \setminus F_1}$; note that $F^* \setminus F_1$ is feasible to this f-connectivity instance. In the second stage, we use Theorem 23 to compute a 2-approximate solution $F_2 \subseteq E \setminus F_1$ for this f-connectivity instance. We return the solution $F = F_1 \sqcup F_2$.

To prove Theorem 4, we need to argue the following: (i) f is uncrossable; (ii) we can compute minimal violated sets (w.r.t f and any $F' \subseteq E \setminus F_1$) in polynomial time; (iii) F is a feasible $(k,1)$-FGC solution; (iv) $c(F) \leq 4c(F^*)$; (v) the whole algorithm runs in polynomial time. We defer the proofs of (i) and (ii) to the end of this section. Assuming that they are true, (v) follows from Theorem 23. The following lemma covers (iii) and (iv).

▶ **Lemma 24.** *The edge-set F is feasible for $(k,1)$-FGC and satisfies $c(F) \leq 4c(F^*)$.*

Proof. We first argue that F is feasible. Since F_1 and F_2 are edge-disjoint, F is a subgraph of G. We use the characterization of feasible solutions given by Proposition 20. Let $S \subsetneq V$ be an arbitrary nonempty cut. Since $H_1 = (V, F_1)$ is a k-edge connected subgraph of G, we have $|F_1 \cap \delta(S)| \geq k$. If $|F_1 \cap \delta(S)| \geq k+1$, then $|F \cap \delta(S)| \geq k+1$, and we are done. Otherwise, S is a k-cut in H_1, i.e., $S \in \mathcal{C}$. If $F_1 \cap \delta(S)$ contains only safe edges, then $F \cap \delta(S)$ contains k safe edges, and we are done. Otherwise, by definition, $f(S) = 1$. Next, by feasibility of F_2 for f-connectivity, we have $F_2 \cap \delta(S) \neq \emptyset$. So, $|F \cap \delta(S)| = |F_1 \cap \delta(S)| + |F_2 \cap \delta(S)| \geq k+1$, and we are done.

We show that F is 4-approximate by arguing that $c(F_1)$ and $c(F_2)$ are bounded by $2c(F^*)$. The bound on $c(F_1)$ is immediate from the fact that F^* is feasible for the k-ECSS instance considered in Stage 1. Next, by feasibility of $F^* \setminus F_1$ for the f-connectivity instance, we have $c(F_2) \leq 2c(F^* \setminus F_1) \leq 2c(F^*)$, so we are done. ◀

▷ Claim 25. For any $F' \subseteq E \setminus F_1$, we can compute all minimal violated sets w.r.t. f and F'.

Proof. Since a graph on n vertices has at most $O(n^2)$ min-cuts [9], we have $|\mathcal{C}| = O(|V|^2)$. Using standard network flow algorithms, we can compute $\mathcal{C}$ in polynomial time (for instance, see [4]). Since we have explicit access to $\mathcal{C}$, we have a value oracle for f. Fix some $F \subseteq E \setminus F_1$. Any violated set must have $f(S) = 1$, so there are at most $|\mathcal{C}|$ many violated sets. We can exhaustively go through all violated sets and find the minimal elements. ◁

Lastly, we show that f is uncrossable.

▶ **Lemma 26.** *f is uncrossable.*

Proof. We check if the two properties of an uncrossable function hold for f (recall Definition 21); $f(V) = 0$ is trivial. Symmetry of f follows from symmetry of cuts in undirected graphs. To check the second property, consider nonempty $A, B \subsetneq V$ satisfying $f(A) = f(B) = 1$. By definition of f, in the subgraph $H_1 = (V, F_1)$, both A and B are (minimum) k-cuts with at least one unsafe edge on their respective boundaries. Let e_1 be an unsafe edge in $\delta_{H_1}(A)$ and let e_2 be an unsafe edge in $\delta_{H_1}(B)$. Let $r \in V$ be an arbitrary vertex. By symmetry of the cut function, we may assume without loss of generality that $r \notin A \cup B$. If $A \cap B = \emptyset$, then $f(A \setminus B) = f(B \setminus A) = 1$, so we are done. If $A \subseteq B$ or $A \supseteq B$, then $f(A \cap B) = f(A \cup B) = 1$, so we are done. Thus, we may assume that $A \cap B, V \setminus (A \cup B), A \setminus B, B \setminus A$ are all nonempty. By submodularity of the function $d(S) := |\delta_{H_1}(S)|$, we get:

$$|\delta_{H_1}(A \cap B)| = |\delta_{H_1}(A \cup B)| = |\delta_{H_1}(A \setminus B)| = |\delta_{H_1}(B \setminus A)| = k. \tag{1}$$

Furthermore, we also have:

$$F_1 \cap E(A \setminus B, B \setminus A) = \emptyset \quad \text{and} \quad F_1 \cap E(A \cap B, V \setminus (A \cup B)) = \emptyset, \tag{2}$$

where $E(S, T)$ denotes the set of edges in G with one endpoint in S and the other endpoint in T. We finish the proof by doing a case analysis on e_1 and e_2. By (2), exactly one of the following happens: (i) $e_1 \in E(A \setminus B, V \setminus (A \cup B))$; or (ii) $e_1 \in E(A \cap B, B \setminus A)$. If (i) happens, then $f(A \setminus B) = f(A \cup B) = 1$. Otherwise, $f(A \cap B) = f(B \setminus A) = 1$. We do a similar analysis on e_2. Exactly one of the following happens: (a) $e_2 \in E(B \setminus A, V \setminus (A \cup B))$; or (b) $e_2 \in E(A \cap B, A \setminus B)$. If (a) happens, then $f(B \setminus A) = f(A \cup B) = 1$. Otherwise, $f(A \cap B) = f(A \setminus B) = 1$. It is easy to verify that for each of the four combinations, we either have $f(A \cap B) = f(A \cup B) = 1$ or we have $f(A \setminus B) = f(B \setminus A) = 1$. ◀

5 The Capacitated k-Connected Subgraph Problem

In this section we consider the Cap-k-ECSS problem. We are given a multigraph $G = (V, E)$, nonnegative integer edge-capacities $\{u_e\}_e$, nonnegative edge-costs $\{c_e\}_e$, and a positive integer k. Our goal is to find a spanning subgraph $H = (V, F)$ such that for all nonempty sets $R \subsetneq V$ we have $\sum_{e \in \delta(R) \cap F} u_e \geq k$, and the cost $c(F)$ is minimized.

Given an instance of the Cap-k-ECSS problem, we may assume without loss of generality that $u_e \in \{1, \ldots, k\}$ for all $e \in E$ (we can drop edges with zero capacity and replace edge-capacities $\geq k + 1$ by k). We also assume that the instance is feasible. This can be verified in polynomial time by checking if G has any cut with capacity less than k. Let $u_{max} = \max_{e \in E} u_e$ denote the maximum capacity of an edge in G. Our main result in this section is a $\min(k, 2u_{max})$-approximation algorithm for the Cap-k-ECSS problem (Theorem 5); our algorithm is based on a reduction to the min-cost k arborescence problem.

Description of Our Algorithm for the Cap-k-ECSS Problem

Let $D = (V, A)$ be the directed graph obtained from G by replacing every edge $xy \in E$ by u_{xy} pairs of bidirected arcs $(x, y), (y, x)$, each with the same cost as the edge xy (thus, each edge e in G has $2u_e$ corresponding arcs in D, each of cost c_e). Designate an arbitrary vertex $r \in V$ as the root. By feasibility of the Cap-k-ECSS instance, we know that D contains an r-out k-arborescence (see Theorem 9). We use Theorem 10 on (D, c) to obtain a minimum-cost r-out k-arborescence T' in polynomial time. Let F' be the set of all edges $e \in E$ for which at least one of the corresponding $2u_e$ arcs in D appears in the optimal r-out k-arborescence T'.

▶ **Lemma 27.** *The edge-set F' obtained by the above algorithm is feasible for the given* Cap-k-ECSS *instance and it has cost at most $c(T')$.*

Proof. Let $R \subsetneq V \setminus \{r\}$ be an arbitrary nonempty vertex-set that excludes the root vertex r. Since T' contains k arc-disjoint r-out arborescences, $|\delta^{\text{in}}_{T'}(R)| \geq k$. For each edge $e \in E$, at most u_e of the corresponding arcs in D can appear in the set of T'-arcs entering R. Thus, $\sum_{e\in\delta(R)\cap F'} u_e \geq |\delta^{\text{in}}_{T'}(R)| \geq k$, and F' is a feasible solution for the Cap-k-ECSS instance, as required. For any edge $e \in E$, we only include a single copy of e in F' whenever any of the corresponding $2u_e$ arcs appear in T', so we have $c(F') \leq c(T')$. ◀

We now prove Theorem 5 by showing that F' is the desired $\min(k, 2u_{max})$-approximate solution.

Proof of Theorem 5. Let $(G(V, E), u, c, k)$ be a feasible instance of the Cap-k-ECSS problem. Let $D = (V, A)$ be the digraph and T' be the r-out k-arborescence as constructed by our algorithm. Let F^* be an optimal solution to the Cap-k-ECSS instance, and let $D^* = (V, A^*)$ be the digraph obtained from (V, F^*) by replacing every edge $xy \in F^*$ by u_{xy} pairs of bidirected arcs $(x, y), (y, x)$ each with the same cost as edge xy. Let $r \in V$ be the root vertex fixed by the algorithm. By feasibility of F^* (for the Cap-k-ECSS instance), we know that D^* contains an r-out k-arborescence. Let T^* denote an optimal r-out k-arborescence in D^*. Since D^* is a subgraph of D and T' is an optimal r-out k-arborescence in D, we have $c(T') \leq c(T^*)$. By Lemma 27, $c(F') \leq c(T')$, so to prove the theorem it suffices to argue that $c(T^*) \leq \min(t, 2u_{max})c(F^*)$. To this end, observe that for any edge $e \in F^*$ there can be at most $2u_e$ arcs in A^* by construction of D^*. Hence, $c(T^*) \leq c(A^*) \leq 2u_{max}c(F^*)$ holds. Next, by definition, T^* can be partitioned into k (arc-disjoint) r-out arborescences, each of which can use at most one of the $2u_e$ arcs corresponding to an edge e of G. It follows that for each edge $e \in F^*$ at most k of the corresponding $2u_e$ arcs can appear in T^*. Therefore, $c(T^*) \leq kc(F^*)$. This completes the proof. ◀

References

1 David Adjiashvili, Felix Hommelsheim, and Moritz Mühlenthaler. Flexible Graph Connectivity. In *Proceedings of the 21st Integer Programming and Combinatorial Optimization Conference*, volume 12125 of *Lecture Notes in Computer Science*, pages 13–26, 2020. doi:10.1007/978-3-030-45771-6_2.

2 David Adjiashvili, Felix Hommelsheim, and Moritz Mühlenthaler. Flexible Graph Connectivity. *Mathematical Programming*, pages 1–33, 2021. doi:10.1007/s10107-021-01664-9.

3 Deeparnab Chakrabarty, Chandra Chekuri, Sanjeev Khanna, and Nitish Korula. Approximability of Capacitated Network Design. *Algorithmica*, 72(2):493–514, 2015. doi:10.1007/s00453-013-9862-4.

4 Lisa Fleischer. Building Chain and Cactus Representations of All Minimum Cuts from Hao-Orlin in the Same Asymptotic Run Time. *Journal of Algorithms*, 33(1):51–72, 1999. doi:10.1006/jagm.1999.1039.

5 Harold N. Gabow, Michel X. Goemans, Éva Tardos, and David P. Williamson. Approximating the Smallest k-Edge Connected Spanning Subgraph by LP-rounding. *Networks*, 53(4):345–357, 2009. doi:10.1002/net.20289.

6 Michel X. Goemans, Andrew V. Goldberg, Serge A. Plotkin, David B. Shmoys, Éva Tardos, and David P. Williamson. Improved Approximation Algorithms for Network Design Problems. In *Proceedings of the 5th Symposium on Discrete Algorithms*, pages 223–232, 1994. doi:10.5555/314464.314497.

7 Michel X. Goemans and David P. Williamson. *The Primal-Dual Method for Approximation Algorithms and its Application to Network Design Problems*, chapter 4, pages 144–191. PWS Publishing Company, Boston, MA, 1997. URL: https://math.mit.edu/~goemans/PAPERS/book-ch4.pdf.

8 Kamal Jain. A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem. *Combinatorica*, 21(1):39–60, 2001. doi:10.1007/s004930170004.

9 David R. Karger. Global Min-Cuts in $\mathcal{RNC}$, and Other Ramifications of a Simple Min-Cut Algorithm. In *Proceedings of the 4th Symposium on Discrete Algorithms*, pages 21–30, 1993. doi:10.5555/313559.313605.

10 Samir Khuller and Uzi Vishkin. Biconnectivity Approximations and Graph Carvings. *Journal of the ACM*, 41(2):214–235, 1994. doi:10.1145/174652.174654.

11 Thomas L. Magnanti and Richard T. Wong. Network Design and Transportation Planning: Models and Algorithms. *Transportation Science*, 18(1):1–55, 1984. doi:10.1287/trsc.18.1.1.

12 Polina Rozenshtein, Aristides Gionis, B. Aditya Prakash, and Jilles Vreeken. Reconstructing an Epidemic Over Time. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1835–1844, 2016. doi:10.1145/2939672.2939865.

13 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag Berlin Heidelberg, 2003.

14 András Sebö and Jens Vygen. Shorter Tours by Nicer Ears: 7/5-Approximation for the Graph-TSP, 3/2 for the Path Version, and 4/3 for Two-Edge-Connected Subgraphs. *Combinatorica*, 34(5):597–629, 2014. doi:10.1007/s00493-014-2960-3.

15 Lawrence V. Snyder, Maria P. Scaparra, Mark S. Daskin, and Richard L. Church. *Planning for Disruptions in Supply Chain Networks*, pages 234–257. Institute for Operations Research and the Management Sciences (INFORMS), 2014. doi:10.1287/educ.1063.0025.

16 Vijay V. Vazirani. *Approximation Algorithms*. Springer Berlin Heidelberg, 2003. doi:10.1007/978-3-662-04565-7.

17 David P. Williamson, Michel X. Goemans, Milena Mihail, and Vijay V. Vazirani. A Primal-Dual Approximation Algorithm for Generalized Steiner Network Problems. *Combinatorica*, 15(3):435–454, 1995. doi:10.1007/BF01299747.

Tight Chang's-Lemma-Type Bounds for Boolean Functions

Sourav Chakraborty
Indian Statistical Institute, Kolkata, India

Nikhil S. Mande [1]
CWI, Amsterdam, The Netherlands

Rajat Mittal
Indian Institute of Technology, Kanpur, India

Tulasimohan Molli
Tata Institute of Fundamental Research, Mumbai, India

Manaswi Paraashar
Indian Statistical Institute, Kolkata, India

Swagato Sanyal
Indian Institute of Technology, Kharagpur, India

Abstract

Chang's lemma (Duke Mathematical Journal, 2002) is a classical result in mathematics, with applications spanning across additive combinatorics, combinatorial number theory, analysis of Boolean functions, communication complexity and algorithm design. For a Boolean function f that takes values in $\{-1, 1\}$ let $r(f)$ denote its Fourier rank (i.e., the dimension of the span of its Fourier support). For each positive threshold t, Chang's lemma provides a lower bound on $\delta(f) := \Pr[f(x) = -1]$ in terms of the dimension of the span of its characters with Fourier coefficients of magnitude at least $1/t$. In this work we examine the tightness of Chang's lemma with respect to the following three natural settings of the threshold:

- the Fourier sparsity of f, denoted $k(f)$,
- the Fourier max-supp-entropy of f, denoted $k'(f)$, defined to be the maximum value of the reciprocal of the absolute value of a non-zero Fourier coefficient,
- the Fourier max-rank-entropy of f, denoted $k''(f)$, defined to be the minimum t such that characters whose coefficients are at least $1/t$ in magnitude span a $r(f)$-dimensional space.

In this work we prove new lower bounds on $\delta(f)$ in terms of the above measures. One of our lower bounds, $\delta(f) = \Omega\left(r(f)^2/(k(f)\log^2 k(f))\right)$, subsumes and refines the previously best known upper bound $r(f) = O(\sqrt{k(f)}\log k(f))$ on $r(f)$ in terms of $k(f)$ by Sanyal (Theory of Computing, 2019). We improve upon this bound and show $r(f) = O(\sqrt{k(f)\delta(f)}\log k(f))$. Another lower bound, $\delta(f) = \Omega\left(r(f)/(k''(f)\log k(f))\right)$, is based on our improvement of a bound by Chattopadhyay, Hatami, Lovett and Tal (ITCS, 2019) on the sum of absolute values of level-1 Fourier coefficients in terms of $\mathbb{F}_2$-degree. We further show that Chang's lemma for the above-mentioned choices of the threshold is asymptotically outperformed by our bounds for most settings of the parameters involved.

Next, we show that our bounds are tight for a wide range of the parameters involved, by constructing functions witnessing their tightness. All the functions we construct are modifications of the Addressing function, where we replace certain input variables by suitable functions. Our final contribution is to construct Boolean functions f for which our lower bounds asymptotically match $\delta(f)$, and for any choice of the threshold t, the lower bound obtained from Chang's lemma is asymptotically smaller than $\delta(f)$.

Our results imply more refined deterministic one-way communication complexity upper bounds for XOR functions. Given the wide-ranging application of Chang's lemma to areas like additive combinatorics, learning theory and communication complexity, we strongly feel that our refinements of Chang's lemma will find many more applications.

[1] Work mostly done while the author was a postdoc at Georgetown University.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 10; pp. 10:1–10:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Theory of computation → Oracles and decision trees; Theory of computation → Communication complexity

Keywords and phrases Analysis of Boolean functions, Chang's lemma, Parity decision trees, Fourier dimension

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.10

Related Version *Full Version*: `https://arxiv.org/abs/2012.02335` [6]

Funding R.M. thanks DST (India) for grant DST/INSPIRE/04/2014/001799. S.S. is supported by an ISIRD Grant from SRIC, IIT Kharagpur. N.S.M. is supported by the NWO through QuantERA project QuantAlgo 680-91-034.

Acknowledgements T.M. would like to thank Prahladh Harsha and Ramprasad Saptharishi for helpful discussions.

1 Introduction

Chang's lemma [8, 14] is a classical result in additive combinatorics. Informally, the lemma states that all the large Fourier coefficients of the indicator function of a large subset of an Abelian group reside in a low dimensional subspace. The discovery of this lemma was motivated by an application to improve Frieman's theorem on set additions [8]. The lemma has subsequently found many applications in additive combinatorics and combinatorial number theory. Chang's lemma and the ideas developed in Chang's paper [8] have been used to prove theorems about arithmetic progressions in sumsets [12, 23], structure of Boolean functions with small spectral norm [16], and improved bounds for Roth's theorem on three-term arithmetic progressions in the integers [24, 4, 5]. Green and Ruzsa [15] used the ideas of Chang's lemma to prove a generalization of Frieman's theorem for arbitrary Abelian groups. The lemma is known to be sharp for various settings of parameters for the group $\mathbb{Z}_N$ [13].

In this paper, our focus is a specialization of Chang's lemma for the Boolean hypercube. Let $f : \{-1,1\}^n \to \{-1,1\}$ be a Boolean function. For any positive real number t (which we refer to as the *threshold*) define $\mathcal{S}_t := \{S \subseteq [n] : |\widehat{f}(S)| \geq \frac{1}{t}\}$.[2,3] Viewing elements of $\mathcal{S}_t$ as vectors in $\mathbb{F}_2^n$, Chang's lemma gives a lower bound on $\delta(f) := \Pr[f(x) = -1]$ (called the weight of f), in terms of t and the dimension of the span of $\mathcal{S}_t$ (denoted by $\dim(\mathcal{S}_t)$). Formally, we have the following lemma, referred to as Chang's lemma in this paper. In the literature Chang's lemma is more commonly stated as an upper bound on d in terms of $\delta(f)$ and t. We refer the reader to the full version of our paper for a proof of the equivalence of the two forms, and also for other missing proofs from this paper.

▶ **Lemma 1.1** (Chang's lemma [8]). *There exists a universal constant $c > 0$ such that the following is true for every integer $n > 0$. Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function and t be any positive real number. Let $\delta(f) := \Pr_x[f(x) = -1]$ and $d = \dim(\mathcal{S}_t) > 1$. If $\delta(f) < c$, then $\delta(f) = \Omega\left(\frac{\sqrt{d}}{t\sqrt{\log(t^2/d)}}\right)$.*

This lemma has found numerous applications in complexity theory and algorithms [2, 7], analysis of Boolean functions [16, 26], communication complexity [26, 19] and extremal combinatorics [10]. See [20] for a proof of Lemma 1.1.

[2] The function f is implicit in the definition of $\mathcal{S}_t$ and will be clear from context.
[3] We refer the reader to Section A for preliminaries on Fourier analysis.

In this paper, we investigate the tightness of Lemma 1.1 for three natural choices of the threshold t based on the Fourier spectrum of the function (see Section 1.1 for details about these thresholds). We prove additional lower bounds on $\delta(f)$, and compare relative performances of all the bounds under consideration. Our results imply that the bounds given by Chang's lemma for the choices of the threshold that we consider are asymptotically outperformed by one of the bounds we prove for a broad range of the parameters involved. For most regimes of the parameters we are able to construct classes of functions that witness the tightness of our bounds.

Interestingly, for each choice of threshold that we consider, $\dim(\mathcal{S}_t)$ equals the Fourier rank of f (denoted by $r(f)$, see Definition A.12). In particular, setting t to be the Fourier sparsity of f (denoted by $k(f)$) leads to a very natural question about the relationship among $r(f), k(f)$ and $\delta(f)$ for a Boolean function f. The best known upper bound on $r(f)$ in terms of $k(f)$ is $r(f) = O(\sqrt{k(f)} \log k(f))$ [25]. We improve upon this bound by incorporating $\delta(f)$ into it, and show $r(f) = O(\sqrt{k(f)\delta(f)} \log k(f))$. Moreover, we also show that this bound is tight; see Section 1.2 for a detailed discussion.

Throughout this paper, we assume that f is not a constant function or a parity or a negative parity (unless mentioned otherwise). In other words, $k(f), r(f) \geq 2$.

1.1 Thresholds considered for Chang's lemma

For a Boolean function f, let $\mathrm{supp}(f)$ denote the Fourier support of f. In this section, we discuss and motivate the choices of the threshold t considered in this work.

a) The Fourier sparsity of f. It was shown in [11, Theorem 3.3] that for all $S \in \mathrm{supp}(f)$, $|\widehat{f}(S)| \geq \frac{1}{k(f)}$. It follows that $\mathcal{S}_{k(f)} = \mathrm{supp}(f)$ and hence $\dim(\mathcal{S}_{k(f)}) = r(f)$. Moreover, there exist functions (e.g. $f = \mathsf{AND}_n$) for which $\dim(\mathcal{S}_t) = 0$ for $t = o(k(f))$, justifying the choice of threshold $k(f)$.

This choice also leads us to a fundamental structural problem of bounding the weight of a Boolean function f from below, in terms of its Fourier sparsity and Fourier rank. The *uncertainty principle* (see, for example, [17] for a statement and a proof) asserts that $\delta(f) = \Omega\left(\frac{1}{k(f)}\right)$. Chang's lemma with $t = k(f)$ and the fact that $\log\left(k(f)^2/r(f)\right) = \Theta(\log k(f))$ (Lemma A.16 (part 1)) implies that

$$\delta(f) = \Omega\left(\frac{1}{k(f)}\sqrt{\frac{r(f)}{\log k(f)}}\right), \tag{1}$$

thereby subsuming the uncertainty principle (note that $r(f)/\log k(f) \geq 1$) and refining it by incorporating $r(f)$ into the bound.

b) The Fourier max-supp-entropy of f. The next choice of the threshold that we consider is the *Fourier max-supp-entropy* of f, denoted by $k'(f)$, which we define to be $\max_{S \in \mathrm{supp}(f)} \frac{1}{|\widehat{f}(S)|}$ (Definition A.15). By its definition $k'(f)$ is the smallest value of t such that $\mathcal{S}_t = \mathrm{supp}(f)$. Since $k'(f) \leq k(f)$ (see the discussion in the last item), the knowledge of $k'(f)$ can potentially offer us a more fine-grained lower bound on $\delta(f)$ than as in the last item; Chang's lemma with $t = k'(f)$ and $\log\left(k'(f)^2/r(f)\right) = \Theta(\log k'(f))$ (Lemma A.16 (part 2)) implies

$$\delta(f) = \Omega\left(\frac{1}{k'(f)}\sqrt{\frac{r(f)}{\log k'(f)}}\right). \tag{2}$$

Notice that Equation (2) subsumes the bound in Equation (1).

In [18] an equivalent statement of the well-known sensitivity conjecture was presented in terms of $k'(f)$.[4] Granularity is another widely-studied measure that is closely associated with Fourier max-supp-entropy.

c) The Fourier max-rank-entropy of f. Our final choice of the threshold is the *Fourier max-rank-entropy of f*, denoted by $k''(f)$, which we define to be the smallest positive real number t such that $\dim(\mathcal{S}_t) = r(f)$ (Definition A.15). We have that $k''(f) \le k'(f) \le k(f)$ by their definitions. Amongst all settings of the threshold t for which $\dim(\mathcal{S}_t) = r(f)$, the value $t = k''(f)$ yields the best lower bound from Chang's lemma. Chang's lemma with $t = k''(f)$ implies

$$\delta(f) = \Omega\left(\frac{1}{k''(f)}\sqrt{\frac{r(f)}{\log\left(k''(f)^2/r(f)\right)}}\right), \tag{3}$$

which subsumes the bounds in Equations (2) and (1).

1.2 Our contributions

We prove the following results regarding the three natural instantiations of the threshold t (mentioned in the preceding section) for Chang's lemma.

a) The Fourier sparsity of f. Recall that Chang's lemma with threshold $t = k(f)$ (Equation (1)) implies that $\delta(f) = \Omega\left(\frac{1}{k(f)}\sqrt{\frac{r(f)}{\log k(f)}}\right)$. It was shown in [1] that $\delta(f) = \Omega\left(\frac{1}{k(f)}\left(\frac{r(f)}{\log k(f)}\right)\right)$, improving upon this bound asymptotically (note that $r(f)/\log k(f) \ge 1$). In this work we improve their bound further.

▶ **Theorem 1.2.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function such that $k(f) > 1$. Then $\delta(f) = \Omega\left(\frac{1}{k(f)}\left(\frac{r(f)}{\log k(f)}\right)^2\right)$.*

Observe that the statement of Theorem 1.2 is equivalent to $r(f) = O(\sqrt{k(f)\delta(f)}\log k(f))$. This bound subsumes the bound $r(f) = O(\sqrt{k(f)}\log k(f))$ shown by Sanyal [25]. We prove Theorem 1.2 by incorporating $\delta(f)$ in Sanyal's arguments and thereby refining his proof. See Section 2.1 for the proof of Theorem 1.2.

We also show that Theorem 1.2 is tight. For nearly all admissible values of ρ and κ we construct many Boolean functions f such that $k(f) = O(\kappa)$, $r(f) = O(\rho)$ and $\delta(f) = O\left(\frac{1}{\kappa}\left(\frac{\rho}{\log\kappa}\right)^2\right)$ (Theorem 1.4 and Claim B.2). For a comparison with Sanyal's bound see Section 1.3.

b) The Fourier max-supp-entropy of f. Recall from Section 1.1 that the Fourier max-supp-entropy of f, denoted $k'(f)$, is defined as $k'(f) = \max_{S \in \mathrm{supp}(f)} \frac{1}{|\widehat{f}(S)|}$. It can be shown that $\sqrt{k(f)} \le k'(f) \le k(f)/2$ (Lemma A.16 (part 2)). We prove the following lower bound.

▶ **Theorem 1.3.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function such that $k(f) > 1$. Then, $\delta(f) = \Omega\left(\max\left\{\frac{1}{k(f)}\left(\frac{r(f)}{\log k(f)}\right)^2, \frac{k(f)}{k'(f)^2}\right\}\right)$.*

[4] In [18] $\log(k'(f)^2)$ is called the Fourier max-entropy while we refer to $k'(f)$ as the Fourier max-supp-entropy.

As is evident from the statement, Theorem 1.3 presents two lower bounds, one of which is Theorem 1.2. The other lower bound $\delta(f) \geq \frac{k(f)}{k'(f)^2}$ is Claim A.17.

Chang's lemma with the threshold t set to $k'(f)$ (Equation (2)), together with the observation that $\log k(f) = \Theta(\log k'(f))$, implies $\delta(f) = \Omega\left(\frac{1}{k'(f)}\sqrt{\frac{r(f)}{\log k(f)}}\right)$. Theorem 1.3 subsumes this bound since

$$\delta(f) = \Omega\left(\frac{1}{k(f)}\left(\frac{r(f)}{\log k(f)}\right)^2 \cdot \frac{k(f)}{k'(f)^2}\right)^{1/2} = \Omega\left(\frac{1}{k'(f)}\sqrt{\frac{r(f)}{\log k(f)}}\right),$$

where the equality follows from $r(f)/\log k(f) \geq 1$.

In addition, observe from the last equality above that the bound of Theorem 1.3 is asymptotically larger than the bound obtained from Chang's lemma for $t = k'(f)$ (Equation (2)) except when $r(f)/\log k(f) = \Theta(1)$. Theorem 1.4 complements Theorem 1.3 by showing that for nearly all admissible values of $r(f), k(f)$ and $k'(f)$, there exists a function for which the larger of the two bounds presented in Theorem 1.3 is tight.

▶ **Theorem 1.4.** *For all $\rho, \kappa, \kappa' \in \mathbb{N}$ such that κ is sufficiently large, for all constants $\epsilon > 0$ such that $\log \kappa \leq \rho \leq \kappa^{\frac{1}{2}-\epsilon}$ and $\kappa^{\frac{1}{2}} \leq \kappa' \leq \kappa$, there exists a Boolean function $f_{\rho,\kappa,\kappa'}$ such that $r(f_{\rho,\kappa,\kappa'}) = \Theta(\rho)$, $k(f_{\rho,\kappa,\kappa'}) = \Theta(\kappa)$, $k'(f_{\rho,\kappa,\kappa'}) = \Theta(\kappa')$ and*

$$\delta(f_{\rho,\kappa,\kappa'}) = \Theta\left(\max\left\{\frac{1}{\kappa}\left(\frac{\rho}{\log \kappa}\right)^2, \frac{\kappa}{\kappa'^2}\right\}\right).$$

The range of parameters considered in Theorem 1.4 is justified by Lemma A.16. We prove Theorem 1.4 in two parts. Fix any ρ, κ such that $\log \kappa \leq \rho \leq \kappa^{\frac{1}{2}-\epsilon}$ for some constant $\epsilon > 0$. First, for each value of $\kappa' \in [\frac{\kappa \log \kappa}{\rho}, \kappa]$ we construct a function f for which the first lower bound on $\delta(f)$ from Theorem 1.3 is tight (Claim B.2). Next, for each value of $\kappa' \in [\kappa^{\frac{1}{2}}, \frac{\kappa \log \kappa}{\rho}]$ we construct a function f for which the second lower bound on $\delta(f)$ from Theorem 1.3 is tight (Claim B.3). See Figure 1 for a graphical visualization of the bounds in Theorem 1.3 for any fixed values of ρ and κ.

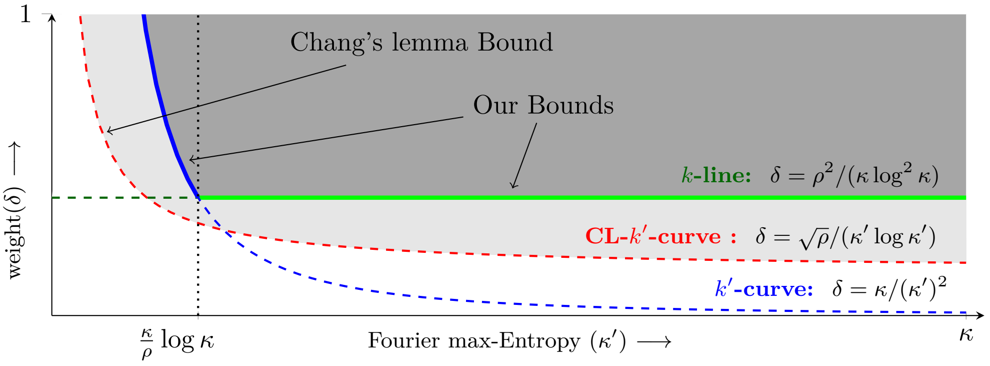


Figure 1 This plot is constructed for any fixed values of ρ, κ for which $\log \kappa \leq \rho \leq \sqrt{\kappa}$, and depicts the relationship between $\delta(f)$ and $k'(f)$ for functions f with $r(f) = \Theta(\rho)$ and $k(f) = \Theta(\kappa)$. For any fixed values of ρ, κ, we will refer to this plot as the (ρ, κ)-k'-plot. Chang's lemma implies that Boolean functions lie above the CL-k'-curve. Theorem 1.3 improves upon Chang's lemma and shows that Boolean functions lie above both the k-line and the k'-curve, highlighted by the dark grey region in the figure. Roughly speaking, Theorem 1.4 exhibits functions that lie on the boundary of the dark grey region described by the k-line and the k'-curve.

c) The Fourier max-rank-entropy of f. Recall from Section 1.1 that the Fourier max-rank-entropy of f, denoted $k''(f)$, is the smallest positive real number t such that $\dim(\mathcal{S}_t) = r(f)$. It can be shown that $\max\left\{\sqrt{r(f)}, \frac{r(f)}{\log k(f)}\right\} \leq k''(f) \leq k(f)$ (Lemma A.16 (part 2)). We prove the following lower bound.

▶ **Theorem 1.5.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function such that $k(f) > 1$. Then,*

$$\delta(f) = \Omega\left(\max\left\{\frac{1}{k(f)}\left(\frac{r(f)}{\log k(f)}\right)^2, \frac{r(f)}{k''(f)\log k(f)}\right\}\right).$$

Theorem 1.5 yields a better lower bound than Chang's lemma with the threshold $t = k''(f)$ (Equation (3)), except when $r(f) < (\log k(f))^2$ (see the caption of Figure 2). Theorem 1.5 presents two lower bounds: the first one is Theorem 1.2, and the second one is Lemma 2.5. Lemma 2.5 is proven by strengthening a bound due to [9] on the sum of absolute values of level-1 Fourier coefficients of a Boolean function in terms of its $\mathbb{F}_2$-degree. A proof of Theorem 1.5 can be found in Section 2.2.

We also show that for nearly all admissible values of $r(f), k(f)$ and $k''(f)$, there exist functions for which the larger of the two bounds presented in Theorem 1.5 is nearly tight.

▶ **Theorem 1.6.** *For all $\rho, \kappa, \kappa'' \in \mathbb{N}$ such that κ is sufficiently large, for all $\epsilon > 0$ such that $\log\kappa \leq \rho \leq \kappa^{\frac{1}{2}-\epsilon}$ and $\rho \leq \kappa'' \leq \kappa$ there exists a Boolean function $f_{\rho,\kappa,\kappa''}$ such that $r(f_{\rho,\kappa,\kappa''}) = \Theta(\rho)$, $k(f_{\rho,\kappa,\kappa''}) = \Theta(\kappa)$, $k''(f_{\rho,\kappa,\kappa''}) = \Theta(\kappa'')$ and*

$$\delta(f_{\rho,\kappa,\kappa''}) = \Theta\left(\max\left\{\frac{1}{\kappa}\left(\frac{\rho}{\log\kappa}\right)^2, \frac{\rho}{\kappa''\log(\kappa''/\rho)}\right\}\right).$$

The range of parameters considered in Theorem 1.6 is justified by Lemma A.16. Theorem 1.6 is proved in two parts. Fix any ρ, κ such that $\log\kappa \leq \rho \leq \kappa^{\frac{1}{2}-\epsilon}$ for some constant $\epsilon > 0$. First, for each value of $\kappa'' \in [\frac{\kappa\log\kappa}{\rho}, \kappa]$ we construct a function f for which the first lower bound on $\delta(f)$ from Theorem 1.5 is tight (Claim B.5). In fact these are the same functions that are used to prove the first bound in Theorem 1.4. Next, for each value of $\kappa'' \in [e\rho, \frac{\kappa\log\kappa}{\rho}]$ we construct a function f for which $\delta(f) = \Theta(\frac{\rho}{\kappa''\log(\kappa''/\rho)})$ (Claim B.4). From the above discussion one may verify that for every ρ, κ that we consider and for every $\kappa'' \geq \rho \cdot \kappa^{\Omega(1)}$, the function that we construct witnesses tightness of the lower bound in Theorem 1.5.

In general, for all settings of ρ, κ and κ'' that we consider, the upper bound on $\delta(f)$ from Theorem 1.6 is off by a factor of at most $O(\log\kappa)$ from the lower bound in Theorem 1.5. See Figure 2 for a graphical visualization of the bounds in Theorem 1.5 for any fixed values of ρ and κ.

Dominating Chang's lemma for all thresholds. Our final contribution is to show that there exists a function for which: our lower bounds (Theorem 1.3 and 1.5) asymptotically match the weight, but for any choice of the threshold the lower bound obtained from Chang's lemma (Lemma 1.1) is asymptotically smaller than the weight. See [6, Section 7] in the full version of our paper for a proof of the below claim.

▷ Claim 1.7 (Beating Chang's lemma for all thresholds). For any integer $t > 4$ there exists a function $f : \{-1,1\}^{\log t} \times \{-1,1\}^{t\log t} \to \{-1,1\}$ such that

- $\delta(f) = \frac{1}{t}$.
- For all real $x > 0$ for which $\dim(\mathcal{S}_x) > 1$, we have $\frac{\sqrt{\dim(\mathcal{S}_x)}}{x\sqrt{\log(x^2/\dim(\mathcal{S}_x))}} = O\left(\frac{1}{t^{3/2}}\right)$.
-

$$\frac{1}{k(f)}\left(\frac{r(f)}{\log k(f)}\right)^2 = \Omega\left(\frac{1}{t}\right), \quad \frac{k(f)}{k'(f)^2} = \Omega\left(\frac{1}{t}\right) \quad \text{and} \quad \frac{r(f)}{k''(f)\log k(f)} = \Omega\left(\frac{1}{t}\right).$$

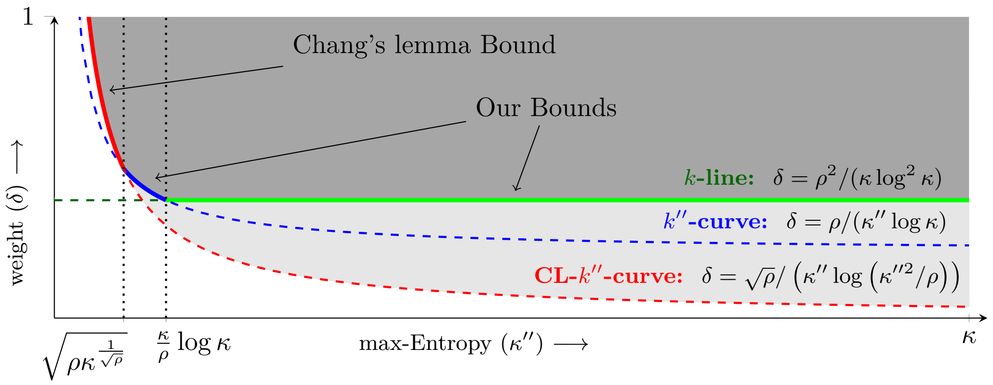


Figure 2 This plot is constructed for any fixed values of ρ, κ for which $\log\kappa \leq \rho \leq \sqrt{\kappa}$, and depicts the relationship between $\delta(f)$ and $k''(f)$ for functions f with $r(f) = \Theta(\rho)$ and $k(f) = \Theta(\kappa)$. For any fixed values of ρ, κ, we will refer to this plot as (ρ, κ)-k''-plot. Chang's lemma implies that Boolean functions lie above the CL-k''-curve. Theorem 1.5 improves upon Chang's lemma and shows that Boolean functions lie above both the k-line and the k''-curve, highlighted by the dark grey region in the figure. Although the picture indicates that the CL-k''-curve is better than the k''-curve for certain ranges of κ'', this is actually only possible for certain values of ρ and κ. This is because the CL-k''-curve and the k''-curve intersect at $\sqrt{\rho\kappa^{1/\sqrt{\rho}}}$, which is less than $\sqrt{\rho}$ if $\rho \geq (\log\kappa)^2$. By Lemma A.16 we know that for any function f on this plot, the range of $k''(f)$ is between $\max\{\sqrt{\rho}, \rho/\log\kappa\}$ and κ. Thus our bounds in Theorem 1.5 dominate those given by the CL-k''-curve in all (ρ, κ)-k'' plots where $\rho \geq \log^2\kappa$.

In particular, Claim 1.7 shows that our bounds can be strictly stronger than those given by Chang's lemma, in the following sense.

- All the lower bounds on $\delta(f)$ from Theorems 1.3 and 1.5 are tight, as witnessed by f from Claim 1.7.
- For the function f from Claim 1.7, no matter what threshold x is chosen in Lemma 1.1, the best possible lower bound on $\delta(f)$ that we get can get from Lemma 1.1 is $\Omega\left(\frac{1}{t^{3/2}}\right)$. This is polynomially smaller than $1/t$, the actual weight of f.

1.3 Applications of our results

An application of our result is an enhanced understanding of the bound $r(f) = O(\sqrt{k(f)}\log k(f))$ proven by Sanyal [25]. This bound is a special case of Theorem 1.2 for $\delta(f) = \Theta(1)$. It is not known whether the $\log k(f)$ term is required in Sanyal's upper bound on $r(f)$ (when f equals the Addressing function, $r(f) = \Omega(\sqrt{k(f)})$, see Definition A.10 and Observation A.19). For all the functions we construct witnessing the tightness of the bound in Theorem 1.2, $\delta(f) = o(1)$. We prove Theorem 1.2 by generalizing Sanyal's proof. As stated before, our bound is tight in this generality, i.e. the logarithmic factor is required in the upper bound on $r(f)$. This sheds light on the presence of the logarithmic term in the bound $r(f) = O(\sqrt{k(f)}\log k(f))$.

Also, Fourier sparsity and Fourier rank of f have intimate connections with the communication complexity of functions of the form $F := f \circ \mathsf{XOR}$. The Fourier sparsity of f equals the real rank ($\mathsf{rank}(M_F)$) of the communication matrix M_F of F, and the Fourier rank of f equals the deterministic (and even exact quantum) one-way communication complexity of F [22]. Theorem 1.2 thus implies an improved upper bound of $O(\sqrt{k(f)\delta(f)}\log k(f))$ on the one-way communication complexity of F in these models, which asymptotically beats the best known upper bound of $O(\sqrt{\mathsf{rank}(M_F)})$ even for two-way protocols [26, 21], for the special case of functions of this form (when $\delta(f) = o(1/\log k)$).

Given the wide-ranging application of Chang's lemma to areas like additive combinatorics, learning theory and communication complexity, we strongly feel that our refinements of Chang's lemma will find many more applications.

2 Lower bound proofs

For lower bounds on $\delta(f)$ of a Boolean function f, we need to prove two theorems: Theorems 1.3 and 1.5. The proof of Theorem 1.3 is given in Section 2.1 and the proof of Theorem 1.5 is given in Section 2.2.

2.1 Proof of Theorem 1.3 (and Theorem 1.2)

Remember that we defined the *Fourier max-supp-entropy* of a Boolean function f, denoted by $k'(f)$, to be $\max_{S \in \text{supp}(f)} \frac{1}{|\widehat{f}(S)|}$.

The main aim of this section is to give a lower bound on $\delta(f)$ with respect to $k'(f)$ for a Boolean function f (Theorem 1.3).

We first prove Theorem 1.2 which implies Theorem 1.3 (together with Claim A.17).

Theorem 1.2 can be viewed as an upper bound of $O(\sqrt{k(f)\delta(f)} \log k(f))$ on the Fourier rank of f. In order to prove Theorem 1.2, we give an algorithm (Algorithm 1) which takes a Boolean function f as input and outputs a set of $O(\sqrt{\delta(f)k(f)} \log k(f))$ parities such that any assignment of these parities makes the function constant. From Observation A.14, this implies an upper bound of $O(\sqrt{\delta(f)k(f)} \log k(f))$ on Fourier rank of the function. We start by formally describing this algorithm. The central ingredient in the algorithm is a lemma in [26, Lemma 28].

▶ **Lemma 2.1** ([26]). *Let $f : \{-1,1\}^n \to \{-1,1\}$ a function. There is an affine subspace $V \subseteq \{-1,1\}^n$ of co-dimension at most $3\sqrt{\delta(f)k(f)}$ such that f is constant on V.*

Recall that for a function $f : \{-1,1\}^n \to \{-1,1\}$, a set of parities Γ and an assignment $b \in \{-1,1\}^\Gamma$, we define the restriction $f|_{(\Gamma,b)} := f|_{\{x \in \{-1,1\}^n : \chi_\gamma(x) = b_\gamma \text{ for all } \gamma \in \Gamma\}}$. Also let $\mathcal{B}_\Gamma := \{b \in \{-1,1\}^\Gamma : f|_{(\Gamma,b)} \textit{ is not constant}\}$.

Algorithm 1

```
Input: A function $f : \{-1,1\}^n \to \{-1,1\}$.
Output: A set $\Gamma$ of parities whose evaluation determines $f$.
Initialization: $f_{\min} \leftarrow f$, $\Gamma \leftarrow \emptyset$.
while $\mathcal{B}_\Gamma$ is non-empty do
    (a) Update $\Gamma$: Let $\Gamma'$ be the smallest set of parities, such that, there exists
        $b \in \{-1,1\}^{\Gamma'}$ for which $f_{\min}|_{(\Gamma',b)}$ is constant,
            $\Gamma \leftarrow \Gamma \cup \Gamma'$.
    (b) Update $f_{\min}$: Define $b^* := \text{argmin}_{b \in \mathcal{B}_\Gamma} \left\{ \frac{\delta(f|_{(\Gamma,b)})}{k(f|_{(\Gamma,b)})} \right\}$, and update
            $f_{\min} \leftarrow f|_{(\Gamma,b^*)}$.
end
Return $\Gamma$.
```

Since number of parities are finite and we fix at least one parity at each iteration of Step a of the **while** loop, the algorithm terminates. The termination condition implies that the algorithm outputs a set of parities Γ such that for any assignment $b \in \{-1,1\}^{\Gamma}$ of Γ, the restricted function $f_{(\Gamma,b)}$ becomes constant.

The only remaining step is to show is that the number of parities fixed in Algorithm 1 is $O(\sqrt{\delta(f)k(f)}\log k(f))$. For this we define an equivalence relation and observe a few properties of restricted functions (restricted according to an assignmen t of a set of parities).

Equivalence relation for a set of parities

Let f be the input to Algorithm 1, first we define an equivalence relation given a set of parities over the variables of f. Given a set of parities Γ, define the following equivalence relation among parities in $\mathrm{supp}(f)$.

$$\forall \gamma_1, \gamma_2 \in \mathrm{supp}(f), \gamma_1 \equiv \gamma_2 \text{ iff } \gamma_1 + \gamma_2 \in \mathrm{span}(\Gamma). \tag{4}$$

Let ℓ be the number of equivalence classes according to the equivalence relation for Γ. For $j \in [\ell]$, let k_j be the size of the j-th equivalence class. Since the equivalence classes form a partition of $\mathrm{supp}(f)$, we have

▶ **Observation 2.2.** *Following the notation of the paragraph above,* $\sum_{j=1}^{\ell} k_j = k(f)$.

Let $\beta_1, \ldots, \beta_\ell \in \mathrm{supp}(f)$ be some representatives of the equivalence classes. For $j \in [\ell]$, let $\beta_j + \alpha_{j,1}, \ldots, \beta_j + \alpha_{j,k_j}$ be the elements of the j-th equivalence class. This notation gives a compact representation of f in terms of these equivalence classes. For all $x \in \{-1,1\}^n$,

$$f(x) = \sum_{j=1}^{\ell} P_j(x)\chi_{\beta_j}(x), \tag{5}$$

where

$$P_j(x) = \sum_{r=1}^{k_j} \widehat{f}(\beta_j + \alpha_{j,r}) \cdot \chi_{\alpha_{j,r}}(x). \tag{6}$$

Note that P_j are non-zero multilinear polynomials and depend only on the parities in Γ. So, fixing parities in Γ collapses all the parities in an equivalence class to their representative, thereby making P_j's constant.

We will denote Γ after the i-th iteration of the **while** loop by $\Gamma^{(i)}$ (so $\Gamma^{(0)} = \emptyset$). Let $f_{\min}^{(i)}$ be the selected function $f_{\min}$ after the i-th iteration (thus $f_{\min}^{(0)} = f$).

With the above properties of restricted functions we are ready to prove the main technical lemma needed to show Theorem 1.2.

▶ **Lemma 2.3.** *Let* $f : \{-1,1\}^n \to \{-1,1\}$ *a function. Suppose* Γ *be a set of parities and* ℓ *be the number of equivalence classes of* $\mathrm{supp}(f)$ *under the equivalence relation defined by in Equation* (4)*, Then, there exists a* $b \in \{-1,1\}^{\Gamma}$ *such that* $f|_{(\Gamma,b)}$ *is non-constant and* $\frac{\delta(f|_{(\Gamma,b)})}{k(f|_{(\Gamma,b)})} \leq \frac{4k(f)\delta(f)}{\ell^2}$.

Proof. For the sake of succinctness, when Γ is clear from the context, let $V_b = \{x \in \{-1,1\}^n : \forall \gamma \in \Gamma, x_\gamma = b_\gamma\}$, for all $b \in \{-1,1\}^{\Gamma}$, and $f|_b = f|_{\{x : x \in V_b\}}$.

Since we are interested in a non-constant $f|_b$, define $k_{\{\emptyset\}^c}(f)$ to be the number of non-zero non-empty monomials in Fourier representation of f. We first need to prove the following two bounds on the expected values of $\delta(f|_b)$ and $k_{\{\emptyset\}^c}(f|_b)$.

- $\mathbb{E}_b\left[\delta(f|_b)\right] = \delta(f)$,
- $\mathbb{E}_b\left[k_{\{\emptyset\}^c}(f|_b)\right] \geq \frac{\ell^2}{4k(f)}$.

Expected value of $\delta(f|_b)$. Since $\left\{V_b : b \in \{-1,1\}^{\Gamma^{(i)}}\right\}$ form a partition on $\{-1,1\}^n$ and all partitions are of the same size, we get the expected value of $\delta(f|_b)$.

$$\mathbb{E}_b\left[\delta(f|_b)\right] = \delta(f). \tag{7}$$

Expected value of $k_{\{\emptyset\}^c}(f|_b)$. From Equation (5), for all $b \in \{-1,1\}^{\Gamma}$ and for all $x \in \{-1,1\}^n$,

$$f|_b(x) = \sum_{j=1}^{\ell} P_j(b)\chi_{\beta_j}(x). \tag{8}$$

For each $j \in [\ell]$ and $b \in \{-1,1\}^{\Gamma}$, let $I_j(b)$ be the indicator function for $P_j(b) \neq 0$,

$$I_j(b) = \begin{cases} 1 & \text{if } P_j(b) \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

From Equation (6), each P_j is a polynomial having monomials $\{\chi_{\alpha_{j,r}} : r \in [k_j]\}$ with Fourier sparsity of P_j being equal to k_j. Since each P_j is a non-zero polynomial, by Lemma A.2

$$\mathbb{E}_b\left[I_j(b)\right] = \Pr_{b\sim\{-1,1\}^{\Gamma}}\left[P_j(b) \neq 0\right] \geq \frac{1}{k_j}. \tag{9}$$

We calculate the expectation of $k_{\{\emptyset\}^c}(f|_b)$.

$$\begin{aligned}
\mathbb{E}_b\left[k_{\{\emptyset\}^c}(f|_b)\right] &= \mathbb{E}_b\left[\sum_{j=1}^{\ell-1} I_j(b)\right] && \text{by Equation (8)}\\
&= \sum_{j=1}^{\ell-1} \mathbb{E}_b\left[I_j(b)\right] && \text{by linearity of expectation}\\
&\geq \sum_{j=1}^{\ell-1} \frac{1}{k_j} && \text{by Equation (9)}\\
&\geq \frac{(\ell-1)^2}{\sum_{j=1}^{\ell-1} k_j} && \text{by Cauchy-Schwarz inequality}\\
&\geq \frac{\ell^2}{4k(f)}. && \text{by Observation 2.2}
\end{aligned}$$

To finish the proof of the theorem, we use bounds on the two expected values,[5]

$$\begin{aligned}
&\frac{\mathbb{E}_b\left[\delta(f|_b)\right]}{\mathbb{E}_b\left[k_{\{\emptyset\}^c}(f|_b)\right]} \leq \frac{4k(f)\delta(f)}{\ell^2}\\
\iff &\mathbb{E}_b\left[\delta(f|_{V_b}) - \frac{4k(f)\delta(f)}{\ell^2} k_{\{\emptyset\}^c}(f|_{V_b})\right] \leq 0. && \text{by linearity of expectation}
\end{aligned}$$

[5] this part of our proof is inspired by a proof of the Cheeger's inequality in spectral graph theory. See, for example, the proof of Fact 2 in `https://people.eecs.berkeley.edu/~luca/expanders2016/lecture04.pdf`.

If $\delta(f|_{V_b}) - \frac{4k(f)\delta(f)}{\ell^2} k_{\{\emptyset\}^c}(f|_{V_b}) = 0$ for all b, then pick any non-constant $f|_b$. Otherwise, there exists a b_0 such that $\delta(f|_{V_{b_0}}) - \frac{4k(f)\delta(f)}{\ell^2} k_{\{\emptyset\}^c}(f|_{V_{b_0}}) < 0$. Since this equation can only be satisfied when $k_{\{\emptyset\}^c}(f|_{V_{b_0}}) > 0$, $f|_{V_{b_0}}$ is not constant. Dividing by $k_{\{\emptyset\}^c}(f|_{V_{b_0}})$,

$$\frac{\delta(f|_{b_0})}{k(f|_{b_0})} \leq \frac{\delta(f|_{b_0})}{k_{\{\emptyset\}^c}(f|_{b_0})} \leq \frac{4k(f)\delta(f)}{\ell^2},$$

and $f|_{b_0}$ is non-constant. ◀

Lemma 2.3 allows us to bound the number of parities fixed in the i-th iteration (in terms of the decrease in number of equivalence classes).

▶ **Lemma 2.4.** *Suppose f is given as input to Algorithm 1. Consider the i-th iteration of Algorithm 1. Let q_i be the be number of parities fixed in Step a of the i-th iteration of the* **while** *loop, and ℓ_i be the number of equivalence classes after Step a of the i-th iteration. Then*

$$\frac{q_i}{(\ell_{i-1} - \ell_i)} \leq \frac{6\sqrt{\delta(f)k(f)}}{\ell_{i-1}}.$$

Proof. Recall that $\Gamma = \Gamma^{(i)}$ after the i-th of Step a of Algorithm 1. Again, for the sake of succinctness, let $V_b = \{x \in \{-1,1\}^n : \forall\gamma \in \Gamma^{(i)}, x_\gamma = b_\gamma\}$, for all $b \in \{-1,1\}^{\Gamma^{(i)}}$, and $f|_b = f|_{\{x:x\in V_b\}}$. Let $f_{\min}$ be the function chosen after the i-th iteration of Step b of Algorithm 1. Since Step b of Algorithm 1 chooses $f_{\min}$ to be a non-constant function such that weight-to-sparsity ratio is minimized, from Lemma 2.3 we have,

$$\frac{\delta(f_{\min})}{k(f_{\min})} \leq \frac{4k(f)\delta(f)}{\ell_{i-1}^2}. \tag{10}$$

Write every $f|_b$ as in Equation (5), and define $\mathcal{S}^{(i)} := \bigcup_{b\in\{-1,1\}^{\Gamma^{(i)}}} \mathrm{supp}(f|_b)$. We now prove that $|\mathcal{S}^{(i)}| = \ell_i$.

- $|\mathcal{S}^{(i)}| \leq \ell_i$: Follows from the representation in Equation (5), since each $\mathrm{supp}(f|_b)$ is a subset of $\{\chi_{\beta_j^{(i)}} \mid j \in [\ell_i]\}$.
- $|\mathcal{S}^{(i)}| \geq \ell_i$: Since $P_j^{(i)}$ is a non-zero polynomial, there exists an assignment to parities in $\Gamma^{(i)}$, such that, $P_j^{(i)}$ is non-zero. Thus, for all $j \in [\ell_i]$, we have $\chi_{\beta_j^{(i)}} \in \mathcal{S}^{(i)}$.

Since $|\mathcal{S}^{(i)}| = \ell_i$, Lemma 2.1 guarantees that $q_i \leq 3\sqrt{k(f_{\min})\delta(f_{\min})}$. Since $f_{\min}$ becomes constant after fixing these q_i parities, every parity in $\mathrm{supp}(f_{\min})$ is paired with at least one other parity in $\mathrm{supp}(f_{\min})$ for the equivalence class with respect to $\Gamma^{(i)}$.[6] This implies that $\ell_{i-1} - \ell_i \geq \frac{k(f_{\min})}{2}$ Combining the two inequalities in the last paragraph we have,

$$\frac{q_i}{(\ell_{i-1} - \ell_i)} \leq 6\sqrt{\frac{\delta(f_{\min})}{k(f_{\min})}}.$$

From Equation (10),

$$\frac{q_i}{(\ell_{i-1} - \ell_i)} \leq \frac{6\sqrt{\delta(f)k(f)}}{\ell_{i-1}}. \tag{11}$$

◀

[6] There is a boundary case ($k(f) = 1$) which can be dealt with separately, as in [25, Lemma 3.4]. For readability, we assume $k(f) \geq 2$.

We are now ready to prove Theorem 1.2.

Proof of Theorem 1.2. We only need to show that the number of parities fixed in Algorithm 1 is $O(\sqrt{\delta(f)k(f)}\log k(f))$ (Observation A.14). Suppose the while loop runs for t iterations. Let q_i be the number of queries made in Step a of Algorithm 1 in the i-th iteration. From Lemma 2.3, we have

$$q_i \leq \frac{6\sqrt{\delta(f)k(f)}}{\ell_{i-1}}(\ell_{i-1}-\ell_i)$$

Thus when Algorithm 1 is run of f, the total number of queries made by the algorithm is

$$\begin{aligned}\sum_{i=1}^{t} q_i &\leq 6\sqrt{\delta(f)k(f)}\sum_{i=1}^{t}\frac{(\ell_{i-1}-\ell_i)}{\ell_{i-1}}\\ &\leq 6\sqrt{\delta(f)k(f)}\sum_{i=1}^{t}\left(\frac{1}{\ell_{i-1}}+\frac{1}{\ell_{i-1}-1}\dots+\frac{1}{\ell_i+1}\right)\\ &\leq 6\sqrt{\delta(f)k(f)}\sum_{i=1}^{\ell_0}\frac{1}{i}\\ &\leq 6\sqrt{\delta(f)k(f)}\log \ell_0 = 6\sqrt{\delta(f)k(f)}\log k(f).\end{aligned}$$

Observation A.14 implies $r(f) = O(\sqrt{\delta(f)k(f)}\log k(f))$. ◀

Along with Theorem 1.2, this proves Theorem 1.3.

Proof of Theorem 1.3. The bound $\delta(f) = \Omega\left(\frac{1}{k(f)}\left(\frac{r(f)}{\log k(f)}\right)^2\right)$ follows from Theorem 1.2 and the bound $\delta(f) = \Omega\left(\frac{k(f)}{(k'(f))^2}\right)$ from Claim A.17. ◀

2.2 Proof of Theorem 1.5

Recall that we defined *max-rank-entropy* of a Boolean function f, denoted by $k''(f)$, to be $\text{argmin}_t\{\dim(\mathcal{S}_t)\} = r(f)$. The main aim of this section is to give a lower bound on $\delta(f)$ with respect to $k''(f)$ for a Boolean function f (Theorem 1.5). The second bound of Theorem 1.5 is given by the following lemma.

▶ **Lemma 2.5.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function such that $k(f) > 1$. Then, $\delta(f) = \Omega\left(\frac{r(f)}{k''(f)\log k(f)}\right)$.*

Together with Theorem 1.2 proved in Section 2.1, Lemma 2.5 implies Theorem 1.5. We now give the proof of Lemma 2.5.

Lemma 2.5 gives a lower bound of $\Omega\left(\frac{r(f)}{k''(f)\log k(f)}\right)$ on $\delta(f)$. The crucial ingredient for this lower bound is Lemma 2.7, which is a refinement of the following theorem.

▶ **Theorem 2.6** ([9, Theorem 13]). *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function such that $\deg_{\mathbb{F}_2}(f) = d$. Then, $\sum_{i\in[n]}|\widehat{f}(\{i\})| \leq 4d$.*

▶ **Lemma 2.7.** *For any Boolean function f, $\sum_{i=1}^{n}|\widehat{f}(i)| = O(\delta(f)\deg_{\mathbb{F}_2}(f))$.*

The proof of Lemma 2.7 for a Boolean function f essentially applies Theorem 2.6 on the xor of disjoint copies of f. The only difference in the statement of Lemma 2.7 and Theorem 2.6 is that the right hand side becomes $O(\delta(f)\cdot\deg_{\mathbb{F}_2}(f))$ instead of $4\deg_{\mathbb{F}_2}(f)$.

Proof of Lemma 2.7. Assume $\delta(f) \leq 1/4$ (otherwise Theorem 2.6 implies $\sum_{i=1}^{n} |\widehat{f}(i)| = O(\delta(f)d)$). Define $F : \{-1,1\}^{nt} \to \{-1,1\}$ to be $F(x^{(1)}, \ldots, x^{(t)}) := f(x^{(1)}) \times \ldots \times f(x^{(t)})$, where t is a parameter to be fixed later, and $x^{(i)} \in \{-1,1\}^{n}$ for all $i \in [t]$. Since $\deg_{\mathbb{F}_2}(F) = \deg_{\mathbb{F}_2}(f)$, Theorem 2.6 implies

$$\sum_{\substack{S \subseteq [nt] \\ |S|=1}} |\widehat{F}(S)| = O(d). \tag{12}$$

Since $(1-x)^{1/x}$ is a decreasing function in x for $x \in (0, 1/2]$, we have

$$(1-x)^{1/x} \geq 1/4 \quad \text{for all } x \in (0, 1/2]. \tag{13}$$

Expressing the Fourier coefficients of F in terms of the Fourier coefficients of f,

$$\begin{aligned}
\sum_{\substack{S \subseteq [nt] \\ |S|=1}} |\widehat{F}(S)| &= t \cdot \widehat{f}(\emptyset)^{t-1} \sum_{i=1}^{n} |\widehat{f}(i)| \\
&= \left(1 + \frac{1}{2\delta(f)}\right) \cdot (1 - 2\delta(f))^{\frac{1}{2\delta(f)}} \sum_{i=1}^{n} |\widehat{f}(i)| \\
&\qquad\qquad \text{Choosing } t = 1 + \tfrac{1}{2\delta(f)}, \text{ and since } \widehat{f}(\emptyset) = 1 - 2\delta(f) \\
&\geq \left(1 + \frac{1}{2\delta(f)}\right) \cdot \left(\frac{1}{4}\right) \sum_{i=1}^{n} |\widehat{f}(i)| \qquad \text{by Equation (13)} \\
&\geq \frac{1}{8\delta(f)} \cdot \sum_{i=1}^{n} |\widehat{f}(i)|.
\end{aligned}$$

Now, Equation (12) implies the desired bound, $\sum_{i=1}^{n} |\widehat{f}(i)| = O(\delta(f)d)$. ◀

We would like to extend the upper bound of Lemma 2.7 to any basis of span(supp(f)) instead of just the standard basis of the set of parities.

▶ **Corollary 2.8.** *Let $f : \{-1,1\}^{n} \to \{-1,1\}$ be any function with $\deg_{\mathbb{F}_2}(f) = d$. Suppose $\mathcal{S} \subseteq \operatorname{supp}(f)$ is a basis of $\operatorname{span}(\operatorname{supp}(f))$, then*

$$\sum_{S \in \mathcal{S}} |\widehat{f}(S)| = O(\delta(f)d) = O(\delta(f) \log k(f)).$$

Proof. The main idea of the proof is to do a basis change on parities and construct another function h, the corollary will follow by applying Lemma 2.7 on h.

Recall that we denote both a subset of $[n]$ and the corresponding indicator vector in $\mathbb{F}_2^n$, by the same notation.

Let $\mathcal{S} = \{S_1, \ldots, S_{r(f)}\}$, extend $\mathcal{S}$ to $\mathcal{S}' = \{S_1, \ldots, S_{r(f)}, S_{r(f)+1}, \ldots, S_n\}$, a complete basis of $\mathbb{F}_2^n$. Observe that $\widehat{f}(S_i) = 0$, for $i \in \{r(f)+1, \ldots, n\}$ (since $\mathcal{S}$ spans supp(f)). Fix the change of basis matrix $B \in \mathbb{F}_2^{n \times n}$ with i-th column as S_i, $i \in [n]$.

Consider the function $h : \{-1,1\}^n \to \mathbb{R}$ satisfying $\widehat{h}(\alpha) = \widehat{f}(B\alpha)$, for all $\alpha \in \mathbb{F}_2^n$. By Claim A.4, h is Boolean and $\deg_{\mathbb{F}_2}(h) = \deg_{\mathbb{F}_2}(f)$. Using Lemma 2.7, $\sum_{i \in [n]} |\widehat{h}(\{i\})| = O(\delta(f)d)$. From the definition of h, $\widehat{h}(e_i) = \widehat{f}(S_i)$ for $i \in [r(f)]$ and $\widehat{h}(e_i) = 0$ for $i \in \{r(f)+1, \ldots, n\}$, we have $\sum_{S \in \mathcal{S}} |\widehat{f}(S)| = O(\delta(f)d)$. The second equality in the statement of the lemma follows from Lemma A.3. ◀

Proof of Lemma 2.5. Observe that every term on the left hand side of Corollary 2.8 is bigger than $1/k''(f)$, giving the required lower bound on $\delta(f)$ and finishing the proof of Lemma 2.5. ◀

Proof of Theorem 1.5. From Lemma 2.5 we have $\delta(f) = \Omega\left(\frac{r(f)}{k''(f)\log k(f)}\right)$, and from Theorem 1.2 we have $\delta(f) = \Omega\left(\frac{r(f)^2}{k(f)\log^2 k(f)}\right)$. ◀

The following corollary combines the lower bounds on $\delta(f)$ from Theorem 1.5 and Lemma 1.1 by setting $k''(f)$ as the threshold.

▶ **Corollary 2.9.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function such that $k(f) > 1$. Then,*

$$\delta(f) = \Omega\left(\max\left\{\frac{r(f)^2}{k(f)\log^2 k(f)}, \frac{r(f)}{k''(f)\log k(f)}, \frac{\sqrt{r(f)}}{k''(f)\log(k''(f)^2/r(f)}\right\}\right).$$

3 Proof techniques for upper bound results

In this section we give the overview of our two upper bound results, Theorems 1.4 and 1.6. For presenting the overview of the proofs of these theorems we will use (ρ,κ)-k'-plots (Figure 1) and (ρ,κ)-k''-plots (Figure 2), respectively. In an (ρ,κ)-k'-plot ((ρ,κ)-k''-plot, respectively) we will refer to the "intersection point" as the point of intersection between the k-line and k'-curve (the point of intersection between the k-line and k''-curve, respectively). Which intersection point we are referring to should be clear from the context.

3.1 Proof techniques for Theorem 1.4

To prove Theorem 1.4, we split our goal into two natural parts: constructing functions on the k-line and constructing functions on the k'-curve. Both the classes of functions are modifications of the Addressing function (Definition A.10). In these modifications, all or some of the target variables of the Addressing function are replaced with an AND function or a Bent function or a combination of them. We first provide a description of some functions that lie on the intersection point. While we do not require this, we choose to describe these functions in order to provide more intuition.

Construction of functions at the intersection point in any (ρ,κ)-k'-plot. Note that a function lies at the intersection point when

$$k'(f) = \frac{k(f)\log(k(f))}{r(f)}. \tag{14}$$

Thus, we want to construct a function f with $k(f) = \Theta(\kappa)$, $r(f) = \Theta(\rho)$, $k'(f) = \Theta\left(\frac{\kappa\log\kappa}{\rho}\right)$ and $\delta(f) = \rho^2/\kappa(\log^2\kappa)$. In particular, we want to construct functions for all ρ,κ satisfying $\log\kappa \le \rho \le \kappa^{\frac{1}{2}}$. Note that, the Addressing function $\mathsf{AD}_t : \{-1,1\}^{\log t + t} \to \{-1,1\}$ has sparsity t^2, rank $(t + \log t)$, max-supp-entropy t and weight $1/2$ (Observation A.19) and thus, AD_t satisfies Equation (14). This only gives functions on the intersection point on all (ρ,κ)-k'-plots where $\rho = \Theta(\sqrt{\kappa})$, while we have to exhibit such functions for all (ρ,κ)-k'-plots where $\log\kappa \le \rho = O(\sqrt{\kappa})$.

Our next step is to tweak AD_t in such a way that the rank of the new function f does not change significantly while the sparsity and max-supp-entropy both increase by the same multiplicative factor. This would ensure that the resulting function satisfies Equation (14). If the resulting function's weight decreases to the required value, we would have a function at the intersection point.

In order to tweak AD_t, we consider a special kind of composed function $f := \mathsf{AD}_t \circ_{\text{target}} g$ (see Definition A.11 for a precise definition) obtained by replacing each target variable in the addressing function with a function g where each copy of g acts on a set of new variables. We prove a *composition lemma* (Lemma B.1) that gives the properties of such composed functions. Due to the structure of the Fourier spectrum of the Addressing function, Lemma B.1 gives us $r(f) \approx t \cdot r(g)$, $k(f) \approx t^2 \cdot k(g)$, $k'(f) = t \cdot k'(g)$ and $\delta(f) = \delta(g)$.

So, if g is a function on a small number of variables (say $\log t'$) with near-maximal sparsity and max-supp-entropy ($\Theta(t')$), then the resulting function satisfies Equation (14). The AND function is a natural choice for g. We denote the resulting function by $\mathsf{AD}_{t,t'}$, and this is a function at the intersection point for all plots by suitably varying t and t'.

Constructing functions on the k-line. We start with $\mathsf{AD}_{t,t'}$, the function at the intersection point in (ρ, κ)-k'-plots. We modify $\mathsf{AD}_{t,t'}$ in such a way that its sparsity, rank and weight do not change much, while the max-supp-entropy increases. We replace a single $\mathsf{AND}_{\log t'}$ in $\mathsf{AD}_{t,t'}$ by $\mathsf{AND}_{\log a}$ for some suitable $a > t$, denote the new function by $\mathsf{AD}_{t,t',a}$. A suitable setting of the parameters t, t' and a yields functions on the k-line for all plots (Claim B.2).

Constructing functions on the k'-curve of the (ρ, κ)-k'-plot. We start with $\mathsf{AD}_{t,t'}$ at the intersection point on $(\rho, \kappa/\ell)$-k'-plot (for some parameter $\ell > 0$). We modify $\mathsf{AD}_{t,t'}$ in such a way that its rank and weight do not change, the sparsity increases by a multiplicative factor of ℓ and the max-supp-entropy increases by a factor of $\sqrt{\ell}$. The new function f will be on the k'-curve in the (ρ, κ)-k'-plot because $\frac{k(f)}{k'(f)^2} = \frac{k(\mathsf{AD}_{t,t'})}{k'(\mathsf{AD}_{t,t'})^2} = \delta(\mathsf{AD}_{t,t'}) = \delta(f)$. Note that $k'(f) \approx \frac{\kappa \log(\kappa)}{\rho\sqrt{\ell}}$, thus making ℓ suitably large yields functions on the k'-curve for all $\rho \leq \kappa' \leq \frac{\kappa \log(\kappa)}{\rho}$ for all plots.

We now change $\mathsf{AD}_{t,t'}$ to have the properties mentioned above. We modify each $\mathsf{AND}_{\log t'}$ in $\mathsf{AD}_{t,t'}$ as follows: replace a single variable x by $x \cdot B$, where B is a bent function on $\log \ell$ new variables. We denote this new inner function by $\mathsf{AB}_{t',\ell}$, and $\mathsf{AD}_t \circ_{\text{target}} \mathsf{AB}_{t',\ell}$ by $\mathsf{AAB}_{t,t',\ell}$. The effect of changing $\mathsf{AND}_{\log t'}$ to $\mathsf{AB}_{t',\ell}$ keeps its rank and weight roughly the same, while increasing its sparsity by a factor of ℓ and increasing its max-supp-entropy by a factor of $\sqrt{\ell}$. We show, using our composition lemma (Lemma B.1), that the properties of $\mathsf{AD}_t \circ_{\text{target}} \mathsf{AND}_{\log t'}$ and $\mathsf{AD}_t \circ_{\text{target}} \mathsf{AB}_{t',\ell}$ change in a similar fashion. Thus, a suitable setting of the parameters t, t', ℓ yields functions on the k'-curve for all plots (Claim B.3).

3.2 Proof techniques for Theorem 1.6

We split our goal into two parts: constructing functions on the k-line when $\frac{\kappa}{\rho} \log \kappa \leq \kappa'' \leq \kappa$, and constructing functions on the k''-curve when $\kappa \leq \frac{\kappa}{\rho} \log \kappa$. To construct functions on the k-line, we use the functions $\mathsf{AD}_{t,t',a}$ constructed for the proof of Theorem 1.4, since $k'(\mathsf{AD}_{t,t',a}) = k''(\mathsf{AD}_{t,t',a})$.

For constructing functions on the k''-curve, we need to construct functions f such that

$$\delta(f) = \Theta\left(\frac{r(f)}{k''(f)\log\left(k''(f)/r(f)\right)}\right). \tag{15}$$

We will use a similar technique as in our construction of functions on the k'-curve in Theorem 1.4. We start from the function $\mathsf{AD}_{t,t'}$ at the intersection point. Note that $\mathsf{AD}_{t,t'}$ satisfies Equation (15). We modify $\mathsf{AD}_{t,t'}$ such that the rank, weight and max-rank-entropy changes very little but the sparsity increases by a multiplicative parameter 2^p. We achieve this by replacing a variable (say x) in $\mathsf{AD}_{t,t'}$ with $x \cdot \mathsf{AND}(y_1, \ldots, y_p)$, where x and y_is are all variables in $\mathsf{AD}_{t,t'}$, but for any i, x and y_i do not appear in the same monomial (Claim B.4). The new function f still satisfies Equation (15). This places f on the k''-curve in a plot corresponding to the same rank as that of $\mathsf{AD}_{t,t'}$, but where the sparsity increases by a factor of 2^p. By suitably setting p, t and t', we obtain functions on the k''-curve for all plots. This proves the second bound in Theorem 1.6.

4 Conclusions

In this paper, for Boolean functions f, we study the relationship between weight and other Fourier-analytic measures namely rank, sparsity, max-supp-entropy and max-rank-entropy. For a threshold $t > 0$, Chang's lemma gives a lower bound on the weight of a Boolean function f in terms of $\dim\left(\left\{S \subseteq [n] : |\widehat{f}(S)| \geq \frac{1}{t}\right\}\right)$. We consider three natural thresholds t in Chang's lemma, namely $k(f)$, $k'(f)$ and $k''(f)$, yielding three lower bounds on weight in terms of these measures. We prove new lower bounds on weight in Theorems 1.3 and 1.5, and our bounds dominate all the above-mentioned bounds from Chang's lemma for a wide range of parameters.

When $\log k(f) = \Theta(r(f))$, the function $f = \mathsf{AND}$ already shows that all the above lower bounds are tight. To consider all other feasible relationships between $k(f)$ and $r(f)$, we divide our investigation of these lower bounds into two different parts. In the first part, we vary over all feasible settings of $r(f)$, $k(f)$ and $k'(f)$, and construct functions that witness tightness of our lower bounds in Theorem 1.3 for nearly all such feasible settings (Theorem 1.4). In the second part, we vary over all feasible settings of $r(f)$, $k(f)$ and $k''(f)$, and construct functions that witness near-tightness of our lower bounds in Theorem 1.5 for nearly all such feasible settings (Theorem 1.6). These functions are constructed by carefully composing the Addressing function with suitable inner functions. We show a composition lemma (Lemma B.1), which relates the properties of the composed function with those of the inner functions; this allows us to come up with functions that match our lower bounds.

We also construct functions for which our lower bounds are asymptotically stronger than the lower bounds obtained from Chang's lemma for all choices of threshold (see Claim 1.7). All functions that we construct in this work might be of independent interest.

Open Problems. Since our proof of Theorem 1.2 is a generalization of the proof of the upper bound $r(f) = O(\sqrt{k(f)}\log k(f))$ due to Sanyal [25], it sheds light on the presence of the $\log k$ factor in Sanyal's upper bound. This still leaves the following question open: do there exist Boolean functions f for which $r(f) = \omega(\sqrt{k(f)})$?

There are some ranges of parameters where we were not able to construct functions with upper bounds matching our lower bounds from Theorem 1.5. It will be interesting to see if our techniques can be extended to cover these ranges as well.

All thresholds t considered for Chang's lemma in this work satisfy $\dim(\{S \subseteq [n] : |\widehat{f}(S)| \geq \frac{1}{t}\}) = r(f)$. It is an interesting problem to obtain Chang's-lemma-type bounds for thresholds for which this dimension is strictly less than $r(f)$.

References

1 Srinivasan Arunachalam, Sourav Chakraborty, Troy Lee, Manaswi Paraashar, and Ronald de Wolf. Two new results about quantum exact learning. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 16:1–16:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

2 Eli Ben-Sasson, Noga Ron-Zewi, Madhur Tulsiani, and Julia Wolf. Sampling-based proofs of almost-periodicity results and algorithmic applications. In *41st International Colloquium on Automata, Languages, and Programming ICALP 2014*, volume 8572 of *Lecture Notes in Computer Science*, pages 955–966. Springer, 2014.

3 Anna Bernasconi and Bruno Codenotti. Spectral analysis of Boolean functions as a graph eigenvalue problem. *IEEE Trans. Computers*, 48(3):345–351, 1999. doi:10.1109/12.755000.

4 Thomas F Bloom. A quantitative improvement for Roth's theorem on arithmetic progressions. *Journal of the London Mathematical Society*, 93(3):643–663, 2016.

5 Thomas F Bloom and Olof Sisask. Breaking the logarithmic barrier in Roth's theorem on arithmetic progressions. *arXiv preprint*, 2020. arXiv:2007.03528.

6 Sourav Chakraborty, Nikhil S. Mande, Rajat Mittal, Tulasimohan Molli, Manaswi Paraashar, and Swagato Sanyal. Tight chang's-lemma-type bounds for boolean functions. *CoRR*, abs/2012.02335, 2020. arXiv:2012.02335.

7 Siu On Chan, James R. Lee, Prasad Raghavendra, and David Steurer. Approximate constraint satisfaction requires large LP relaxations. *J. ACM*, 63(4):34:1–34:22, 2016.

8 Mei-Chu Chang. A polynomial bound in Freiman's theorem. *Duke mathematical journal*, 113(3):399–419, 2002.

9 Eshan Chattopadhyay, Pooya Hatami, Shachar Lovett, and Avishay Tal. Pseudorandom generators from the second Fourier level and applications to AC0 with parity gates. In *10th Innovations in Theoretical Computer Science Conference, ITCS*, pages 22:1–22:15, 2019. doi:10.4230/LIPIcs.ITCS.2019.22.

10 Ehud Friedgut, Jeff Kahn, Gil Kalai, and Nathan Keller. Chvátal's conjecture and correlation inequalities. *J. Comb. Theory, Ser. A*, 156:22–43, 2018.

11 Parikshit Gopalan, Ryan O'Donnell, Rocco A. Servedio, Amir Shpilka, and Karl Wimmer. Testing Fourier dimensionality and sparsity. *SIAM J. Comput.*, 40(4):1075–1100, 2011. doi:10.1137/100785429.

12 Ben Green. Arithmetic progressions in sumsets. *Geometric and Functional Analysis GAFA*, 12(3):584–597, 2002.

13 Ben Green. Some constructions in the inverse spectral theory of cyclic groups. *Combinatorics, Probability and Computing*, 12(2):127–138, 2003.

14 Ben Green. Spectral structure of sets of integers. In *Fourier analysis and convexity*, pages 83–96. Springer, 2004.

15 Ben Green and Imre Z. Ruzsa. Freiman's theorem in an arbitrary abelian group. *Journal of the London Mathematical Society*, 75(1):163–175, 2007.

16 Ben Green and Tom Sanders. Boolean functions with small spectral norm. *Geometric and Functional Analysis GAFA*, 18:144–162, 2008.

17 Tom Gur and Omer Tamuz. Testing Booleanity and the uncertainty principle. *Chic. J. Theor. Comput. Sci.*, 2013, 2013.

18 Pooya Hatami, Raghav Kulkarni, and Denis Pankratov. Variations on the sensitivity conjecture. *Theory of Computing*, 4:1–27, 2011. doi:10.4086/toc.gs.2011.004.

19 Kaave Hosseini, Shachar Lovett, and Grigory Yaroslavtsev. Optimality of linear sketching under modular updates. In *34th Computational Complexity Conference, CCC 2019*, volume 137 of *LIPIcs*, pages 13:1–13:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

20 Russell Impagliazzo, Cristopher Moore, and Alexander Russell. An entropic proof of Chang's inequality. *SIAM J. Discret. Math.*, 28(1):173–176, 2014.

21 Shachar Lovett. Communication is bounded by root of rank. *Journal of the ACM (JACM)*, 63(1):1–9, 2016.

22 Ashley Montanaro and Tobias Osborne. On the communication complexity of XOR functions. *CoRR*, abs/0909.3392, 2009. arXiv:0909.3392.

23 Tom Sanders. Additive structures in sumsets. *Mathematical Proceedings of the Cambridge Philosophical Society*, 144(2):289–316, 2008.

24 Tom Sanders. On Roth's theorem on progressions. *Annals of Mathematics*, 174:619–636, 2011.

25 Swagato Sanyal. Fourier sparsity and dimension. *Theory of Computing*, 15(11):1–13, 2019. doi:10.4086/toc.2019.v015a011.

26 Hing Yin Tsang, Chung Hoi Wong, Ning Xie, and Shengyu Zhang. Fourier sparsity, spectral norm, and the log-rank conjecture. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 658–667, 2013. doi:10.1109/FOCS.2013.76.

A Preliminaries

All logarithms in this paper are taken to be base 2. We use the notation $[n]$ to denote the set $\{1, 2, \ldots, n\}$. When necessary, we assume t is a power of 2. We use the notation 1^n (respectively, $(-1)^n$) to denote the n-bit string $(1, 1, \ldots, 1)$ (respectively, $(-1, -1, \ldots, -1)$).

For a function $f : \{-1, 1\}^n \to \{-1, 1\}$, its $\mathbb{F}_2$-degree, denoted by $\deg_{\mathbb{F}_2}(f)$, is the degree of its unique $\mathbb{F}_2$-polynomial representation. Throughout this paper, we often identify subsets of $[n]$ with their corresponding characteristic vectors in $\mathbb{F}_2^n$. Thus when we refer to linear algebraic measures of a collection of subsets of $[n]$, we mean the measure on the corresponding subset of $\mathbb{F}_2^n$ (where $\mathbb{F}_2^n$ is viewed as an $\mathbb{F}_2$-vector space).

Throughout this paper, we assume that f is not a constant function or a parity or a negative parity, unless mentioned otherwise.

A.1 Fourier analysis of Boolean functions

Consider the vector space of functions from $\{-1, 1\}^n$ to $\mathbb{R}$ equipped with the following inner product.

$$\langle f, g \rangle := \frac{1}{2^n} \sum_{x \in \{-1,1\}^n} f(x)g(x).$$

For a set $S \subseteq [n]$, define a *parity* function (which we also refer to as *characters*) $\chi_S : \{-1, 1\}^n \to \{-1, 1\}$ by $\chi_S(x) = \prod_{i \in S} x_i$. The set of parity functions $\{\chi_S : S \subseteq [n]\}$ forms an orthonormal basis for this vector space. Hence, every function $f : \{-1, 1\}^n \to \mathbb{R}$ has a unique representation as

$$f = \sum_{S \subseteq [n]} \widehat{f}(S)\chi_S,$$

where $\widehat{f}(S) = \langle f, \chi_S \rangle$ for all $S \subseteq [n]$. The coefficients $\left\{\widehat{f}(S) : S \subseteq [n]\right\}$ are called the *Fourier coefficients* of f. Define the Fourier ℓ_1-norm of a function $f : \{-1, 1\}^n \to \mathbb{R}$ by $\|\widehat{f}\|_1 := \sum_{S \subseteq [n]} |\widehat{f}(S)|$. The Fourier support of f, denoted by $\mathrm{supp}(f)$, is defined as

$$\mathrm{supp}(f) = \left\{S \subseteq [n] : \widehat{f}(S) \neq 0\right\}.$$

▶ Remark A.1. In the literature, Fourier support is generally denoted by $\mathrm{supp}(\widehat{f})$. For ease of notation we drop the hat symbol above f. A similar convention has been adopted in the remaining parts of the paper.

Let $f : \{-1, 1\}^n \to \mathbb{R}$ be any function. The Fourier sparsity of f, denoted by $k(f)$, is defined as $k(f) = |\mathrm{supp}(f)|$. For simplicity we assume that $k(f) \geq 2$ for all Boolean functions f considered in this paper (unless explicitly mentioned otherwise). We often simply refer to the Fourier sparsity as *sparsity*. For ease of notation, we sometimes abuse notation and say that the elements of the Fourier support of f are the characters $\left\{\chi_S : S \subseteq [n], \widehat{f}(S) \neq 0\right\}$, rather than the corresponding sets.

We require the following lemma (see, for example, [17]).

▶ **Lemma A.2** (Uncertainty Principle). *Let $f : \{-1,1\}^n \to \mathbb{R}$ be a polynomial and let U_n denote the uniform distribution on $\{-1,1\}^n$. Then,*

$$\Pr_{x \sim U_n}[f(x) \neq 0] \geq \frac{1}{k(f)}.$$

We also require the following lemma relating the $\mathbb{F}_2$-degree of a Boolean function and its Fourier sparsity (see, for example, [3]).

▶ **Lemma A.3.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function with $k(f) > 1$. Then,*

$$\deg_{\mathbb{F}_2}(f) \leq \log k(f).$$

The next claim shows that $\deg_{\mathbb{F}_2}(f)$ does not change under a change of basis over the Fourier domain.

▷ Claim A.4. Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function and let $B \in \mathbb{F}_2^{n \times n}$ be an invertible matrix. Define the function $f_B : \{-1,1\}^n \to \mathbb{R}$ as

$$\widehat{f_B}(\alpha) = \widehat{f}(B\alpha) \quad \text{for all } \alpha \in \mathbb{F}_2^n,$$

Then f_B is Boolean valued and $\deg_{\mathbb{F}_2}(f_B) = \deg_{\mathbb{F}_2}(f)$.

The following corollary follows from [9, Theorem 13] and Lemma A.3.

▶ **Corollary A.5.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function, and let $\mathcal{S} \subseteq \mathrm{supp}(f)$ be a basis of $\mathrm{span}(\mathrm{supp}(f))$. Then,*

$$\sum_{S \in \mathcal{S}} |\widehat{f}(S)| \leq 4 \log k(f).$$

We now define notions of restriction of a function $f : \{-1,1\}^n \to \{-1,1\}$ to a subset $A \subseteq \{-1,1\}^n$.

▶ **Definition A.6** (Restriction). *Let $f : \{-1,1\}^n \to \{-1,1\}$ and $A \subseteq \{-1,1\}^n$. The restriction of f to A is the function $f|_A : A \to \{-1,1\}$ defined as $f|_A(x) = f(x)$ for all $x \in A$.*

▶ **Definition A.7** (Affine Restriction). *Let $f : \{-1,1\}^n \to \{-1,1\}$, let Γ be a set of parities and $b \in \{-1,1\}^\Gamma$ be an assignment to these parities. Define the function $f|_{(\Gamma,b)}$ to be the restriction of f to the affine subspace obtained by fixing parities in Γ according to b. That is,*

$$f|_{(\Gamma,b)} := f|_{\{x \in \{-1,1\}^n : \chi_\gamma(x) = b_\gamma \text{ for all } \gamma \in \Gamma\}}.$$

A.2 Fourier expansions and properties of some standard functions

For any integer $n > 0$, define the function $\mathsf{AND}_n : \{-1,1\}^n \to \{-1,1\}$ by $\mathsf{AND}_n(x) = -1$ if $x = (-1)^n$, and 1 otherwise. We drop the subscript n when it is clear from the context.

▶ **Definition A.8** (Bent functions). *A function $f : \{-1,1\}^n \to \{-1,1\}$ is said to be a bent function if $|\widehat{f}(S)| = |\widehat{f}(T)|$ for all $S, T \subseteq [n]$.*

▶ **Definition A.9** (Indicator function). *For any integer $n \geq 1$ and $b \in \{-1,1\}^n$, define the function $\mathbb{I}_b : \{-1,1\}^n \to \{0,1\}$ by*

$$\mathbb{I}_b(x) = \begin{cases} 1 & x = b, \\ 0 & \textit{otherwise}. \end{cases}$$

▶ **Definition A.10** (Addressing function). *For any integer $t \geq 2$, define the Addressing function* $\mathsf{AD}_t : \{-1,1\}^{\log t} \times \{-1,1\}^t \to \{-1,1\}$ *by*

$$\mathsf{AD}_t(x, y) = y_{\mathrm{bin}(x)},$$

where $x \in \{-1,1\}^{\log t}$ and $y \in \{-1,1\}^t$, and $\mathrm{bin}(x)$ *denotes the integer in* $[t]$ *whose binary representation is given by x (where -1's are viewed as 1 in the string x, and 1's are viewed as 0). We refer to the x-variables as* addressing variables*, and the y-variables as* target variables*.*

We next define a way of modifying the Addressing function that is of use to us. In this modification, we replace target variables by functions, each acting on disjoint variables.

▶ **Definition A.11** (Composed addressing functions). *Let $t \geq 2$, $\ell_1, \ldots, \ell_t \geq 1$ be any integers. Let $g_i : \{-1,1\}^{\ell_i} \to \{-1,1\}$ be any functions for $i \in [t]$. Define the function* $\mathsf{AD}_t \circ_{\mathrm{target}} (g_1, \ldots, g_t) : \{-1,1\}^{\log t} \times \{-1,1\}^{\ell_1 + \cdots + \ell_t} \to \{-1,1\}$ *by*

$$\mathsf{AD}_t \circ_{\mathrm{target}} (g_1, \ldots, g_t)(x, y_1, \ldots, y_t) = \mathsf{AD}_t(x, g_1(y_1), \ldots, g_t(y_t)),$$

where $x \in \{-1,1\}^{\log t}$ and $y_i \in \{-1,1\}^{\ell_i}$ for all $i \in [t]$.

For any function $g : \{-1,1\}^s \to \{-1,1\}$, we use the notation $\mathsf{AD}_t \circ_{\mathrm{target}} g$ to denote the function $\mathsf{AD}_t \circ_{\mathrm{target}} (g, g, \ldots, g) : \{-1,1\}^{\log t} \times \{-1,1\}^{ts} \to \{-1,1\}$.

A.3 Fourier-analytic measures of Boolean functions

We now introduce a few Fourier-analytic measures on Boolean functions that we use throughout the rest of the paper, and state some important relationships between them. Recall that we use the notation $\dim(S)$ to denote the dimension of the span of the set S.

▶ **Definition A.12** (Fourier rank). *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function. Define the Fourier rank of f, denoted $r(f)$, by*

$$r(f) = \dim(\mathrm{supp}(f)).$$

We often refer to Fourier rank as simply *rank*. Sanyal [25] showed the following upper bound on the rank of Boolean functions in terms of their sparsity.

▶ **Theorem A.13** ([25, Theorem 1.2]). *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function. Then*

$$r(f) = O(\sqrt{k(f)} \log k(f)).$$

We require the following observation which gives a simple upper bound on the rank of a Boolean function.

▶ **Observation A.14.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function and Γ be a set of parities. If for all $b \in \{-1,1\}^\Gamma$ the restricted function $f|_{(\Gamma,b)}$ is constant then $r(f) \leq |\Gamma|$.*

Recall that for any function $f : \{-1,1\}^n \to \{-1,1\}$ and any real $t > 0$, we define $\mathcal{S}_t := \{S \subseteq [n] : |\widehat{f}(S)| \geq 1/t\}$ (we suppress the dependence of $\mathcal{S}_t$ on f as the underlying function will be clear from context).

▶ **Definition A.15.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function. Define the Fourier max-supp-entropy of f, denoted $k'(f)$, by*

$$k'(f) := \operatorname*{argmin}_t \{\mathcal{S}_t = \operatorname{supp}(f)\}.$$

Equivalently,

$$k'(f) := \max_{S \in \operatorname{supp}(f)} \left\{ \frac{1}{|\widehat{f}(S)|} \right\}.$$

Define the Fourier max-rank-entropy of f, denoted $k''(f)$, by

$$k''(f) := \operatorname*{argmin}_t \{\dim(\mathcal{S}_t) = r(f)\}.$$

We often refer to the Fourier max-supp-entropy and Fourier max-rank-entropy as simply *max-supp-entropy* and *max-rank-entropy*, respectively.

▶ **Lemma A.16** (Relationships between parameters)**.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function. Then the following inequalities hold.*

1. $\log k(f) \leq r(f) = O(\sqrt{k(f)} \log k(f))$.
2. $\sqrt{k(f)} \leq k'(f) \leq k(f)/2$.
3. $\max\left\{\sqrt{r(f)}, r(f)/(4 \log k(f))\right\} \leq k''(f) \leq k'(f)$.

▷ Claim A.17. Let $f : \{-1,1\}^n \to \{-1,1\}$ a function with $k(f) \geq 2$. Then

$$\delta(f) = \Omega\left(\frac{k(f)}{k'(f)^2}\right).$$

▷ Claim A.18. Let $f : \{-1,1\}^n \to \{-1,1\}$ be any function. Then

$$\|\widehat{f}\|_1 \leq 3\sqrt{k(f)\delta(f)}.$$

We require the following observation about the rank, sparsity, max-supp-entropy, max-rank-entropy and weight of the addressing function, AD_t, which follows immediately from definitions and first principles. We omit its proof.

▶ **Observation A.19.** *Let $t \geq 2$ be any positive integer. Then the rank, sparsity, max-supp-entropy, max-rank-entropy and weight of AD_t are $(t + \log t)$, t^2, t, t and $1/2$, respectively.*

B Upper bound proofs

The following lemma is a useful tool for our upper bounds. We refer the reader to [6, Section 6.2.1] in the full version of our paper for a proof.

▶ **Lemma B.1** (Composition lemma)**.** *Let $t \geq 2, m \geq 1$ be any positive integers, and let $g : \{-1,1\}^m \to \{-1,1\}$ be a non-constant function such that there exists a non-empty set $S \subseteq [m]$ with $0 \neq |\widehat{g}(S)| \leq |\widehat{g}(\emptyset)|$. Let $f : \{-1,1\}^{\log t + mt} \to \{-1,1\}$ be defined as*

$$f = \mathsf{AD}_t \circ_{\text{target}} g.$$

Then

$$r(f) = t \cdot r(g) + \log t, \tag{16}$$

$$k(f) = 1 + t^2(k(g) - 1), \tag{17}$$

$$k'(f) = t \cdot k'(g), \tag{18}$$

$$k''(f) = t \cdot k''(g), \tag{19}$$

$$\delta(f) = \delta(g). \tag{20}$$

B.1 Setting parameters in our constructed functions

In this section we state the main claims that go into proving Theorems 1.4 and 1.6. Recall that these theorems require us to exhibit functions which achieve certain bounds. Claims B.2 and B.3 correspond to the bounds in Theorem 1.4. Claims B.4 and B.5 correspond to the bounds in Theorem 1.6. All functions referred to below are informally defined in Section 3. See the full version of our paper [6, Section 6.1] for formal definitions and [6, Sections 6.2, 6.3] for proofs of these claims.

▷ Claim B.2. For all $\rho, \kappa, \kappa' \in \mathbb{N}$ such that κ is sufficiently large, for all $\epsilon > 0$ such that $\log \kappa \leq \rho \leq \kappa^{\frac{1}{2}-\epsilon}$ and $\frac{\kappa \log \kappa}{\rho} \leq \kappa' \leq \kappa$, for $t = \frac{2\rho}{\log \kappa}$, $t' = \frac{\kappa \log^2 \kappa}{\rho^2}$ and $a = \frac{2\kappa' \log \kappa}{\rho}$,

- $\Omega(\epsilon\rho) = r(\mathsf{AD}_{t,t',a}) = O(\rho)$.
- $k(\mathsf{AD}_{t,t',a}) = \Theta(\kappa)$.
- $k'(\mathsf{AD}_{t,t',a}) = \Theta(\kappa')$.
- $\delta(\mathsf{AD}_{t,t',a}) = \Theta\left(\frac{1}{\kappa}\left(\frac{\rho}{\log \kappa}\right)^2\right)$.

▷ Claim B.3. For all $\rho, \kappa, \kappa' \in \mathbb{N}$ such that κ is sufficiently large, for all constants $\epsilon > 0$, such that $\kappa^{1/2} \leq \kappa' \leq (\kappa \log \kappa)/\rho$ and $\log \kappa \leq \rho \leq \kappa^{\frac{1}{2}-\epsilon}$ for $t = \frac{2\rho}{\log \kappa}$, $t' = \frac{4\kappa'^2}{\kappa}$ and $\ell = 2\left(\frac{\kappa \log \kappa}{\kappa' \rho}\right)^2$,

- $\Omega(\epsilon\rho) = r(\mathsf{AAB}_{t,t',\ell}) = O(\rho)$.
- $k(\mathsf{AAB}_{t,t',\ell}) = \Theta(\kappa)$.
- $k'(\mathsf{AAB}_{t,t',\ell}) = \Theta(\kappa')$.
- $\delta(f) = O\left(\frac{\kappa}{\kappa'^2}\right)$.

▷ Claim B.4. For all $\rho, \kappa, \kappa'' \in \mathbb{N}$ such that κ is sufficiently large, for all constants $\epsilon > 0$ such that $\log \kappa \leq \rho \leq \kappa^{1/2-\epsilon}$, $e\rho \leq \kappa'' \leq \frac{\kappa \log \kappa}{\rho}$, for $t = \frac{2\rho}{\log(\kappa''/\rho)}$, $t' = \frac{\kappa''}{\rho} \log(\kappa''/\rho)$, $p = \log\left(\frac{4\kappa}{\kappa''}\right)$,

- $r(\mathsf{mAD}_{t,t',p}) = \Theta(\rho)$.
- $\Omega(\kappa) = k(\mathsf{mAD}_{t,t',p}) = O(\kappa/\epsilon)$.
- $k''(\mathsf{mAD}_{t,t',p}) = \Theta(\kappa'')$.
- $\delta(\mathsf{mAD}_{t,t',p}) = \frac{\rho}{\kappa'' \log(\kappa''/\rho)}$.

▷ Claim B.5. For all $\rho, \kappa, \kappa' \in \mathbb{N}$ such that κ is sufficiently large, for all $\epsilon > 0$ such that $\log \kappa \leq \rho \leq \kappa^{\frac{1}{2}-\epsilon}$ and $\frac{\kappa \log \kappa}{\rho} \leq \kappa'' \leq \kappa$, there exists a constant $c \geq 1$ such that the following holds for $t = \frac{2\rho}{\log \kappa}$, $t' = \frac{c\kappa \log^2 \kappa}{\rho^2}$ and $a = \frac{2c\kappa'' \log \kappa}{\rho}$.

- $\Omega(\epsilon\rho) = r(\mathsf{AD}_{t,t',a}) = O(\rho)$.
- $k(\mathsf{AD}_{t,t',a}) = \Theta(\kappa)$.
- $k''(\mathsf{AD}_{t,t',a}) = \Theta(\kappa'')$.
- $\delta(\mathsf{AD}_{t,t',a}) = \Theta\left(\frac{1}{\kappa}\left(\frac{\rho}{\log \kappa}\right)^2\right)$.

Approximate Trace Reconstruction via Median String (In Average-Case)

Diptarka Chakraborty ✉
National University of Singapore, Singapore

Debarati Das ✉
Basic Algorithm Research Copenhagen (BARC), University of Copenhagen, Denmark

Robert Krauthgamer ✉
Weizmann Institute of Science, Rehovot, Israel

Abstract

We consider an *approximate* version of the trace reconstruction problem, where the goal is to recover an unknown string $s \in \{0,1\}^n$ from m traces (each trace is generated independently by passing s through a probabilistic insertion-deletion channel with rate p). We present a deterministic near-linear time algorithm for the average-case model, where s is random, that uses only *three* traces. It runs in near-linear time $\tilde{O}(n)$ and with high probability reports a string within edit distance $\tilde{O}(p^2 n)$ from s, which significantly improves over the straightforward bound of $O(pn)$.

Technically, our algorithm computes a $(1+\epsilon)$-approximate median of the three input traces. To prove its correctness, our probabilistic analysis shows that an approximate median is indeed close to the unknown s. To achieve a near-linear time bound, we have to bypass the well-known dynamic programming algorithm that computes an optimal median in time $O(n^3)$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Trace Reconstruction, Approximation Algorithms, Edit Distance, String Median

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.11

Related Version *Full Version*: `https://arxiv.org/abs/2107.09497`

Funding *Diptarka Chakraborty*: Work partially supported by NUS ODPRT Grant, WBS No. R-252-000-A94-133.
Robert Krauthgamer: Work partially supported by ONR Award N00014-18-1-2364, the Israel Science Foundation grant #1086/18, and a Minerva Foundation grant, and by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center.

1 Introduction

Trace Reconstruction. One of the most common problems in statistics is to estimate an unknown parameter from a set of noisy observations (or samples). The main objectives are (1) to use as few samples as possible, (2) to minimize the estimation error, and (3) to design an efficient estimation algorithm. One such parameter-estimation problem is *trace reconstruction*, where the unknown quantity is a string $s \in \Sigma^n$, and the observations are independent *traces*, where a trace is a string that results from s passing through some noise channel. The goal is to reconstruct s using a few traces. (Unless otherwise specified, in this paper we consider $\Sigma = \{0,1\}$.) Various noise channels have been considered so far. The most basic one only performs substitutions. A more challenging channel performs deletions. Even more challenging is the *insertion-deletion* channel, which scans the string s and keeps the next character with probability $1-p$, deletes it with probability $p/2$, or inserts a uniformly

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 11; pp. 11:1–11:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

randomly chosen symbol (without processing the next character) with probability $p/2$, for some noise-rate parameter $p \in [0, 1)$. We denote this insertion-deletion channel by $R_p(s)$, see Section 2 for a formal definition.[1]

The literature studies mostly two variants of trace reconstruction. In the *worst-case* variant, the unknown string s is an arbitrary string from Σ^n, while in the *average-case* variant, s is assumed to be drawn uniformly at random from Σ^n. The trace reconstruction problem finds numerous applications in computational biology, DNA storage systems, coding theory, etc. Starting from early 1970s [30], various other versions of this problem have been studied, including combinatorial channels [37, 38], smoothed complexity [12], coded trace reconstruction [13], and population recovery [3, 44].

We focus on the average-case variant, where it is known that $\exp(O(\log^{1/3} n))$ samples suffice to reconstruct a (random) unknown string s over the insertion-deletion channel [28]. On the other hand, a recent result [9] showed that $\tilde{\Omega}(\log^{5/2} n)$ samples are necessary, improving upon the previous best lower bound of $\tilde{\Omega}(\log^{9/4} n)$ [27]. We emphasize that all these upper and lower bounds are for *exact* trace reconstruction, i.e., for recovering the unknown string x perfectly (with no errors). A natural question proposed by Mitzenmacher [43] is whether such a lower bound on the sample complexity can be bypassed by allowing approximation, i.e., by finding a string z that is "close" to the unknown string s. One of the most fundamental measures of closeness between a pair of strings z and z', is their *edit distance*, denoted by $\mathrm{ED}(z, z')$ and defined as the minimum number of insertion, deletion, and substitution operations needed to transform z into z'. Observe that a trace generated from s via an insertion-deletion channel G_p has expected edit distance about pn from the unknown string s (see Section 3). We ask how many traces (or samples) are required to construct a string z at a much smaller edit distance from the unknown s. (Since the insertion-deletion channel has no substitutions, we also do not consider substitutions in our analysis of the edit distance.)

A practical application of average-case trace reconstruction is in the portable DNA-based data storage system. In the DNA storage system [23, 51], a file is preprocessed by encoding it into a DNA sequence. This encoded sequence is randomized using a pseudo-random sequence, and thus the final encoding sequence could be treated as a (pseudo-)random string. The stored (encoded) data is retrieved using next-generation sequencing (like single-molecule real-time sequencing (SMRT) [52] that involves $12 - 18\%$, which generates several noisy copies (traces) of the stored data via some insertion-deletion channel. The final step is to decode back the stored data with as few traces as possible. Currently, researchers use multiple sequence alignment algorithms to reconstruct the trace [55, 47]. Unfortunately, such heuristic algorithms are notoriously difficult to analyze rigorously to show a theoretical guarantee. However, the preprocessing step also involves error-correcting code to encode the strings. Thus it suffices to reconstruct the original string up to some small error (depending on the error-correcting codes used). This specific application gives one motivation to study approximate trace reconstruction.

Our main contribution is to show that it is sufficient to use only three traces to reconstruct the unknown string up to a small edit error.

[1] In the literature, an insertion-deletion channel with different probabilities for insertion and for deletion has been studied. For simplicity in exposition, we consider a single error probability throughout this paper, however our results can easily be generalized to different insertion and deletion probabilities. Another possible generalization is to allow substitutions along with insertions and deletions. Again, for simplicity, we do not consider substitutions, but with slightly more careful analysis our results could be extended.

▶ **Theorem 1.** *There is a constant $c_0 > 0$ and a deterministic algorithm that, given as input a noise parameter $p \in (0, c_0]$, and three traces from the insertion-deletion channel $R_p(s)$ for a uniformly random (but unknown) string $s \in \{0,1\}^n$, outputs in time $\tilde{O}(n)$ a string z that satisfies* $\Pr[\mathrm{ED}(s,z) \le O(p^2 \log(1/p)n)] \ge 1 - n^{-1}$.

The probability in this theorem is over the random choice of s and the randomness of the insertion-deletion channel R_p. We note that the term $\log(1/p)$ in the estimation error $\mathrm{ED}(s,z)$ can be shaved by increasing the alphabet size to $\mathrm{poly}(1/\epsilon)$. An edit error of $O(p^2 n)$ is optimal for three traces, because in expectation $O(p^2 n)$ characters of s are deleted in two of the three traces, and look as if they are inserted to s in one of the three traces (which occurs in expectation for even more characters).

Our theorem demonstrates that the number of required traces exhibits a sharp contrast between exact and approximate trace reconstruction. In fact, approximate reconstruction not only beats the $\Omega(\log^{5/2} n)$ lower bound for exact reconstruction, but surprisingly uses only *three* traces! We conjecture that the estimation error $\mathrm{ED}(s,z)$ can be reduced further using more than three traces. We believe that our technique can be useful here, but this is left open for future work.

▶ **Conjecture 2.** *The estimation error* $\mathrm{ED}(s,z)$ *in Theorem 1 can be reduced to $O(\epsilon p n)$ for arbitrarily small $\epsilon > 0$, using* $\mathrm{poly}(1/\epsilon)$ *traces.*

This conjecture holds for $\epsilon < 1/n$, as follows from known bounds for exact reconstruction [28], and perhaps suggests that a number of traces that is sub-polynomial in $1/\epsilon$ it suffices for all $\epsilon > 0$.

Median String. As mentioned earlier, a common heuristic to solve the trace reconstruction problem is multiple sequence alignment, which can be formulated equivalently (see [25] and the references therein) as the problem of finding a median under edit distance. For general context, the median problem is a classical aggregation task in data analysis; its input is a set S of points in a metric space relevant to the intended application, and the goal is to find a point (not necessarily from S) with the minimum sum of distances to points in S, i.e.,

$$\min_y \sum_{x \in S} d(y,x). \tag{1}$$

Such a point is called a *median* (or *geometric median* in a Euclidean space). For many applications, it suffices to find an *approximate median*, i.e., a point in the metric with approximately minimal objective value (1) . The problem of finding an (approximate) median has been studied extensively both in theory and in applied domains, over various metric spaces, including Euclidean [15] (see references therein for an overview), Hamming (folklore), the edit metric [53, 34, 46], rankings [19, 2, 32], Jaccard distance [14], Ulam [8], and many more [21, 41, 6].

The median problem over the *edit-distance metric* is known as the *median string* problem [33], and finds numerous applications in computational biology [25, 50], DNA storage system [23, 51], speech recognition [33], and classification [39]. This problem is known to be NP-hard [18, 46] (even W[1]-hard [46]), and can be solved by standard dynamic programming [53, 34] in time $O(2^m n^m)$ when the input has $m = |S|$ strings of length n each. From the perspective of approximation algorithms, a multiplicative 2-approximation to the median is straightforward (this works in every metric space by simply reporting the best among the input strings, i.e., $y^* \in S$ that minimizes the objective). However, no polynomial-time algorithm is known to break below 2-approximation (i.e., achieve factor $2-\delta$ for fixed $\delta > 0$) for the median string problem, despite several heuristic algorithms and results for special cases [7, 35, 20, 48, 1, 26, 42, 8].

Although the median string (or equivalently multiple sequence alignment) is a common heuristic for trace reconstruction [55, 47], to the best of our knowledge there is no definite connection known between these two problems. We show that both the problems are roughly the same in the average-case model. It is not difficult to show that any string close to the unknown string is an approximate median. To see this, we can show that for a set S of m traces of an (unknown) random string s, their optimal median objective value is at least $(1-O(\epsilon))pnm$, for any $\epsilon \in [110p\log(1/p), 1/6]$, with high probability (see the full version). On the other hand, the median objective value with respect to s itself is at most $(1+\epsilon)pnm$, for any $\epsilon > 0$, with high probability. Hence, the unknown string s is an $(1+O(\epsilon))$-approximate median of S, for any $\epsilon \in [110p\log(1/p), 1/6]$, and by the triangle inequality, every string close (in edit distance) to s is also an approximate median of S. One of the major contributions of this paper is the converse direction, showing that given a set of traces of an unknown string, any approximate median of the traces is close (in edit distance) to the unknown string. This is true even for three traces.

► **Theorem 3.** *For a large enough $n \in \mathbb{N}$ and a noise parameter $p \in (0, 0.001)$, let the string $s \in \{0,1\}^n$ be chosen uniformly at random, and let s_1, s_2, s_3 be three traces generated by the insertion-deletion channel $R_p(s)$. If x_{med} is a $(1+\epsilon)$-approximate median of $\{s_1, s_2, s_3\}$ for $\epsilon \in [110p\log(1/p), 1/6]$, then $\Pr[\mathrm{ED}(s, x_{med}) \le O(\epsilon) \cdot \mathtt{OPT}] \ge 1 - n^{-3}$, where $\mathtt{OPT}$ denotes the optimal median objective value of $\{s_1, s_2, s_3\}$.*

An immediate consequence (see Corollary 14) is that for every $3 \le m < n^{O(1)}$ traces, every $(1+\epsilon)$-approximate median x_{med} satisfies $\mathrm{ED}(s, x_{\text{med}}) \le O(\epsilon)\frac{\mathtt{OPT}}{m}$.

Thus if we could solve any of the two problems (even approximately), we get an approximate solution to the other problem. E.g., the current best (exact) trace reconstruction algorithm for the average-case [28] immediately gives us an $n^{1+o(1)}$ time algorithm to find an $(1+O(\epsilon))$-approximate median of a set of (at least $\exp(O(\log^{1/3} n))$) traces. (Note, to apply the result of [28], we need at least $\exp(O(\log^{1/3} n))$ traces.) We leverage this interplay between the two problems to design an efficient algorithm for approximate trace reconstruction. Since one can compute the (exact) median of three strings s_1, s_2, s_3 in time $O(|s_1| \cdot |s_2| \cdot |s_3|)$ [53, 34], the above theorem immediately provides us the unknown string up to some small edit error in time $O(n^3)$. We further reduce the running time to near-linear by cleverly partitioning each of the traces into polylog n-size blocks and then applying the median algorithm on these blocks. Finally, we concatenate all the block-medians to get an "approximate" unknown string, leading to Theorem 1. One may further note that Theorem 1 also provides a $(1+O(\epsilon))$-approximate median for any set of traces in the average-case (again due to Theorem 3).

Taking the smallest possible ϵ in Theorem 3, we get that for three traces generated from s, with high probability $\mathrm{ED}(s, x_{\text{med}}) \le \tilde{O}(p^2 n)$. In comparison, it is not hard to see that with high probability $\mathtt{OPT}$ is bounded by roughly $3pn$. We conjecture that as the number of traces increases, the median string converges to the unknown string s. In particular, $\mathrm{ED}(s, x_{\text{med}}) \le \epsilon n$ when using $\mathrm{poly}(1/\epsilon)$ traces (instead of just three), with high probability. We hope that our technique can be extended to prove the above conjecture, but we leave it open for future work.

The main implication of this conjecture is an $\tilde{O}(n)$ time approximate trace reconstruction algorithm, for any fixed $\epsilon > 0$, as follows. It is straightforward to extend our approximate median finding algorithm (in Section 5) to more input strings. (For brevity, we present only for three input strings.) For m strings, the running time would be $n(\log n)^{O(m)}$, and thus even for $m = \mathrm{poly}(1/\epsilon)$ strings this running time is $n \operatorname{polylog} n$. As a consequence, we will be able to reconstruct in $\tilde{O}(n)$ time a string z such that $\mathrm{ED}(s, z) \leq \epsilon n$, which in particular implies Conjecture 2.

1.1 Related Work

A systematic study on the trace reconstruction problem has been started since [37, 38, 5]. However, some of its variants appeared even in the early '70s [30]. One of the main objectives here is to reduce the number of traces required, aka the sample complexity. Both the deletion only and the insertion-deletion channels have been considered so far. In the general worst-case version, the problem considers the unknown string s to be any arbitrary string from $\{0,1\}^n$. The very first result by Batu et al. [5] asserts that for small deletion probability (noise parameter) $p \leq \frac{1}{n^{1/2+\epsilon}}$, to reconstruct s considering $O(n \log n)$ samples suffice. A very recent work [11] improved the sample complexity to $\mathrm{poly}(n)$ while allowing a deletion probability $p \leq \frac{1}{n^{1/3+\epsilon}}$. For any constant deletion probability bounded away from 1, the first subexponential (more specifically, $2^{\tilde{O}(\sqrt{n})}$) sample complexity was shown by [29], which was later improved to $2^{O(n^{1/3})}$ [45, 17], and then finally to $2^{O(n^{1/5})}$ [10].

Another natural variant that has also been widely studied is the average-case, where the unknown string s is randomly chosen from $\{0,1\}^n$. It turns out that this version is significantly simpler than the worst-case in terms of the sample complexity. For sufficiently small noise parameter ($p = o(1)$ as a function of n), efficient trace reconstruction algorithms are known [5, 31, 54]. For any constant noise parameter bounded away from 1 in case of insertion-deletion channel, the current best sample complexity is $\exp(O(\log^{1/3} n))$ [28] improving up on $\exp(O(\log^{1/2} n))$ [49]. Both of these results are built on the worst-case trace reconstruction by [45, 17]. Furthermore, the trace reconstruction algorithm of [28] runs in $n^{1+o(1)}$ time.

In the case of the lower bound, information-theoretically, it is easy to see that $\Omega(\log n)$ samples must be needed when the deletion probability is at least some constant. In the worst-case model, the best known lower bound on the sample complexity is $\tilde{\Omega}(n^{3/2})$ [9]. For the average-case, McGregor, Price, and Vorotnikova [40] showed that $\Omega(\log^2 n)$ samples are necessary to reconstruct the unknown (random) string s. This bound was further improved to $\tilde{\Omega}(\log^{9/4} n)$ by Holden and Lyons [27], and very recently to $\tilde{\Omega}(\log^{5/2} n)$ by Chase [9].

The results described above show an exponential gap between the upper bound and lower bound of the sample complexity. The natural question is, instead of reconstructing the unknown string exactly, if we allow some error in the reconstructed string, then can we reduce the sample complexity? Recently, Davies et al. [16] presented an algorithm that for a specific class of strings (considering various run-lengths or density assumptions), can compute an approximate trace with ϵn additive error under the edit distance while using only $\mathrm{polylog}(n)$ samples. The authors also established that to approximate within the edit distance $n^{1/3-\delta}$, the number of required samples is $n^{1+3\delta/2}/\mathrm{polylog}(n)$, for $0 < \delta < 1/3$, in the worst case. Independently, Grigorescu et al. [24] showed assuming deletion probability $p = 1/2$, there exist two strings within edit distance 4 such that any *mean-based algorithm* requires $\exp(\Omega(\log^2 n))$ samples to distinguish them.

1.2 Technical Overview

The key contribution of this paper is a linear-time approximate trace reconstruction algorithm that uses only three traces to reconstruct an unknown (random) string up to some small edit error (Theorem 1). To get our result, we establish a relation between the (approximate) trace reconstruction problem and the (approximate) median string problem. Consider a uniformly random (unknown) string $s \in \Sigma^n$. We show that for any three traces of s generated by the probabilistic insertion-deletion channel R_p, an arbitrary $(1+\epsilon)$-approximate median of the three trace must be, with high probability, $O(\epsilon)\mathtt{OPT}$-close in edit distance to s (Theorem 3). Once we establish this connection, it suffices to solve the median problem (even approximately). The median of three traces can be solved optimally in $O(n^3)$ time using a standard dynamic programming algorithm [53, 34]. It is not difficult to show that the optimal median objective value $\mathtt{OPT}$ is at least $3(1-O(\epsilon))pn$ (see the full version), and thus the computed median is at edit distance at most $O(\epsilon pn)$ from the unknown string s. This result already beats the known lower bound for *exact* trace reconstruction in terms of sample complexity. However, the running time is cubic in n, whereas the current best average-case trace reconstruction algorithm runs in time $n^{1+o(1)}$ [28].

Next we briefly describe the algorithm that improves the running time to $\tilde{O}(n)$. Instead of finding a median of the entire traces, we compute the median block-by-block and then concatenate the resulting blocks. A natural idea is that each such block is just the median of three substrings taken from the three traces, but the challenge is to identify which substring to take from each trace, particularly because s is not known. To mitigate this issue, we take the first trace s_1 and partition it into disjoint blocks of length $\Theta(\log^2 n)$ each. For each such block, we consider its middle $\log^2 n$-size sub-block as an *anchor*. We then locate for each anchor its corresponding substrings in the other two traces s_2 and s_3, using any approximate pattern matching algorithm under the edit metric (e.g. [36, 22]) to find the *best match* of the anchor inside s_2, s_3. Each anchor has a *true match* in s_2 and in s_3, i.e., the portion that the anchor generated under the noise channel. Since the anchors in s_1 are "well-separated" (by at least $\omega(\log n)$), their true matches are also far apart both in s_2, s_3. Further, exploiting the fact that s is a random string, we can argue that each anchor's best match and true match overlap almost completely, i.e., except for a small portion (see Section 5). We thus treat these best match blocks as anchors in s_2 and s_3 and partition them into blocks. From this point, the algorithm is straightforward. Just consider the first block of each of s_1, s_2, s_3 and compute their median. Then consider the second block from each trace and compute their median, and so on. Finally, concatenate all these block medians, and output the resulting string.

The claim that the best match and true match of an anchor in s_1 are the same except for a small portion, is crucial from two aspects. First, it ensures that any r-th block of s_2, s_3 contains the true match of the r-th anchor of s_1. Consequently, computing a median of these blocks reconstructs the corresponding portion of the unknown string s up to edit distance $O(\epsilon)p\log^2 n$ with high probability. Thus for "most of the blocks", we can reconstruct up to such edit distance bound. We can make the length of the non-anchor portions negligible compared to the anchors (simply because a relatively small "buffer" around each anchor suffices), and thus, we may ignore them and still ensure that the output string is $O(\epsilon pn)$-close (in edit distance) to the unknown string s. (See the proof of Lemma 24 for the details.) The second use of that crucial claim is that it helps in searching for the best match of each anchor "locally" (within a $O(\log^2 n)$-size window) in each s_j, $j \in \{2,3\}$. As a result, we bound the running time of the pattern matching step by $\tilde{O}(n)$. The median computations are also on $\Theta(\log^2 n)$-size blocks, and thus takes total $\tilde{O}(n)$ time.

It remains to explain the key contribution, which is the connection between the (approximate) trace reconstruction and the (approximate) median string problem. Its first ingredient is that there is an "almost unique" alignment between the unknown string s and a trace of it generated by the insertion-deletion channel R_p. Next, we use this to argue about the similarity between an approximate median and the unknown string s. Let us now briefly describe how the uniqueness (or robustness) of the alignment between a random string s and $R_p(s)$ helps us in showing the similarity between s and any approximate median of the traces. Let s_1, s_2, s_3 be three independent traces of s, generated by R_p. We can view s_1 as a uniformly random string, and s_2, s_3 are generated from s_1 by a insertion-deletion channel R_q with a higher noise rate $q \approx 2p$. (See Section 2 for the details.) Hence, any near-optimal alignment between s_1, s_2 and s_1, s_3 "agree" with the planted alignment A^q induced by R_q (denoted by $A^q_{1,2}$ and $A^q_{1,3}$ respectively). Next, we consider the alignment $A_{1,2}$ from s_1 to s_2 via s, that we get by composing the planted alignment from s_1 to s induced by R_p (actually the inverse of the alignment from s to s_1) with the planted alignment from s to s_2 induced by R_p. Similarly, consider the alignment $A_{1,3}$ from s_1 to s_3 via s. Then we take any $(1+\epsilon)$-approximate median x_{med} of $\{s_1, s_2, s_3\}$. Consider an optimal alignment between s_1, x_{med}, and x_{med}, s_2, and x_{med}, s_3. Use these three alignments to define an alignment $M_{1,2}$ from s_1 to s_2 via x_{med}, and an alignment $M_{1,3}$ from s_1 to s_3 via x_{med}. It is not hard to argue that both $A_{1,2}$ and $M_{1,2}$ are near-optimal alignments between s_1, s_2. Thus, both of them agree with the planted alignment $A^q_{1,2}$. Similarly, both $A_{1,3}$ and $M_{1,3}$ agree with the planted alignment $A^q_{1,3}$. Observe, s_2, s_3 are not independently generated from s_1 by R_q. The overlap between $A^q_{1,2}$ and $A^q_{1,3}$ essentially provides an alignment from s_1 to s. Again, using the robustness property of the planted alignment, this overlap between $A^q_{1,2}$ and $A^q_{1,3}$ agrees with the planted alignment from s_1 to s by R_p (actually the inverse of the alignment from s to s_1). On the other hand, since $M_{1,2}$ agrees with $A^q_{1,2}$ and $M_{1,3}$ agrees with $A^q_{1,3}$, there is also a huge agreement between the overlap of $M_{1,2}, M_{1,3}$ and the overlap of $A^q_{1,2}, A^q_{1,3}$. The overlap between $M_{1,2}, M_{1,3}$ is essentially the optimal alignment from s_1 to x_{med} (that we have considered before). This in turn implies that there is a huge agreement between the optimal alignment from s_1 to x_{med} and the planted alignment from s_1 to s by R_p. Hence, we can deduce that x_{med} and s are the same in most of the portions, and thus have small edit distance. We provide the detailed analysis in Section 4.

We have just seen that it suffices to show that a near-optimal alignment between a random string s and $R_p(s)$ is almost unique (or robust). We provide below an overview of this analysis (see Section 3 for details). We start by considering the random string s and a string y generated by passing s through the noise channel R_p. For sake of analysis, we can replace R_p with an *equivalent* probabilistic model G_p, that first computes a random alignment A^p between s and y, and only then fills in random characters in s and in the insertion-positions in y. This model is more convenient because it separates the two sources of randomness, for example we can condition on one (A^p) when analyzing typical behavior of the other (characters of s).

In expectation, the channel G_p generates a trace y by performing about pn random edit operations in s (planting insertions/deletions), hence the planted alignment A^p has expected cost about pn. But can these edit operations cancel each other? Can they otherwise interact, leading to the optimal edit distance being smaller? For example, suppose $s[i] = 0$. If G_p first inserts a 0 before $s[i]$ and then deletes $s[i]$, then clearly these two operations cancel each other. We show that such events are unlikely. Following this intuition, we establish our first claim, that with high probability the edit distance between s, y is large, specifically $\text{ED}(s, y) \geq (1 - 6\epsilon)pn$ for $\epsilon \geq 15p\log(1/p)$, see Lemma 7; thus, the planted alignment A^p is near-optimal. Towards proving this, we first show that a vast majority of the planted edit operations are well-separated, i.e., have $\Theta(1/p)$ positions between them. In this case, for one operation to cancel another one, the characters appearing between them in s must all be equal, which happens with a small probability because s is random.

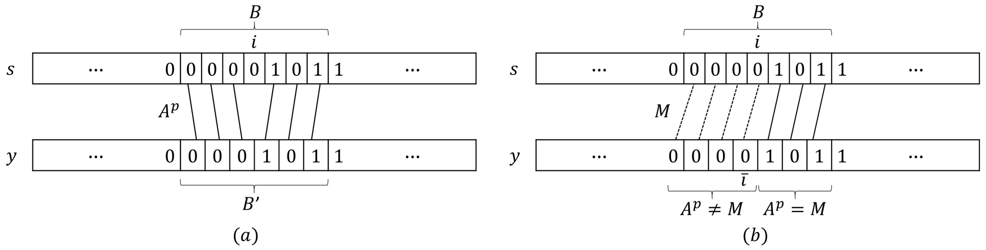


Figure 1 (a) An example of well separated edit operation: A^p deletes $s[i]$ and aligns rest of the characters in block B with block B'. (b) M aligns $s[i]$ and $y[\bar{i}]$. For each index j appearing left of i in B, $A^p[j] \neq M(j)$.

Formally, for almost all indices i where G_p performs some edit operation, the block around it $B = \{i - \frac{c}{r}, i - \frac{c}{r} + 1, \dots, i + \frac{c}{r}\}$ in s (for a small constant $c > 0$), satisfies that i is the only index in B that G_p edits (see Lemma 6). Next we show that in every *optimal* alignment between s and y, almost all these blocks contribute a cost of 1. As otherwise, there is locally an alignment M that aligns each index in B to some character in y, while G_p makes exactly one edit operation, say deletes $s[i]$. See for example Figure 1, where M aligns $s[i]$ and $y[\bar{i}]$ whereas A^p deletes $s[i]$. In this case, M and A^p must disagree on at least c/r indices (all indices either to the right or to the left of i in B). In Figure 1, all $j \in [i - \frac{c}{r}, i]$ satisfy $M[j] \neq A^p[j]$. The crux is that any pair of symbols in s, y are chosen independently at random unless A^p aligns their positions. Thus probability that in each of the $\frac{c}{r}$ pairs aligned by M, the two matched symbols will be equal is $(1/|\Sigma|)^{\frac{c}{r}}$. In the formal proof, we address several technical issues, like having not just one but many blocks, and possible correlations due to overlaps between different pairs, which are overcome by a carefully crafted union bound.

We further need to prove that the planted alignment is robust, in the sense that, with high probability, every near-optimal alignment between s and y must "agree" with the planted alignment A^p on all but a small fraction of the edit operations. Formally, we again consider a partition of s into blocks containing exactly one planted edit operation, and show that for almost all such blocks B, if A^p maps B to a substring B' in y, that near-optimal alignment also maps B to B' (see Lemma 10). To see this, suppose there is a near-optimal alignment that maps B to $\bar{B} \neq B'$. Then following an argument similar to the above, we can show there are many indices in the block B such that A^p and M disagree on them. Thus, in each such block, M tries to match many pairs of symbols that are chosen independently at random, and therefore the probability that M matches B and $\bar{B}$ with a cost at most 1 is small. Compared to Lemma 6, an extra complication here is that now we allow M to match B and $\bar{B}$ with cost at most 1 (and not only 0), and in particular $\bar{B}$ can have three different lengths: $|B|, |B| - 1, |B| + 1$. Hence the analysis must argue separately for all these cases, requiring a few additional ideas/observations.

1.3 Preliminaries

Alignments. For two strings x, y of length n, an *alignment* is a function $A : [n] \to [n] \cup \{\perp\}$ that is monotonically increasing on the *support* of A, defined as $\operatorname{supp}(A) := A^{-1}([n])$, and also satisfies $x[i] = y[A(i)]$ for all $i \in \operatorname{supp}(A)$. An alignment is essentially a common subsequence of x, y, but provides the relevant location information. Define the *length* (or support size) of the alignment as $\operatorname{len}(A) := |\operatorname{supp}(A)|$, i.e., the number of positions in x (equivalently in y)

that are matched by A. Define the *cost of* A to be the number of positions in x and in y that are not matched by A, i.e., $\mathrm{cost}(A) := 2(n - \mathrm{len}(A))$. Let $ED(x, y)$ denotes the minimum cost of an alignment between x, y.

Given a substring $x' = x[i_1, i_2]$ of x, let $\ell_1 := \min\{k \in [i_1, i_2] \mid A(k) \neq \perp\}$ and $\ell_2 := \max\{k \in [i_1, i_2] \mid A(k) \neq \perp\}$, be the first and last positions in the substring x' that are matched by alignment A. If the above is not well-defined, i.e., $A(k) = \perp$ for all $k \in [i_1, i_2]$, then by convention $\ell_1 = \ell_2 = 0$. Let $A(x') := y[A(\ell_1), A(\ell_2)]$ be the *mapping* of x' under A. If $\ell_1 = \ell_2 = 0$, then by convention y' is an empty string. Let $\mathcal{U}_{x'} := \{i_1 \leq k \leq i_2; k \notin \mathrm{supp}(A)\}$ be the positions in x' that are not aligned by A, and similarly let $\mathcal{U}_{y'}$ be the positions in y' not aligned by A. These quantities are related because the number of matched positions in x' is the same as in y', giving us $|x'| - |\mathcal{U}_{x'}| = |y'| - |\mathcal{U}_{y'}|$. Define the *cost of alignment* A *on substring* x' to be

$$\mathrm{cost}_A(x') := |\mathcal{U}_{x'}| + |\mathcal{U}_{y'}|.$$

By abusing the notation, sometimes we will also use $\mathrm{cost}_A([i_1, i_2])$ in place of $\mathrm{cost}_A(x')$. These definitions easily extend to strings of non-equal length, and even of infinite length.

▶ **Lemma 4.** *Given two strings x, y and an alignment A, let $x_1, \dots, x_p$ be disjoint substrings of x. Then*

1. *$A(x_1), \dots, A(x_p)$ are disjoint substrings of y; and*
2. $\mathrm{cost}_A(x) \geq \sum_{i \in [p]} \mathrm{cost}_A(x_i)$

Proof. The first claim directly follows from the fact that $x_1, \dots, x_p$ are disjoint and A is monotonically increasing. Since $x_1, \dots, x_p$ are disjoint, also $\mathcal{U}_{A(x_1)}, \dots, \mathcal{U}_{A(x_p)}$ are disjoint. Similarly, since $A(x_1), \dots, A(x_p)$ are disjoint, also $\mathcal{U}_{x_1}, \dots, \mathcal{U}_{x_p}$ are disjoint. Therefore, $\mathrm{cost}_A(x) \geq \sum_{i \in [p]} (|\mathcal{U}_{x_1}| + |\mathcal{U}_{A(x_1)}|)$. Hence we can claim $\mathrm{cost}_A(x) \geq \sum_{i \in [p]} \mathrm{cost}_A(x_i)$. ◀

For an alignment $A : [n] \to [n] \cup \{\perp\}$, we define the inverse alignment $A^{-1} : [n] \to [n] \cup \{\perp\}$ as follows: For each $j \in [n]$, if $A(i) = j$ for some $i \in [n]$, set $A^{-1}(j) = i$; otherwise, set $A^{-1}(j) = \perp$.

We use the notation $\circ$ for composition of two functions. Composition of two alignments (or inverse of alignments) is defined in a natural way.

Approximate Median. Given a set $S \subseteq \Sigma^*$ and a string $y \in \Sigma^*$, we refer the quantity $\sum_{x \in S} \mathrm{ED}(y, x)$ by the *median objective value* of S with respect to y, denoted by $\mathtt{Obj}(S, y)$.

Given a set $S \subseteq \Sigma^*$, a *median* of S is a string $y^* \in \Sigma^*$ (not necessarily from S) such that $\mathtt{Obj}(S, y^*)$ is minimized, i.e., $y^* = \arg\min_{y \in \Sigma^*} \mathtt{Obj}(S, y)$. We refer $\mathtt{Obj}(S, y^*)$ by $\mathtt{OPT}(S)$. Whenever it will be clear from the context, for brevity we will drop S from both $\mathtt{Obj}(S, y)$ and $\mathtt{OPT}(S)$. We call a string $\tilde{y}$ a *c-approximate median*, for some $c > 0$, of S iff $\mathtt{Obj}(S, \tilde{y}) \leq c \cdot \mathtt{OPT}(S)$.

2 Probabilistic Generative Model

Let us first introduce a probabilistic generative model. For simplicity, our model is defined using infinite-length strings, but our algorithmic analysis will consider only a finite prefix of each string. Fixing a finite alphabet Σ, we denote by $\Sigma^{\mathbb{N}}$ the set of all infinite-length strings over Σ. We write $x \odot y$ to denote the concatenation of two finite-length strings x and y.

We actually describe two probabilistic models that are equivalent. The first model R_p is just the insertion-deletion channel mentioned in Section 1. These models are given an arbitrary string x (base string) to generate a random string y (a trace), but in our intended application x is usually a random string. The second model G_p consists of two stages, first "planting" an alignment between two strings, and only then placing random symbols (accordingly). This is more convenient in the analysis, because we often want to condition on the planted alignment and rely on the randomness in choosing symbols. We provide the formal description of the model R_p and G_p along with a few key properties of them in Appendix A.

3 Robustness of the Insertion-Deletion Channel

In this section we analyze the finite-length version of our probabilistic model G_p (from Section 2), which gets a random string $x \in \Sigma^n$ and generates from it a trace y. We provide a high-probability estimates for the cost of the planted alignment A^p between x and y (Lemma 5), and for the optimal alignment (i.e., edit distance) between the two strings (Lemmas 7 and 8). It follows that with high probability the planted alignment A^p is near-optimal. We then further prove that the planted alignment is robust, in the sense that, with high probability, every near-optimal alignment between the two strings must "agree" with the planted alignment A^p on all but a small fraction of the edit operations (Lemmas 10 and 11).

Assume henceforth that $x \in \Sigma^n$ is a random string x, and given a parameter $p > 0$, generate from it a string y by the random process G_p described in Section 2, denoting by $A^p_{x,y}$ the random mapping used in this process. A small difference here is that now x has finite length, but it can also be viewed as an n-length prefix of an infinite string. Similarly, now y has finite length and is obtained by applying G_p on $x[1,n]$, and it can be viewed also as a finite prefix of an infinite string.

Let $\mathcal{I}^{A^p_{x,y}}$ be the set of indices $i \in [n]$, for which process G_p performs at least one insert/delete operation after (not including) $x[i-1]$ and up to (including) $x[i]$ (i.e., inserting at least one character between $x[i-1], x[i]$, or deleting $x[i]$, or both). This information can clearly be described using $A^p_{x,y}$ alone (independently of x and of the symbols chosen for insertions to y in the second stage of process G_p); we omit the formal definition. When clear from the context, we shorten $\mathcal{I}^{A^p_{x,y}}$ to $\mathcal{I}$.

▶ **Lemma 5.** *For every $i \in [n]$, the probability that $i \in \mathcal{I}$ is*

$$r = r(p) := \sum_{k=1}^{\infty} (2-p)(p/2)^k = p. \tag{2}$$

For all $\epsilon \in [r, 1]$, we have $\Pr[|\mathcal{I}| \notin (1 \pm \epsilon)rn] \leq 2e^{-\epsilon^2 rn/3}$.

Proof. For every $i \in [n]$, the probability that G_p performs $k \geq 1$ edit operations after $x[i-1]$ and up to $x[i]$ is $(p/2)^k + (p/2)^k(1-p) = (p/2)^k(2-p)$, where the first summand represents $k-1$ insertions and one deletion, and the second summand represents k insertions and no deletion. Thus $r = \Pr[i \in \mathcal{I}] = \sum_{k=1}^{\infty}(2-p)(p/2)^k = p$.

For every $i \in [n]$, the probability it appears in $\mathcal{I}$ is r. Then $\mathbb{E}[|\mathcal{I}|] = r \cdot n$. These events are independent, hence by Chernoff's bound, the probability that $|\mathcal{I}| \notin (1 \pm \epsilon)rn$ is at most $2e^{-\epsilon^2 rn/3}$. ◀

As shown in (2), $r := \sum_{k=1}^{\infty}(2-p)(p/2)^k = p$. Hence from now on we replace r by p. Given $\epsilon \in [p, 1]$ and $i \in [n]$, we consider the event $\mathcal{S}_\epsilon(i)$, which informally means that process G_p makes a single "well-spaced" edit operation at position i, i.e., there is an edit operation at position i and no other edit operations within $(\frac{2\epsilon}{p}$ positions away from i. To define it formally, we separate it into two cases, an insertion and a deletion. Observe that these events depend on A^p alone. Let $\mathcal{S}^{del}_\epsilon(i)$ be the event that

1. $A^p(i+1) = A^p(i-1)+1$ (thus $A^p(i) = \perp$); and
2. for all $j \in [2, \frac{2\epsilon}{p}]$, we have $A^p(i+j) = A^p(i+j-1)+1$ and $A^p(i-j) = A^p(i-j+1)-1$ (in particular, they are not $\perp$).

Similarly, let $\mathcal{S}^{ins}_\epsilon(i)$ be the event that

1. $A^p(i) = A^p(i-1)+2$ (thus no index is mapped to $A^p(i-1)+1$);
2. for all $j \in [1, \frac{2\epsilon}{p}]$, we have $A^p(i+j) = A^p(i+j-1)+1$; and
3. for all $j \in [2, \frac{2\epsilon}{p}]$, and $A^p(i-j) = A^p(i-j+1)-1$.

Now define the set of indices for which any of these two events happens

$$\tilde{\mathcal{I}}^{A^p_{x,y}}_\epsilon := \{i \in [n] \mid \text{event } \mathcal{S}_\epsilon(i) := \mathcal{S}^{del}_\epsilon(i) \cup \mathcal{S}^{ins}_\epsilon(i) \text{ occurs}\}.$$

When clear from the context, we shorten $\tilde{\mathcal{I}}^{A^p_{x,y}}_\epsilon$ to $\tilde{\mathcal{I}}$.

▶ **Lemma 6.** *For every $\epsilon \ge p$, we have* $\Pr[|\tilde{\mathcal{I}}| \le (1-5\epsilon)pn] \le e^{-\epsilon^2 p^2 n/2}$.

Proof. For an index $i \in [n]$, define the random variable $X_i \in \{0,1\}$ to be an indicator for the union event $\mathcal{S}^{del}_\epsilon(i) \cup \mathcal{S}^{ins}_\epsilon(i)$. Observe that each of the two events, $\mathcal{S}^{del}_\epsilon(i)$ and $\mathcal{S}^{ins}_\epsilon(i)$, occurs with probability $\frac{p}{2}(1-p)^{4\epsilon/p}/(1-p/2)$, and these events are disjoint. Hence, $\Pr[X_i = 1] = p(1-p)^{4\epsilon/p}/(1-p/2) \ge p(1-p)^{4\epsilon/p} \ge p(1-4\epsilon)$

Next we prove a deviation bound for the random variable $X := \sum_{i\in[n]} X_i = |\tilde{\mathcal{I}}|$, which has expectation is $\mathbb{E}[X] \ge (1-4\epsilon)pn$. Observe that $\{X|X_1,\dots,X_i\}_{i=1}^n$ is a Doob Martingale, and let us apply the method of bounded differences. Revealing X_i (after $X_1,\dots,X_{i-1}$ are already known) might affect the value of X_j's for $j < i + \frac{4\epsilon}{p}$, but by definition their sum is bounded $\sum_{i\le j<i+\frac{4\epsilon}{p}} X_j \le 2$, while the other X_j's are independent of X_i hence the expectation of $\sum_{j\ge i+\frac{2\epsilon}{p}} X_j$ by revealing it. Together, we see that $|\mathbb{E}[X|X_1,\dots,X_i] - \mathbb{E}[X|X_1,\dots,X_{i-1}]| \le 2$, and therefore by Azuma's inequality, $\Pr[X \le (1-5\epsilon)pn] \le \Pr[X \le \mathbb{E}[X] - \epsilon pn] < e^{-2\epsilon^2p^2n^2/(4n)} = e^{-\epsilon^2p^2n/2}$. ◀

Edit Distance (Optimal Alignment) between $\boldsymbol{x}, \boldsymbol{y}$. For each $i \in \tilde{\mathcal{I}}$, define a window $W^i_\epsilon = [i - \frac{\epsilon}{p}, i + \frac{\epsilon}{p}]$.

▶ **Lemma 7.** *For every $\epsilon \in [15p\log\frac{1}{p}, \frac{1}{6}]$, we have* $\Pr[\mathrm{ED}(x,y) < (1-6\epsilon)pn] \le 2e^{-\epsilon^2p^2n/2}$.

At a high level, our proof avoids a direct union bound over all low-cost potential alignments, because there are too many of them. Instead, we introduce a smaller set of basic events that "covers" all these potential alignments, which is equivalent to carefully grouping the potential alignments to get a more "efficient" union bound.

Proof of Lemma 7. We assume henceforth that A^p (the alignment from process G_p) is known and satisfies $|\tilde{\mathcal{I}}| > (1-5\epsilon)pn$, which occurs with high probability by Lemma 6. In other words, we condition on A^p and proceed with a probabilistic analysis based only on the randomness of x and of the characters inserted into y.

Our plan is to define basic events $\mathcal{E}_{S,\bar{S}}$ for every two subsets $S, \bar{S} \subset [n]$ of the same size $\ell = |S| = |\bar{S}|$, representing positions in x and in y, respectively. We will then show that our event of interest is bounded by these events

$$\Big\{\mathrm{ED}(x,y) < (1-6\epsilon)pn\Big\} \subseteq \bigcup_{S,\bar{S}|\ell=\epsilon pn} \mathcal{E}_{S,\bar{S}}, \tag{3}$$

and bound the probability of each basic event by

$$\Pr[\mathcal{E}_{S,\bar{S}}] \le |\Sigma|^{-\epsilon\ell/(3p)}. \tag{4}$$

The proof will then follow easily using a union bound and a simple calculation.

To define the basic event $\mathcal{E}_{S,\bar{S}}$, we need some notation. Write $S = \{i_1, i_2, \ldots, i_\ell\}$ in increasing order, and similarly $\bar{S} = \{\bar{i}_1, \bar{i}_2, \ldots, \bar{i}_\ell\}$, and use these to define ℓ blocks in x and in y, namely, $B_{i_j} = x[i_j - \frac{\epsilon}{p}, i_j + \frac{\epsilon}{p}]$ and $\bar{B}_{i_j} = y[\bar{i}_j - \frac{\epsilon}{p}, \bar{i}_j + \frac{\epsilon}{p}]$. Notice that all the blocks are of the same length $1 + 2\frac{\epsilon}{p}$. Now define $\mathcal{E}_{S,\bar{S}}$ to be the event that (i) $S \subseteq \tilde{\mathcal{I}}$;[2] (ii) the blocks $\bar{B}_{\bar{i}_1}, \ldots, \bar{B}_{\bar{i}_\ell}$ in y are disjoint; and (iii) each block B_{i_j} in x is equal to its corresponding block $\bar{B}_{\bar{i}_j}$ in y. Notice that conditions (i) and (ii) actually depend only on A^p, and thus can be viewed as restrictions on the choice of $S, \bar{S}$ in (3); with this viewpoint in mind, we can simply write

$$\mathcal{E}_{S,\bar{S}} \coloneqq \{B_{i_1} = \bar{B}_{\bar{i}_1}, \ldots, B_{i_\ell} = \bar{B}_{\bar{i}_\ell}\}.$$

We proceed to prove (3). Suppose there is an alignment M from x to y with $\mathrm{cost}(M) < (1 - 6\epsilon)pn$, and consider its cost around each position $i \in \tilde{\mathcal{I}}$, namely, $\mathrm{cost}_M[i - \frac{\epsilon}{p}, i + \frac{\epsilon}{p}]$. These intervals in x are disjoint (by definition of $\tilde{\mathcal{I}}$), and thus by Lemma 4,

$$\sum_{i \in \tilde{\mathcal{I}}} \mathrm{cost}_M(x[i - \tfrac{\epsilon}{p}, i + \tfrac{\epsilon}{p}]) \le \mathrm{cost}(M) < (1 - 6\epsilon)pn.$$

Let $S \subset \tilde{\mathcal{I}}$ include (the indices of) the summands equal to 0. Each other summand contributes at least 1, thus $|\tilde{\mathcal{I}}| - |S| = |\tilde{\mathcal{I}} \setminus S| \cdot 1 < (1 - 6\epsilon)pn$ and by rearranging $|S| > |\tilde{\mathcal{I}}| - (1 - 6\epsilon)pn > \epsilon pn$. To get the exact size $|S| = \epsilon pn$, we can replace S with an arbitrary subset of it of the exact size. Now define $\bar{S} = \{M(i) \mid i \in S\}$. It is easy to verify that the event $\mathcal{E}_{S,\bar{S}}$ holds. Indeed, each $i \in S$ satisfies $\mathrm{cost}_M[i - \frac{\epsilon}{p}, i + \frac{\epsilon}{p}] = 0$, which implies $M(i) \ne \bot$, and thus $|\bar{S}| = |S|$. Moreover, the block $x[i - \frac{\epsilon}{p}, i + \frac{\epsilon}{p}]$ in x is equal to the corresponds block in y, and these blocks in y are disjoint. This completes the proof of (3).

Next, we prove (4). Fix $S, \bar{S} \subset [n]$ of the same size ℓ, and assume requirements (i) and (ii) hold (otherwise, the probability is 0). Let B_{i_j} and $\bar{B}_{\bar{i}_j}$ be the corresponding blocks in x and in y. Consider for now a given $j \in [\ell]$. The requirement $B_{i_j} = \bar{B}_{\bar{i}_j}$ means that for all $t \in \{-\frac{\epsilon}{p}, \ldots, 0, \ldots, +\frac{\epsilon}{p}\}$ we require $x[i_j + t] = y[\bar{i}_j + t]$. The issue is that x and y are random but correlated through A^p; in particular, the symbols $x[i_j + t]$ and $y[\bar{i}_j + t]$ are chosen independently at random unless A^p aligns their positions, i.e., $A^p(i_j + t) = \bar{i}_j + t$. The key observation is that this last event cannot happen for both $t = -1$ and $t = 1$, because in that case, $A^p(i_j + 1) - A^p(i_j - 1) = \bar{i}_j + 1 - (\bar{i}_j - 1) = 2$; however, $i_j \in \tilde{\mathcal{I}}$ implies that A^p has exactly one edit operation (insertion or deletion) in the interval $[i_j - 1, i_j + 1]$ (and not at its endpoints), thus $A^p(i_j + 1) - A^p(i_j - 1) \in \{1, 3\}$. Assume first that $A^p(i_j + t) \ne \bar{i}_j + t$ for $t = 1$. Then the same must hold also for all $t = 2, \ldots, \frac{\epsilon}{p}$; indeed, we again use that $i_j \in \tilde{\mathcal{I}}$, which implies that A^p has no edit operations near position i_j, thus $A^p(i_j + t) = A^p(i_j + 1) + (t - 1) \ne \bar{i}_j + 1 + (t - 1)$. The argument for $t = -1$ is similar, and we conclude that the requirement $B_{i_j} = \bar{B}_{\bar{i}_j}$ encompasses at least $\frac{\epsilon}{p}$ requirements of the form $x[i_j + t] = y[\bar{i}_j + t]$ where these two positions are not aligned by A^p, and thus these two symbols are chosen independently at random.

The above argument applies to every $j \in [\ell]$, yielding overall at least $\ell \cdot \frac{\epsilon}{p}$ requirements of the form $x[i_j + t] = y[\bar{i}_j + t]$, where these two symbols are chosen independently at random. Observe that each $y[\bar{i}_j + t]$ is either a character $x[t']$ (for t' arising from A^p) or completely independent. Since each character of x appears in at most 2 requirements (once on each

[2] This implies that the blocks $B_{i_1}, \ldots, B_{i_\ell}$ in x are disjoint.

side), we can extract a subset of at one-third of the requirements such that the positions in x appearing there are all distinct, and thus the events are independent.[3] We overall obtain at least $\frac{1}{3}\ell \cdot \frac{\epsilon}{p}$ requirements, each occurring independently with probability $1/|\Sigma|$, and thus

$$\Pr[\mathcal{E}_{S,\bar{S}}] \leq |\Sigma|^{-\epsilon\ell/(3p)}.$$

Finally, we are in position to prove the lemma. Combining (3) and (4) and a union bound

$$\begin{aligned}\Pr[\mathrm{ED}(x,y) < (1-6\epsilon)pn] &\leq \binom{n}{\ell}^2 \cdot |\Sigma|^{-\frac{\epsilon\ell}{3p}} \leq \left(\frac{ne}{\ell}\right)^{2\ell} \cdot 2^{-\frac{\epsilon\ell}{3p}} = \left(\frac{e}{\epsilon p}\right)^{2\epsilon pn} \cdot 2^{-\epsilon^2 n/3} \\ &\leq (p^2)^{-2\epsilon pn} \cdot 2^{-\epsilon(15p\log(1/p))n/3} \leq p^{-4\epsilon pn+5\epsilon pn} \leq p^{\epsilon pn}.\end{aligned}$$

Recall that this was all conditioned on A^p, which had error probability at most $e^{-\epsilon^2p^2n/2}$ (by Lemma 6), and now Lemma 7 follows by a union bound. ◀

A similar bound holds for even smaller values of ϵ, provided that the alphabet size is large. The proof is the same, except for the final calculation.

▶ **Lemma 8.** *Suppose* $|\Sigma| \geq (\frac{1}{p})^{15}$. *Then for every* $\epsilon \in [p.\frac{1}{6}]$, *we have* $\Pr[\mathrm{ED}(x,y) < (1-6\epsilon)pn] \leq 2e^{-\epsilon^2p^2n/2}$.

Following an argument similar to the proof of Lemma 7 we can make the following claim.

▶ **Lemma 9.** *Let* $\epsilon \in [15p\log\frac{1}{p}, \frac{1}{6}]$. *Then with probability at least* $1 - 2e^{-\epsilon^2p^2n/2}$, *every alignment* M *between* x, y *satisfies* $|\{i \in \tilde{\mathcal{I}} \mid \mathrm{cost}_M(x[i-\frac{\epsilon}{r}, i+\frac{\epsilon}{r}]) = 0\}| \leq 6\epsilon pn$.

Near-Optimal Alignments between x, y. Given $\epsilon > 0$, a potential alignment M between x, y, and an index $i \in [n]$, define the event

$$\mathcal{E}^M_\epsilon(i) := \begin{cases} A^p(i-\frac{\epsilon}{p}) = \min\{M(k) \neq \bot \mid k \in [i-\frac{\epsilon}{p}, i+\frac{\epsilon}{p}]\}; \text{ and} \\ A^p(i+\frac{\epsilon}{p}) = \max\{M(k) \neq \bot \mid k \in [i-\frac{\epsilon}{p}, i+\frac{\epsilon}{p}]\}. \end{cases} \tag{5}$$

By convention, $\mathcal{E}^M_\epsilon(i)$ is *not* satisfied if the minimization/maximization is over the empty set (because $M(k) = \bot$ for all relevant k). We will only use it for $i \in \tilde{\mathcal{I}}$, in which case both $A^p(i-\frac{\epsilon}{p}), A^p(i+\frac{\epsilon}{p}) \neq \bot$. Intuitively, this event means that A^p and M agree on the block boundaries; for example, in the simpler case where all relevant $M(k) \neq \bot$, this event simply means that $A^p(i-\frac{\epsilon}{p}) = M(i-\frac{\epsilon}{p})$ and $A^p(i+\frac{\epsilon}{p}) = M(i+\frac{\epsilon}{p})$.

Denote the set of indices where the event $\mathcal{E}^M_\epsilon(i)$ occurs and the cost of M over substring $x[i-\frac{\epsilon}{p}, i+\frac{\epsilon}{p}]$ is 1, by

$$\tilde{\mathcal{I}}^{A^p_{x,y}}_{\epsilon,M} := \{i \in \tilde{\mathcal{I}} \mid \text{ event } \mathcal{E}^M_\epsilon(i) \text{ occurs and } \mathrm{cost}_M([i-\tfrac{\epsilon}{p}, i+\tfrac{\epsilon}{p}]) = 1 \}.$$

When clear from the context, we shorten $\tilde{\mathcal{I}}^{A^p_{x,y}}_{\epsilon,M}$ to $\tilde{\mathcal{I}}_M$.

▶ **Lemma 10.** *Let* $\epsilon \in [42p\log\frac{1}{p}, \frac{1}{6}]$ *and* $p \leq \delta \leq \epsilon$. *Then with probability at least* $1 - 4e^{-\frac{\epsilon^2p^2n}{2}}$, *every alignment* M *between* x, y *with* $\mathrm{cost}(M) \leq (1+\delta)pn$ *satisfies* $|\tilde{\mathcal{I}}_M| \geq (1-23\epsilon-\delta)pn$.

[3] To see this, consider an auxiliary graph whose a vertex for each character $x[t]$, and connect two by an edge if they appear in the same constraint. Since every vertex has degree at most 2, a greedy matching contains at least one third of the edges.

At a high level, the proof follows the outline of Lemma 7, and avoids a direct union bound over all (relevant) potential alignments, because there are too many of them. Instead, we introduce a smaller set of basic events that "covers" all these potential alignments. However the analysis is more elaborate with additional cases that require new technical ideas. The proof appears in the full version.

A similar bound holds for even smaller values of ϵ, provided that the alphabet size is large.

▶ **Lemma 11.** *Suppose $|\Sigma| \geq (\frac{1}{p})^{42}$. Then for every $\epsilon \in [p, \frac{1}{6}]$ and every $p \leq \delta \leq \epsilon$, with probability at least $1 - 4e^{-\frac{\epsilon^2 p^2 n}{2}}$, every alignment M between x, y with* $\text{cost}(M) \leq (1+\delta)pn$ *satisfies $|\tilde{\mathcal{I}}_M| \geq (1 - 23\epsilon - \delta)pn$.*

4 Robustness of Approximate Median

In this section, we consider the (approximate) median string problem on a set of strings generated by our probabilistic model G_p (from Section 2). For a random (unknown) string $s \in \Sigma^n$, G_p generates a set $S = \{s_1, s_2, \cdots, s_m\}$ of independent traces of s. We show that with high probability, any $(1+\epsilon)$-approximate median of S must be close (in edit distance) to the unknown string s. In other words, any $(1+\epsilon)$-approximate median must "agree" with the unknown string s in most of the portions. It is true even when $m = 3$. In this section, we state the results and the proofs by considering $m = 3$. In particular, we prove Theorem 3. At the end of the section, we remark on why such result with three traces also directly provides a similar result for any $m > 3$ traces. Another way to interpret this result is the following. Suppose we take a set of three traces and find its $(1+\epsilon)$-approximate median. Then if we add more traces in the set, its $(1+\epsilon)$-approximate median does not change by much. So in some sense, $(1+\epsilon)$-approximate median is robust in the case of average-case traces.

For the purpose of the analysis, we start by considering infinite length strings (as in Section 2), and then later we will move to the finite-length versions. Recall, U denotes the uniform distribution over strings $x \in \Sigma^{\mathbb{N}}$, i.e., each character $x[i]$, for $i \in \mathbb{N}$, is chosen uniformly at random and independently from Σ. Consider a parameter $p \in (0, 0.001)$ and define $q := \frac{p(4-3p)}{2-p^2}$. (Note, $q = 2p - \Theta(p^2)$.) Then consider the following two processes:

- **Process 1**: Draw a string s from U. Then draw three strings s_1, s_2, s_3 independently from $G_p(s)$. Output the tuple (s, s_1, s_2, s_3).
- **Process 2**: Draw a string x_1 from U. Then draw $\bar{x}$ from $G_p(x_1)$ (and denote the corresponding alignment function by $A^p_{1,\bar{x}}$). Finally, draw x_2, x_3 independently from $G_p(\bar{x})$ (and denote the corresponding alignment functions by $A^p_{\bar{x},2}, A^p_{\bar{x},3}$ respectively). Output the tuple $(\bar{x}, x_1, x_2, x_3)$.

As an immediate corollary of Proposition 18 (see Appendix A), we know that the distributions on (s, s_1) and $(\bar{x}, x_1)$ are the same. So we conclude the following about the above two processes.

▷ Claim 12. The probability distributions on (s, s_1, s_2, s_3) and $(\bar{x}, x_1, x_2, x_3)$, the tuples generated by Process 1 and Process 2 respectively, are identical.

Note, we want to investigate the property of an approximate median of the strings generated through Process 1. Due to the above claim, instead of considering the strings s_1, s_2, s_3 from now on we focus on x_1, x_2, x_3 generated through Process 2. By Proposition 16 (see Appendix A), both x_2 and x_3 can be viewed as strings drawn from $G_q(x_1)$. Let us use the notations $A^q_{1,2}$ and $A^q_{1,3}$ to denote the alignment functions produced by the random process G_q while generating x_2 and x_3 respectively, from x_1. We want to emphasize that the process G_q is considered solely for the purpose of the analysis.

Next, we use the alignments $A^p_{1,\bar{x}}$, $A^p_{\bar{x},2}$ (and $A^p_{\bar{x},3}$) to define an alignment between x_1, x_2 (and x_1, x_3) via $\bar{x}$. Let $A^p_{1,\bar{x},2}$ and $A^p_{1,\bar{x},3}$ denote $A^p_{\bar{x},2} \circ A^p_{1,\bar{x}}$ and $A^p_{\bar{x},3} \circ A^p_{1,\bar{x}}$ respectively. (See Section 1.3 for the definition of the notation $\circ$.)

Median of n-length prefixes of x_1, x_2, x_3. So far in this section we have talked about infinite length strings. From now on we restrict ourselves to the the n-length prefixes of x_1, x_2 and x_3 denoted by $x_1[1,n], x_2[1,n]$ and $x_3[1,n]$ respectively. By abusing the notations, we simply use x_1, x_2 and x_3 to also denote $x_1[1,n], x_2[1,n]$ and $x_3[1,n]$ respectively. Also, we consider the (n-length) restriction of all the alignment functions (defined so far) accordingly. Again, for simplicity, we use the same notations to refer to these restricted alignment functions.

Now, we consider the (approximate) median string problem on the set $S = \{x_1, x_2, x_3\}$. Recall, for any string y, $\texttt{Obj}(S, y) := \sum_{k=1}^{3} \mathrm{ED}(x_k, y)$, and $\texttt{OPT}(S) = \min_{y \in \Sigma^*} \texttt{Obj}(S, y)$. Since throughout this section, $S = \{x_1, x_2, x_3\}$, to simplify the notations, we drop S from both $\texttt{Obj}$ and $\texttt{OPT}$. The main result of this section is the following.

▶ **Theorem 13.** *For a large enough $n \in \mathbb{N}$ and a noise parameter $p \in (0, 0.001)$, let $\bar{x}, x_1, x_2$ and x_3 be the n-length prefixes of the strings generated by Process 2. If x_{med} is a $(1+\epsilon)$-approximate median of $S = \{x_1, x_2, x_3\}$ for $\epsilon \in [110p \log(1/p), 1/6]$, then* $\Pr[\mathrm{ED}(\bar{x}, x_{med}) \leq 195\epsilon \cdot \texttt{OPT}(S)] \geq 1 - e^{-\log^2 n}$.

We would like to emphasize that (for the simplicity in the analysis) we have made no attempt to optimize the constants. By a more careful analysis, both the range of p and the constant involved in the bound of $\mathrm{ED}(\bar{x}, x_{\mathrm{med}})$ could be improved significantly. The above theorem together with Claim 12 immediately gives us Theorem 3. Note, in Theorem 3, we do not have any length restrictions on the traces. On the other hand, the above theorem considers $\bar{x}, x_1, x_2$ and x_3 to be of length n. However, by a standard application of Chernoff-Hoeffding bound, it suffices to restrict ourselves to the $(n - \sqrt{n} \log n)$-length prefixes of all the traces (of Theorem 3). Then we can apply the above theorem over them, to get Theorem 3. The proof of Theorem 13 appears in the full version.

For more than three traces. So far, we have shown that for any set $\{s_1, s_2, s_3\}$ of three traces of s, its any $(1+\epsilon)$-approximate median is close to s. Below we argue that a similar result for any arbitrary number (less than some poly(n)) of traces directly follows.

▶ **Corollary 14.** *For a large enough $n \in \mathbb{N}$ and a noise parameter $p \in (0, 0.001)$, let the string $s \in \{0,1\}^n$ be chosen uniformly at random, and let $s_1, \cdots, s_m$ be $m = n^{O(1)}$ traces generated by $G_p(s)$. If x_{med} is a $(1+\epsilon)$-approximate median of $S = \{s_1, \cdots, s_m\}$ for any $\epsilon \in [110p \log(1/p), 1/6]$, then* $\Pr[\mathrm{ED}(s, x_{med}) \leq O(\epsilon) \cdot \frac{\texttt{OPT(S)}}{m}] \geq 1 - n^{-1}$.

We defer the proof to the full version.

5 Near-Linear time Approximate Trace Reconstruction

In this section, we describe a linear-time algorithm that reconstructs the unknown string using only three traces, up to some small edit error. In particular, we prove Theorem 1. Before describing our linear-time algorithm, first note, we can compute an (exact) median of three traces using a standard dynamic programming algorithm [53, 34] in cubic time. Then by Theorem 13, that median string will be close (in edit distance) to the unknown string. More specifically, the edit distance between the computed median string and the unknown string will be at most $O(\epsilon pn)$ with high probability. In this section, we design a more sophisticated method to compute an approximation of the unknown string. For that purpose, we first divide each trace into "well-separated" blocks of size $\log^2 n$ each. Then we run the dynamic programming-based median algorithm [53, 34] on these small blocks. Thus we spend only $\operatorname{poly}\log n$ time per block, and hence in total $\tilde{O}(n)$ time. Since we consider "well-separated" blocks, they are independent. Thus we apply Theorem 13 for each of these blocks (instead of the whole string). Using standard Chernoff-Hoeffding bound, we get that most of these block medians are close to their corresponding block of the unknown string. Hence, by concatenating these block medians, we get back the whole unknown string up to some small edit error. We defer the detailed description of our algorithm to Appendix B.

6 Conclusion

Trace reconstruction in the average case is a well-studied problem. The problem is to reconstruct an unknown (random) string by reading a few traces of it generated via some noise (insertion-deletion) channel. The main objective here is to minimize the sample complexity and also the efficiency of the reconstruction algorithm. There is an exponential gap between the current best upper and lower bound in the sample complexity despite several attempts. The best lower bound is $\tilde{\Omega}(\log^{5/2} n)$ [9]. A natural question is whether it is possible to beat this lower bound by allowing some error in the reconstructed string. This version is also referred to as the approximate trace reconstruction problem; however, nothing is known except for a few special cases.

Our result not only beats the lower bound of the exact trace reconstruction but uses only three traces. The reconstructed string is $O(\epsilon pn)$ close (in edit distance) to the unknown string with high probability. We establish a connection between the approximate trace reconstruction and the approximate median string problem, another utterly significant problem. We show that both the problems are essentially the same. We leverage this connection to design a near-linear time approximate reconstruction algorithm using three traces.

An exciting future direction is to get a similar result for the worst-case, where the unknown string is arbitrary. It will also be fascinating if we could show some non-trivial sample complexity lower bound for that version.

References

1 J. Abreu and Juan Ramón Rico-Juan. A new iterative algorithm for computing a quality approximate median of strings based on edit operations. *Pattern Recognition Letters*, 36:74–80, 2014.

2 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. doi:10.1145/1411509.1411513.

3 Frank Ban, Xi Chen, Adam Freilich, Rocco A Servedio, and Sandip Sinha. Beyond trace reconstruction: Population recovery from the deletion channel. In *60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 745–768. IEEE, 2019.

4 Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, STOC '03, pages 316–324. ACM, 2003.

5 Tugkan Batu, Sampath Kannan, Sanjeev Khanna, and Andrew McGregor. Reconstructing strings from random traces. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004*, pages 910–918. SIAM, 2004.

6 Hervé Cardot, Peggy Cénac, and Antoine Godichon-Baggioni. Online estimation of the geometric median in Hilbert spaces: Nonasymptotic confidence balls. *Annals of Statistics*, 45(2):591–614, 2017. doi:10.1214/16-AOS1460.

7 Francisco Casacuberta and M. D. Antonio. A greedy algorithm for computing approximate median strings. In *Proc. of National Symposium on Pattern Recognition and Image Analysis*, pages 193–198, 1997.

8 Diptarka Chakraborty, Debarati Das, and Robert Krauthgamer. Approximating the median under the ulam metric. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 761–775. SIAM, 2021. doi:10.1137/1.9781611976465.48.

9 Zachary Chase. New lower bounds for trace reconstruction. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 57(2):627–643, 2021.

10 Zachary Chase. Separating words and trace reconstruction. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 21–31, 2021.

11 Xi Chen, Anindya De, Chin Ho Lee, Rocco A. Servedio, and Sandip Sinha. Polynomial-time trace reconstruction in the low deletion rate regime. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021*, volume 185 of *LIPIcs*, pages 20:1–20:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

12 Xi Chen, Anindya De, Chin Ho Lee, Rocco A. Servedio, and Sandip Sinha. Polynomial-time trace reconstruction in the smoothed complexity model. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms*, pages 54–73. SIAM, 2021.

13 Mahdi Cheraghchi, Ryan Gabrys, Olgica Milenkovic, and Joao Ribeiro. Coded trace reconstruction. *IEEE Transactions on Information Theory*, 66(10):6084–6103, 2020.

14 Flavio Chierichetti, Ravi Kumar, Sandeep Pandey, and Sergei Vassilvitskii. Finding the Jaccard median. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 293–311. SIAM, 2010. doi:10.1137/1.9781611973075.25.

15 Michael B. Cohen, Yin Tat Lee, Gary Miller, Jakub Pachocki, and Aaron Sidford. Geometric median in nearly linear time. In *Proceedings of the forty-eighth annual ACM Symposium on Theory of Computing*, pages 9–21, 2016.

16 Sami Davies, Miklós Z. Rácz, Cyrus Rashtchian, and Benjamin G. Schiffer. Approximate trace reconstruction. *CoRR*, abs/2012.06713, 2020. arXiv:2012.06713.

17 Anindya De, Ryan O'Donnell, and Rocco A Servedio. Optimal mean-based algorithms for trace reconstruction. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1047–1056, 2017.

18 Colin de la Higuera and Francisco Casacuberta. Topology of strings: Median string is NP-complete. *Theor. Comput. Sci.*, 230(1-2):39–48, 2000. doi:10.1016/S0304-3975(97)00240-5.

19 Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10*, pages 613–622, 2001. doi:10.1145/371920.372165.

20 Igor Fischer and Andreas Zell. String averages and self-organizing maps for strings. *Proceedings of the neural computation*, pages 208–215, 2000.

21 P. Thomas Fletcher, Suresh Venkatasubramanian, and Sarang Joshi. Robust statistics on riemannian manifolds via the geometric median. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

22 Zvi Galil and Kunsoo Park. An improved algorithm for approximate string matching. *SIAM Journal on Computing*, 19(6):989–999, 1990.

23 Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M. LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77–80, 2013.

24 Elena Grigorescu, Madhu Sudan, and Minshen Zhu. Limitations of mean-based algorithms for trace reconstruction at small distance. *CoRR*, abs/2011.13737, 2020. arXiv:2011.13737.

25 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

26 Morihiro Hayashida and Hitoshi Koyano. Integer linear programming approach to median and center strings for a probability distribution on a set of strings. In *BIOINFORMATICS*, pages 35–41, 2016.

27 Nina Holden and Russell Lyons. Lower bounds for trace reconstruction. *The Annals of Applied Probability*, 30(2):503–525, 2020.

28 Nina Holden, Robin Pemantle, Yuval Peres, and Alex Zhai. Subpolynomial trace reconstruction for random strings and arbitrary deletion probability. *Mathematical Statistics and Learning*, 2(3):275–309, 2020.

29 Thomas Holenstein, Michael Mitzenmacher, Rina Panigrahy, and Udi Wieder. Trace reconstruction with constant deletion probability and related results. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008*, pages 389–398. SIAM, 2008.

30 V. V. Kalashnik. Reconstruction of a word from its fragments. *Computational Mathematics and Computer Science (Vychislitel'naya matematika i vychislitel'naya tekhnika), Kharkov*, 4:56–57, 1973.

31 Sampath Kannan and Andrew McGregor. More on reconstructing strings from random traces: insertions and deletions. In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, pages 297–301. IEEE, 2005.

32 Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 95–103. ACM, 2007. doi:10.1145/1250790.1250806.

33 Teuvo Kohonen. Median strings. *Pattern Recognition Letters*, 3(5):309–313, 1985. doi:10.1016/0167-8655(85)90061-3.

34 Joseph B Kruskal. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM review*, 25(2):201–237, 1983. doi:10.1137/1025045.

35 Ferenc Kruzslicz. Improved greedy algorithm for computing approximate median strings. *Acta Cybernetica*, 14(2):331–339, 1999.

36 Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989.

37 Vladimir I. Levenshtein. Efficient reconstruction of sequences. *IEEE Transactions on Information Theory*, 47(1):2–22, 2001. doi:10.1109/18.904499.

38 Vladimir I. Levenshtein. Efficient reconstruction of sequences from their subsequences or supersequences. *Journal of Combinatorial Theory, Series A*, 93(2):310–332, 2001. doi:10.1006/jcta.2000.3081.

39 Carlos D. Martínez-Hinarejos, Alfons Juan, and Francisco Casacuberta. Use of median string for classification. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 2, pages 903–906. IEEE, 2000. doi:10.1109/ICPR.2000.906220.

40 Andrew McGregor, Eric Price, and Sofya Vorotnikova. Trace reconstruction revisited. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 689–700. Springer, 2014.

41 Stanislav Minsker. Geometric median and robust estimation in Banach spaces. *Bernoulli*, 21(4):2308–2335, 2015.

42 P. Mirabal, J. Abreu, and D. Seco. Assessing the best edit in perturbation-based iterative refinement algorithms to compute the median string. *Pattern Recognition Letters*, 120:104–111, April 2019.

43 Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels. *Probability Surveys*, 6:1–33, 2009.

44 Shyam Narayanan. Population recovery from the deletion channel: Nearly matching trace reconstruction bounds. *arXiv preprint*, 2020. arXiv:2004.06828.

45 Fedor Nazarov and Yuval Peres. Trace reconstruction with $\exp(o(n^{1/3}))$ samples. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1042–1046. ACM, 2017.

46 François Nicolas and Eric Rivals. Complexities of the centre and median string problems. In *14th Annual Symposium on Combinatorial Pattern Matching, CPM 2003*, pages 315–327, 2003.

47 Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarychev, Miklos Z. Racz, Govinda Kamath, Parikshit Gopalan, Bichlien Nguyen, et al. Random access in large-scale dna data storage. *Nature biotechnology*, 36(3):242, 2018.

48 Oscar Pedreira and Nieves R. Brisaboa. Spatial selection of sparse pivots for similarity search in metric spaces. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 434–445. Springer, 2007.

49 Yuval Peres and Alex Zhai. Average-case reconstruction for the deletion channel: Subpolynomially many traces suffice. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 228–239. IEEE Computer Society, 2017.

50 Pavel Pevzner. *Computational molecular biology: an algorithmic approach*. MIT press, 2000.

51 Cyrus Rashtchian, Konstantin Makarychev, Miklós Z. Rácz, Siena Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. Clustering billions of reads for DNA data storage. In *Advances in Neural Information Processing Systems 30*, pages 3360–3371. Curran Associates, Inc., 2017.

52 Richard J Roberts, Mauricio O Carneiro, and Michael C Schatz. The advantages of smrt sequencing. *Genome Biology*, 14(7):405, 2013.

53 David Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, 1975. `doi:10.1137/0128004`.

54 Krishnamurthy Viswanathan and Ram Swaminathan. Improved string reconstruction over insertion-deletion channels. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 399–408, 2008.

55 SM Hossein Tabatabaei Yazdi, Ryan Gabrys, and Olgica Milenkovic. Portable and error-free dna-based data storage. *Scientific reports*, 7(1):1–6, 2017.

A Description of Probabilistic Channel

Model $R_p(x)$. Given an infinite-length string $x \in \Sigma^{\mathbb{N}}$ and a parameter $p \in [0,1]$. Consider the following random procedure:

1. Initialize $i = 1$. (We use i to point to the current index positions of the input string.) Also, initialize an empty string Out.
2. Do the following independently at random:
   a. With probability $1-p$, set $Out \leftarrow Out \odot x[i]$ and increment i. (Match $x[i]$.)
   b. With probability $p/2$, increment i. (Delete $x[i]$.)
   c. With probability $p/2$, choose independently uniformly at random a character $a \in \Sigma$ and set $Out \leftarrow Out \odot a$. (Insert a random character.)

We call this procedure R_p, and denote the randomized output string Out by $R_p(x)$.

Model $G_p(x)$. This model first provides a randomized mapping (alignment) $A^p : \mathbb{N} \to \mathbb{N} \cup \{\perp\}$, and then uses this alignment (and x) to generate the output string. First, given a parameter $p \in [0,1]$, consider the following random procedure to get a mapping A^p:

1. Initialize $i = 1$ and $j = 1$. (Indices of current positions in the input and output strings, respectively.)
2. Do the following independently at random:
   a. With probability $1-p$, set $A^p(i) \leftarrow j$ and increment both i and j. (Match $x[i]$.)
   b. With probability $p/2$, set $A^p(i) \leftarrow \perp$ and increment i. (Delete $x[i]$.)
   c. With probability $p/2$, increment j. (Insertion.)

Next, use this A^p and a given string $x \in \Sigma^{\mathbb{N}}$ to generate a string $y \in \Sigma^{\mathbb{N}}$ as follows. For each $j \in \mathbb{N}$,

- If there is $i \in \mathbb{N}$ with $A^p(i) = j$ then set $y[j] \leftarrow x[i]$. (Match $x[i]$.)
- Otherwise, choose independently uniformly at random a character $a \in \Sigma$ and set $y[j] \leftarrow a$. (Insert a random character.)

We denote by $G_p(x)$ the randomized string y generated as above. By construction, A^p is an alignment between x and $G_p(x)$.

Basic Properties. We claim next that R_p and G_p are equivalent, which is useful because we find it more convenient to analyze G_p. We use $X_1 \overset{\text{dist}}{=} X_2$ to denote that two random variable $X_i \sim D_i$, $i \in \{1, 2\}$ have equal distribution, i.e., $D_1 = D_2$. The next two propositions are immediate.

▶ **Proposition 15.** *For every string $x \in \Sigma^{\mathbb{N}}$ and $p \in [0, 1]$, we have $R_p(x) \overset{\text{dist}}{=} G_p(x)$.*

▶ **Proposition 16** (Transitivity). *For every $x \in \Sigma^{\mathbb{N}}$ and $p \in [0, 1]$, let*

$$q(p) := \frac{p(4 - 3p)}{2 - p^2}. \tag{6}$$

Then $G_p(G_p(x)) \overset{\text{dist}}{=} G_{q(p)}(x)$.

Additional Properties (Random Base String). Let U be the uniform distribution over strings $x \in \Sigma^{\mathbb{N}}$, i.e., each character $x[i]$ is chosen uniformly at random and independently from Σ. We now state two important observations regarding the process G_p. The first one is a direct corollary of Proposition 16. The second observation follows because the probability of insertion is the same as that of deletion at any index in the random process G_p.

▶ **Corollary 17** (Transitivity). *Let $p \in [0, 1]$ and let $q(p)$ be as in (6). Draw a random string $X \sim U$, and use it to draw $Y \sim G_p(G_p(X))$ and $Z \sim G_{q(p)}(X)$. Then $(X, Y) \overset{\text{dist}}{=} (X, Z)$.*

▶ **Proposition 18** (Symmetry). *Let $p \in [0, 1]$. Draw a random string $X \sim U$ and use it to draw another string $Y \sim G_p(X)$. Then $(X, Y) \overset{\text{dist}}{=} (Y, X)$.*

B Near-Linear-Time Median Algorithm

Formally, our result is the following.

▶ **Theorem 19.** *There is a small non-negative constant $c_0 < 1$ and a deterministic algorithm that, for every sufficiently large $n \in \mathbb{N}$ and noise parameter $p \in (0, c_0]$, given as input three traces $s_1, s_2, s_3 \sim G_p(s)$, for a uniformly random (but unknown) string $s \in \{0, 1\}^n$, and an accuracy parameter $\epsilon \in [110p \log(1/p), 1/6]$, outputs in time $\tilde{O}(n)$ a string z that satisfies* $\Pr[\mathrm{ED}(s, z) \leq 5270\epsilon pn] \geq 1 - n^{-1}$.

Before describing the algorithm we would like to introduce a few notations, which we use in this section. For a string $x \in \Sigma^n$, let $y = x[i, i + 1, \cdots, j]$ be a substring of it. Then we use the notation $\texttt{start}(y)$ to denote the index i and $\texttt{end}(y)$ to denote the index j.

Description of the algorithm. Let us now describe the algorithm. First, partition s_1 into $r = \frac{|s_1|}{\ell}$ disjoint blocks $s_1^1, s_1^2, \cdots, s_1^r$ each of length $\ell = \log^2 n + \frac{240}{p}\log^{3/2} n$. For each s_1^i, let us call the middle $\log^2 n$-size sub-block, denoted by y_1^i, an *anchor*. Next, for each $i \in [r]$ and $j \in \{2,3\}$, find the *best match* of the anchor y_1^i in the string s_j i.e., for each y_1^i find a substring (breaking ties arbitrarily) in s_j that has the minimum edit distance with y_1^i. Let us denote the matched substrings in s_2 and s_3 by y_2^i and y_3^i respectively. Then for each $j \in \{2,3\}$, we divide s_j into blocks $s_j^1, \cdots, s_j^r$ (some of the s_j^i's could be empty) by treating $y_j^1, \cdots, y_j^r$ as anchors. More specifically,

- Set the start index of s_j^1 to be 1. For any other non-empty block s_j^i, set its start index to be $\lfloor(\texttt{end}(y_j^{i-1}) + \texttt{start}(y_j^i))/2\rfloor$.
- For the last non-empty block in s_j, set its end index to be $|s_j|$. For any other non-empty block s_j^i, set its end index to be $\lfloor(\texttt{end}(y_j^i) + \texttt{start}(y_j^{i+1}))/2\rfloor - 1$.

Next, for each $i \in [r]$, compute a median of $\{s_1^i, s_2^i, s_3^i\}$, and let it be denoted by z^i. Finally, output $z = z^1 \odot \cdots \odot z^r$ (i.e., the concatenation of all the z^i's).

Correctness proof. Before proceeding with the correctness proof, let us state a known fact about the edit distance between two random strings from [4].

▶ **Proposition 20** ([4]). *For any two strings $x \in \Sigma^m$ and $y \in \Sigma^n$ drawn uniformly at random,* $\Pr[\mathrm{ED}(x,y) \ge \frac{\max\{m,n\}}{10}] \ge 1 - 2^{-\max\{m,n\}/10}$.

▷ Claim 21. For every two substrings x, y of length at least $60\log n$, of s_i, s_j respectively, where $i, j \in [3]$, such that $(A^p_{s,s_i})^{-1}(x)$ and $(A^p_{s,s_j})^{-1}(y)$ are two disjoint substrings of s, $\Pr[\mathrm{ED}(x,y) \ge \frac{\max\{|x|,|y|\}}{10}] \ge 1 - n^{-4}$.

Proof. By Proposition 18, x and y are two strings chosen uniformly at random by picking each of its symbols independently uniformly at random from Σ. Then the claim directly follows from Proposition 20 together with a standard application of union bound. ◁

Let us now define *true match* for each block y_1^i in strings s_2 and s_3. For each $j \in \{2,3\}$, we call the block $A^p_{s,s_j}((A^p_{s,s_1})^{-1}(y_1^i))$ in s_j the true match of y_1^i, denoted by t_j^i. Next, we want to claim that for each block y_1^i, its best match y_j^i in a string s_j, for $j \in \{2,3\}$, is close to its true match t_j^i. The following lemma is crucial to show the correctness of the algorithm and also to establish a linear-time bound for the algorithm.

▷ Claim 22. For each $i \in [r]$ and $j \in \{2,3\}$, with probability $1 - 5n^{-2}$,

1. $|\texttt{start}(t_j^i) - \texttt{start}(y_j^i)| \le \frac{200}{p}\log n$, and
2. $|\texttt{end}(t_j^i) - \texttt{end}(y_j^i)| \le \frac{200}{p}\log n$.

Proof. Let us partition y_1^i into $\frac{p\log n}{10}$ sub-blocks $y_1^{i,1}, \cdots, y_1^{i,(p\log n)/10}$, each of size $\frac{10\log n}{p}$. Next, for each of these sub-blocks $y_1^{i,k}$ consider its true match in the string s_j (for any $j \in \{2,3\}$) defined as $t_j^{i,k} := A^p_{s,s_j}((A^p_{s,s_1})^{-1}(y_1^{i,k}))$.

Now, consider an (arbitrary) optimal alignment B between y_1^i and y_j^i. Observe, if for all $1 \le k \le (p\log n)/10$, $B(y_1^{i,k})$ has a non-empty overlap with the corresponding true match $t_j^{i,k}$, then the claim is true. So from now on, let us assume that at least for some block $y_1^{i,k}$, $B(y_1^{i,k})$ does not overlap with $t_j^{i,k}$. Let $y_1^{i,k'}$ be the right-most sub-block such that there is a non-empty overlapping between $B(y_1^{i,k'})$ and $t_j^{i,k'}$. Then for each $k'+1 \le k \le (p\log n)/10$, by Claim 21, $\mathrm{cost}_B(y_1^{i,k}) \ge \frac{\log n}{p}$ with probability at least $1 - n^{-4}$.

We want to claim that $k' \geq \frac{p \log n}{10} - \frac{10}{1-24p}$. If not, then we deduce that y_j^i is not the best match of y_1^i in the string s_j. To argue this, suppose $k' < \frac{p \log n}{10} - \frac{10}{1-24p}$. Then we modify the mapping B to derive another mapping B' as follows: B' respects B till the block $y_1^{i,k'-1}$. Next, B' deletes the block $y_1^{i,k'}$, and then use the mapping $A^p_{s,s_j} \circ (A^p_{s,s_1})^{-1}$ to map the remaining blocks $y_1^{i,k'+1}, \cdots, y_1^{i,(p\log n)/10}$. By Lemma 5 (applied on strings of size at least $\frac{10 \log n}{p}$) together with an union bound, we get that for all the blocks of s_1 of size at least $\frac{10 \log n}{p}$, the cost of the alignment $A^p_{s,s_j} \circ (A^p_{s,s_1})^{-1}$ is at most $24 \log n$ with probability at least $1 - 2n^{-2}$.

Clearly, the cost of this new alignment B' is at least $(\frac{10}{1-24p}+1)\frac{\log n}{p} - (\frac{10 \log n}{p} + \frac{240 \log n}{1-24p}) > 0$ less than that of B. Hence, y_j^i cannot be the best match of y_1^i in s_j. So we deduce that $k' \geq \frac{p \log n}{10} - \frac{10}{1-24p}$. Note, if an alignment function just deletes all the blocks $y_1^{i,k'+1}, \cdots, y_1^{i,(p\log n)/10}$, it would cost at most $\frac{100 \log n}{p(1-24p)}$. Thus, since t_j^i is the best match of y_1^i, the cost of B for these blocks $y_1^{i,k'+1}, \cdots, y_1^{i,(p\log n)/10}$ must be at most $\frac{100 \log n}{p(1-24p)}$. From this we conclude that $|\mathtt{end}(t_j^i) - \mathtt{end}(y_j^i)| \leq \frac{100}{p(1-24p)} \log n \leq \frac{200}{p} \log n$ (for the choice of p we have).

Similarly, we can argue that $|\mathtt{start}(t_j^i) - \mathtt{start}(y_j^i)| \leq \frac{200}{p} \log n$. This concludes the proof. ◁

The following is an immediate corollary of the above claim.

► **Corollary 23.** *With probability at least $1 - 10n^{-2}$, for each $i \in [r]$ and $j \in \{2,3\}$, y_j^i and y_j^{i+1} do not overlap.*

Proof. Consider the substring between the blocks y_1^i and y_1^{i+1}, which is of length $\frac{480}{p} \log^{3/2} n$. By Lemma 5, $A^p_{s,s_j} \circ (A^p_{s,s_1})^{-1}$ maps that substring into a substring of length at least $240 \log^{3/2} n > \frac{400}{p} \log n$ in s_j with probability at least $1 - n^{-3}$. Now, it directly follows from Claim 22 that y_j^i and y_j^{i+1} do not overlap. ◄

Next, we use the above to establish an upper bound on the edit distance between the unknown string s and the recovered string z.

► **Lemma 24.** *With probability at least $1 - n^{-1}$, $\mathrm{ED}(s,z) \leq 1550\epsilon pn$.*

Proof. For any $i \in [r]$, consider the set $S^i \coloneqq \{s_1^i, s_2^i, s_3^i\}$. Consider the substring y^i of the string s such that $A^p_{s,s_1}(y^i) = y_1^i$ (i.e., y^i maps to y_1^i by the alignment A^p_{s,s_1}). Next, for the analysis purpose, consider the set $T^i = \{y_1^i, t_2^i, t_3^i\}$. Recall, by the definition of $t_j^i = A^p_{s,s_j}(y^i)$, for $j \in \{2,3\}$ are the traces generated by G_p from the block y^i.

Thus by Lemma 5, with probability at least $1 - 3n^{-4}$, for each $t \in T^i$, $\mathrm{ED}(y^i, t) \leq (1+\epsilon)p|y^i|$. Since y_1^i is a substring of s_1^i (where $|s_1^i| = |y_1^i| + \frac{240}{p} \log^{3/2} n$), by triangular inequality, $\mathrm{ED}(y^i, s_1^i) \leq (1+\epsilon)p|y^i| + \frac{240}{p} \log^{3/2} n$. Next observe, for each $j \in \{2,3\}$, by Corollary 23, t_j^i is a substring of s_j^i. Furthermore, by definition, $|s_j^i| \leq |y_j^i| + \frac{240}{p} \log^{3/2} n$. Thus, again by triangular inequality, $\mathrm{ED}(y^i, s_j^i) \leq (1+\epsilon)p|y^i| + \frac{240}{p} \log^{3/2} n$. So we get

$$\mathtt{Obj}(S^i, y^i) \leq 3(1+\epsilon)p|y^i| + \frac{750}{p} \log^{3/2} n. \tag{7}$$

Since z^i is an (exact) median of S^i, $\mathtt{Obj}(S^i, z^i) \leq \mathtt{Obj}(S^i, y^i)$. Next, it follows from Claim 22 and the construction of the blocks s_j^i, for $j \in \{2,3\}$, that t_j^i is a substring of s_j^i where $|t_j^i| \geq |s_j^i| - \frac{500}{p} \log^{3/2} n$. Hence, we can deduce that

$$\begin{aligned}\texttt{Obj}(T^i, z^i) &\le \texttt{Obj}(S^i, z^i) + \frac{1500}{p}\log^{3/2} n\\ &\le \texttt{Obj}(S^i, y^i) + \frac{1500}{p}\log^{3/2} n\\ &\le 3(1+\epsilon)p|y^i| + \frac{2500}{p}\log^{3/2} n \qquad \text{by (7).} \qquad (8)\end{aligned}$$

Further observe, it follows from Proposition 16 and Lemma 7, for each $j \in \{2,3\}$,

$$\mathrm{ED}(y_1^i, t_j^i) \ge (1-6\epsilon)q|y_1^i| = (1-6\epsilon)q\log^2 n.$$

Recall, $q = 2p - \Theta(p^2)$. Then

$$\texttt{OPT}(T^i) \ge 3(1-7\epsilon)p\log^2 n. \qquad (9)$$

From (8) and (9), we conclude that z^i is an $(1+9\epsilon)$-approximate median of T^i. So by Theorem 13, $\mathrm{ED}(y^i, z^i) \le 1755\epsilon \cdot \texttt{OPT}(T^i)$ with probability at least $1 - e^{-2\log^2(\log n)}$.

Now, since all the y^i's are generated by picking each symbol uniformly at random and by our construction for each $j \in \{2,3\}$ t_j^i's are disjoint, the sets T^i's are independent. Hence, by applying standard Chernoff-Hoeffding bound, we get that with probability at least $1 - n^{-1}$, all but at most $e^{-2\log^2(\log n)}r + \sqrt{r\log r}$ many blocks satisfy, $\mathrm{ED}(y^i, z^i) \le 1755\epsilon \cdot \texttt{OPT}(T^i)$.

Let s' denote the string $y^1 \odot \cdots \odot y^r$. Note as s' is a subsequence of s,

$$\mathrm{ED}(s, s') = |s| - |s'| \le \frac{500n}{p}\log^{1/2} n$$

Then,

$$\begin{aligned}\mathrm{ED}(s, z) &\le \mathrm{ED}(s', z) + \frac{500n}{p\log^{1/2} n}\\ &\le \sum_{i=1}^{r}\mathrm{ED}(y^i, z^i) + \frac{500n}{p\log^{1/2} n}\\ &\le 1755\epsilon\sum_{i=1}^{r}\texttt{OPT}(T^i) + (e^{-2\log^2(\log n)}r + \sqrt{r\log r})2\log^2 n + \frac{500n}{p\log^{1/2} n}\\ &\le 5270\epsilon pn.\end{aligned}$$

The first inequality follows by triangular inequality and the last inequality follows by (8) and $r = \Theta(n/\log^2 n)$. ◀

Running time analysis. Partitioning the string s_1 into r blocks clearly takes linear time. The main challenge here is to find the best match y_j^i (for $j \in \{2,3\}$) for each block y_1^i. To do this, for each $j \in \{2,3\}$, we start with the first $10\log^2 n$-sized substring of s_j and run the approximate pattern matching algorithm under the edit metric by [36, 22] to find the best match y_j^1 for y_1^1 (which takes $O(\log^4 n)$ time). Next, we consider the $10\log^2 n$-sized substring of s_j starting from the end index of y_j^1, and in a similar way find the best match y_j^2 for y_1^2. We continue until we find the best matches for all the blocks $y_1^1, \cdots, y_1^r$. Lemma 5 ensures that y_j^1 indeed lies on the first $10\log^2 n$-sized substring of s_j with probability at least $1 - n^{-4}$. Then Corollary 23 together with Lemma 5 guarantees that to find the best match for a block y_1^i, it suffices to look into the $10\log^2 n$-sized substring of s_j after the best match of the previous block y_1^{i-1}. Hence, we can identify the best matches for all the blocks $y_1^1, \cdots, y_1^r$ in time $\tilde{O}(n)$ (since $r = \Theta(\frac{n}{\log^2 n})$). Once we get s_1^i, s_2^i, s_3^i for each $i \in [r]$, we can compute their median using the dynamic programming algorithm [53, 34] in time $O(\log^6 n)$ time. So, the total running time is $\tilde{O}(n)$.

Approximating the Center Ranking Under Ulam

Diptarka Chakraborty
National University of Singapore, Singapore

Kshitij Gajjar
National University of Singapore, Singapore

Agastya Vibhuti Jha[1]
EPFL, Lausanne, Switzerland

Abstract

We study the problem of approximating a *center* under the *Ulam metric*. The Ulam metric, defined over a set of permutations over $[n]$, is the minimum number of move operations (deletion plus insertion) to transform one permutation into another. The Ulam metric is a simpler variant of the general edit distance metric. It provides a measure of dissimilarity over a set of rankings/permutations. In the center problem, given a set of permutations, we are asked to find a permutation (not necessarily from the input set) that minimizes the maximum distance to the input permutations. This problem is also referred to as *maximum rank aggregation under Ulam*. So far, we only know of a folklore 2-approximation algorithm for this NP-hard problem. Even for constantly many permutations, we do not know anything better than an exhaustive search over all $n!$ permutations.

In this paper, we achieve a $\left(\frac{3}{2} - \frac{1}{3m}\right)$-approximation of the Ulam center in time $n^{O(m^2 \ln m)}$, for m input permutations over $[n]$. We therefore get a polynomial time bound while achieving better than a 3/2-approximation for constantly many permutations. This problem is of special interest even for constantly many permutations because under certain dissimilarity measures over rankings, even for four permutations, the problem is NP-hard.

In proving our result, we establish a surprising connection between the approximate Ulam center problem and the *closest string with wildcards* problem (the center problem over the Hamming metric, allowing wildcards). We further study the closest string with wildcards problem and show that there cannot exist any $(2-\epsilon)$-approximation algorithm (for any $\epsilon > 0$) for it unless $\mathsf{P} = \mathsf{NP}$. This inapproximability result is in sharp contrast with the same problem without wildcards, where we know of a PTAS.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Center Problem, Ulam Metric, Edit Distance, Closest String, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.12

Funding This work was supported by NUS ODPRT Grant, WBS No. R-252-000-A94-133.

1 Introduction

Finding a representative of a data set is a classical aggregation task heavily used in data analysis. Given a set S of points in a metric space, one of the more popular versions asks to find a point (not necessarily from S) that minimizes the maximum distance to the points in S, i.e.,

$$\min_{y} \max_{x \in S} d(y, x). \tag{1}$$

Such a point is called a *center*. The question of finding a center in a metric space dates back to the nineteenth century [42]. In several applications, it suffices to compute an *approximate center*, i.e., a point in the metric space that approximates the objective value (1). The problem

[1] This work was done while the author was a student at IIIT-Delhi

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 12; pp. 12:1–12:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of finding an (approximate) center has been studied widely both in theory and practice. Various metric spaces have been considered for the center problem, including Euclidean (both constant [33] and high dimension [7, 43]), Hamming [19, 29, 32, 30], Hamming with wildcard [22], the edit metric [36], Jaccard distance [10], rankings [6, 8, 38], etc. A similar task is to find a *median* point, which asks to minimize the *sum* of the distances to the data points (instead of the maximum distance). The median problem has also been studied extensively in various metric spaces [15, 40, 18, 1, 24, 41, 13, 34]. Despite being similar, finding a center is a much harder task than finding a median. For instance, in the Hamming metric, finding a median is folklore (just take a coordinate-wise majority), whereas finding a center is NP-hard [19].

In this paper, we primarily focus on approximating the center over the *Ulam metric*, which is a close variant of the edit metric. The Ulam metric of dimension n is the metric space $(\mathcal{S}_n, d)$, where $\mathcal{S}_n$ is the set of all permutations over $[n]$ and $d(x, y)$ is the minimum number of character moves needed to transform x into y [2].[2] The importance of studying the Ulam metric is twofold. First, it is an interesting measure of dissimilarity between rankings. The problem of finding a consensus ranking on a set of alternatives based on the preferences of voters arise in many application domains like sports, databases, elections, search engines, and statistics. A common ranking that best captures the preferences among the alternatives is often characterized by the center objective function (1) (a.k.a. *maximum rank aggregation*). The second aspect of the Ulam metric is that it captures some of the inherent difficulties of the edit metric. Thus, any progress in the Ulam metric may provide insights to tackle the more general edit metric which finds numerous applications in computational biology [21, 37], DNA storage system [20, 39], speech recognition [26], and classification [31]. The Ulam metric has also been studied from different algorithmic perspectives [17, 12, 3, 4, 35, 9].

There is a folklore algorithm that finds a 2-approximate center by simply reporting the best input permutation that minimizes the objective (1). This 2-approximation, in fact, holds for every metric space. Unfortunately, so far, we do not know any polynomial-time algorithm that attains better than the folklore 2-approximation, even when the number of input permutations is constant. For the exact computation (or even to beat the 2-factor), nothing better than the exhaustive search (over $n!$ permutations) is known, even for constantly many input permutations. On the hardness side, we only know that it is NP-complete [6]. On the contrary, for the Ulam median problem, very recently, [11] broke below the 2-factor in polynomial time. [11] also provided a polynomial-time 3/2-approximation algorithm for constantly many input permutations. It is not difficult to show that the Ulam center is at least as hard as the Ulam median, even for constantly many inputs (see Appendix A).

Our main result is a deterministic polynomial-time algorithm that breaks below the 3/2-approximation for the Ulam center problem for constantly many inputs.

▶ **Theorem 1.** *There is a deterministic algorithm that, given as input a set of m permutations $S \subseteq \mathcal{S}_n$, computes a $\left(\frac{3}{2} - \frac{1}{3m}\right)$-approximate center of S in time $n^{O(m^2 \ln m)}$.*

The above running time could be improved by increasing the approximation factor slightly. More specifically, for every $\epsilon > 0$, we can compute a $\left(\frac{3}{2} + \epsilon - \frac{1}{m}\right)$-approximate center in time $n^{3m} + n^{O\left(\frac{\ln m}{\epsilon^2}\right)}$ (see Remark 10). It is straightforward to see that when m is constant, the algorithm in the above theorem runs in polynomial time and computes a better than 3/2-approximate center. The question of approximating a center for constantly many ranks/permutations is particularly interesting because even for four inputs, it is known to be NP-complete with respect to *Kendall's tau distance* [18, 8], another often used dissimilarity measure for rankings [23, 44, 45].

[2] One may also consider one deletion and one insertion operation instead of a character move, and define the distance accordingly [17].

Nevertheless, we provide a polynomial-time algorithm to solve the (exact) Ulam center problem for three permutations (Theorem 12). It is worth noting that for Kendall's tau distance, it is unknown whether the center problem is in P or NP-complete for three permutations.

We show our result (Theorem 1) by establishing a surprising connection between the Ulam center and (a generalization of) the *closest string with wildcards* problem. We will explain this connection in the technical overview. In this paper, we further study the closest string with wildcards problem. In the *closest string* problem, given a set of n-length strings over some fixed alphabet Σ, the objective is to find a center (a string from Σ^n) under the Hamming distance. This problem is NP-complete [19], but a PTAS is known [29].

A variant of the closest string problem is the closest string with wildcards. In this variant, each input string may include any number of a wildcard character $*$. The wildcard character $*$ can be matched with all the characters of Σ. For two strings $s, s' \in (\Sigma \cup \{*\})^n$, the Hamming distance between them is defined as $d_H(s, s') := |\{i \in [n] \mid s[i] \neq s'[i] \text{ and } s[i] \neq *, s'[i] \neq *\}|$. Given a set of strings with wildcards, we are asked to find a center string (with no wildcard character) of length n with respect to the Hamming distance. (Note, if wildcards are allowed in the center string, the all-wildcard string will trivially become a center.) This problem is also NP-complete [22]. However, no better than a 2-factor (polynomial-time) algorithm is known. The parameterized complexity of this problem has also been considered [22, 25]. The Hamming distance with wildcard has been studied widely (e.g. [16, 28, 14]) due to its numerous applications in computational biology, large scale web searching, database systems.

As we mentioned earlier, for the simpler variant without any wildcard, there is a PTAS. Can we get a similar PTAS when wildcards are allowed? In this paper, we refute such a possibility. We show that attaining much better than a 2-approximation factor for the closest string with wildcards problem (even for a binary alphabet) is not possible unless $\mathsf{P} = \mathsf{NP}$.

▶ **Theorem 2.** *There is no deterministic polynomial-time $(2 - \epsilon)$-approximation algorithm (for any $\epsilon > 0$) for the closest string with wildcards problem, unless $\mathsf{P} = \mathsf{NP}$.*

The above hardness result holds even for a binary alphabet. The above theorem is in sharp contrast with the (typical) closest string problem for which a PTAS is known [29]. To the best of our knowledge, this is the first $(2 - \epsilon)$-factor inapproximability result for any center problem defined over a set of strings.

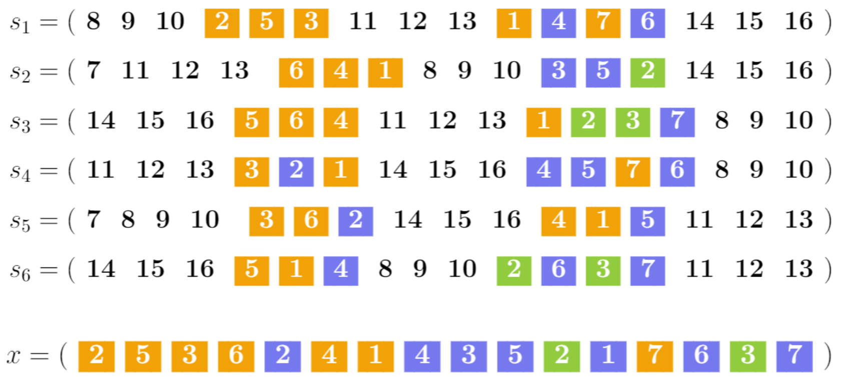
1.1 Technical overview

Approximating the Ulam center. One of our main contributions is a polynomial-time (better than) 3/2-factor approximation algorithm for the Ulam center problem for constantly many permutations. Our algorithm runs in $n^{O(m^2 \ln m)}$ time for m permutations, and achieves $\left(\frac{3}{2} - \frac{1}{3m}\right)$ approximation. For simplicity in exposition, we briefly describe our algorithm that achieves 3/2-approximation by assuming m is a constant. At the very high level, we first compute an exact n-length center (not necessarily a permutation) using dynamic programming and then convert that into a permutation that incurs approximation. The idea is similar to what was used for the Ulam median problem in [11]. However, the similarity ends here. The transformation algorithm that converts an n-length center into a permutation is more intricate, and the analysis is more involved. Another interesting aspect of our algorithm is that we establish a surprising connection between the approximate Ulam center and a generalization of the closest string problem. Let us now briefly explain our algorithm.

If the optimal center objective OPT is small (bounded by a constant), we can find a center permutation by performing an exhaustive search up to a small distance from any input. Thus as long as OPT is at most some constant, in polynomial time, we find an exact center permutation. So, from now, assume that OPT is at least some constant. Our main algorithm has two main steps. The first step constitutes a dynamic programming algorithm that returns an n-length string which maximizes the minimum LCS (Longest Common Substring) with the input permutations (see Subsection 4.1). This algorithm is essentially a generalization of [40, 27]. Let x_n^* be the string we get from this step. Note, the metric defined by $n - |\mathsf{LCS}|$ is essentially the Ulam distance over permutations. So our dynamic programming provides us a center string that minimizes the maximum $n - |\mathsf{LCS}|$. Let us denote this optimum value as OPT_n. Clearly, n-length center string is a relaxation of center permutation. Thus, $\mathsf{OPT}_n \leq \mathsf{OPT}$.

If x_n^* is a permutation, we are done, as we have found an optimal center permutation. Otherwise, we modify x_n^* to get a permutation. Let there be ℓ symbols $a_1, a_2, \ldots, a_\ell$ that appear more than once in x_n^*. For any a_j, each occurrence might be part of a LCS with a subset of input permutations. Now, suppose we delete any one of the occurrences arbitrarily. In that case, we will increase the distances to the corresponding subset of inputs. Consequently, we may end up with a string far from a particular input permutation, causing a much worse objective value than OPT. Thus, we need to delete them in a "balanced" way such that distances to all the inputs increase in "a uniform manner". For that purpose, we introduce a generalization of the closest string problem, which we call *matrix bi-coloring*. We create a matrix having ℓ columns, each corresponding to a repeated symbol. Each row of the matrix corresponds to an input permutation. Thus the number of rows is equal to the number of inputs. Then for a column (corresponding to a_j), we color the entries as follows: If a_j of an input permutation s_i is aligned (with respect to some fixed optimal alignment) with the c-th occurrence of a_j in x_n^*, we color the corresponding entry ((i, j)-th entry) of the matrix by c. Essentially, for each symbol a_j, we have a set of color classes. Each color class c denotes the subset of inputs whose a_j aligns with c-th occurrence of x_n^*. There could be some uncolored entries as well. (See Figure 2 for an example.) Then, we select exactly one color per symbol/column (denoting which occurrence to keep in x_n^*) and cover (alternatively, mark as $\mathfrak{red}$) all the "un-matched" colored entries of that column. Next, we come up with a "coloring scheme" (i.e., a choice of colors per column) such that after covering (marking as $\mathfrak{red}$) the un-matched colored entries, the maximum covered ($\mathfrak{red}$) entries per row is minimized. In general (for an arbitrary colored matrix), there may not exist a coloring scheme leading to a bounded number of covered ($\mathfrak{red}$) entries per row. (This could happen for "tall" matrices, with significantly more rows than columns.) Fortunately, that is not the case for us. Since we have constantly many input permutations and the number of repeated symbols is large (follows from our large OPT assumption), there will always exist a "good" coloring scheme. If we keep the occurrences of repeated symbols in x_n^* according to an optimal coloring scheme, we end up not increasing the center objective value (maximum distances) by much. It is possible to find an optimal coloring scheme using another dynamic programming algorithm (Appendix B). Once we delete all the repeated occurrences, we insert the missing symbols into x_n^*, again in a balanced manner. In the end, we are left with a permutation z over $\mathcal{S}_n$.

The main point of removing repeated entries and the insertion of missing symbols in a balanced manner is to keep the distances to each input within a $3/2$-factor of the initial distance. We argue that, in the end, the distance between an input permutation and the final permutation z will be at most $3/2$ times the initial (maximum) distance to x_n^*. Hence, the center objective value of the output z is at most $\frac{3}{2}\mathsf{OPT}_n \leq \frac{3}{2}\mathsf{OPT}$. We refer the reader to Section 4 for the detailed analysis.



$s_1 = (\ 8\ 9\ 10\ \boxed{2}\ \boxed{5}\ \boxed{3}\ 11\ 12\ 13\ \boxed{1}\ \boxed{4}\ \boxed{7}\ \boxed{6}\ 14\ 15\ 16\ )$

$s_2 = (\ 7\ 11\ 12\ 13\ \boxed{6}\ \boxed{4}\ \boxed{1}\ 8\ 9\ 10\ \boxed{3}\ \boxed{5}\ \boxed{2}\ 14\ 15\ 16\ )$

$s_3 = (\ 14\ 15\ 16\ \boxed{5}\ \boxed{6}\ \boxed{4}\ 11\ 12\ 13\ \boxed{1}\ \boxed{2}\ \boxed{3}\ \boxed{7}\ 8\ 9\ 10\ )$

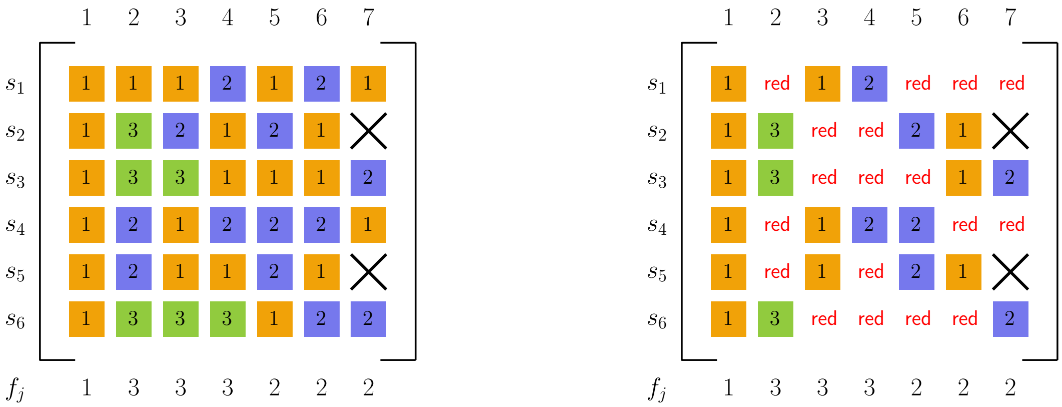
$s_4 = (\ 11\ 12\ 13\ \boxed{3}\ \boxed{2}\ \boxed{1}\ 14\ 15\ 16\ \boxed{4}\ \boxed{5}\ \boxed{7}\ \boxed{6}\ 8\ 9\ 10\ )$

$s_5 = (\ 7\ 8\ 9\ 10\ \boxed{3}\ \boxed{6}\ \boxed{2}\ 14\ 15\ 16\ \boxed{4}\ \boxed{1}\ \boxed{5}\ 11\ 12\ 13\ )$

$s_6 = (\ 14\ 15\ 16\ \boxed{5}\ \boxed{1}\ \boxed{4}\ 8\ 9\ 10\ \boxed{2}\ \boxed{6}\ \boxed{3}\ \boxed{7}\ 11\ 12\ 13\ )$

$x = (\ \boxed{2}\ \boxed{5}\ \boxed{3}\ \boxed{6}\ \boxed{2}\ \boxed{4}\ \boxed{1}\ \boxed{4}\ \boxed{3}\ \boxed{5}\ \boxed{2}\ \boxed{1}\ \boxed{7}\ \boxed{6}\ \boxed{3}\ \boxed{7}\ )$

Figure 1 Let $n = 16$ and $m = 6$. Each of $s_1, s_2, \ldots, s_6$ is a permutation in $\mathcal{S}_{16}$. x is a string (not a permutation) of length 16 over the alphabet [16]. Different occurrences of the same symbol are colored differently in x. The colored entries of s_i also denote an *alignment* with x (in this example, $\mathsf{LCS}(x, s_i)$). Note that $\min_{i\in[6]} |\mathsf{LCS}(x, s_i)| = 6$. Figure 2 shows the colored matrix for this example.



$$\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
s_1 & 1 & 1 & 1 & 2 & 1 & 2 & 1 \\
s_2 & 1 & 3 & 2 & 1 & 2 & 1 & \times \\
s_3 & 1 & 3 & 3 & 1 & 1 & 1 & 2 \\
s_4 & 1 & 2 & 1 & 2 & 2 & 2 & 1 \\
s_5 & 1 & 2 & 1 & 1 & 2 & 1 & \times \\
s_6 & 1 & 3 & 3 & 3 & 1 & 2 & 2 \\
f_j & 1 & 3 & 3 & 3 & 2 & 2 & 2
\end{array}
\qquad
\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
s_1 & 1 & \text{red} & 1 & 2 & \text{red} & \text{red} & \text{red} \\
s_2 & 1 & 3 & \text{red} & \text{red} & 2 & 1 & \times \\
s_3 & 1 & 3 & \text{red} & \text{red} & \text{red} & 1 & 2 \\
s_4 & 1 & \text{red} & 1 & 2 & 2 & \text{red} & \text{red} \\
s_5 & 1 & \text{red} & 1 & \text{red} & 2 & 1 & \times \\
s_6 & 1 & 3 & \text{red} & \text{red} & \text{red} & \text{red} & 2 \\
f_j & 1 & 3 & 3 & 3 & 2 & 2 & 2
\end{array}$$

Figure 2 (Left) A colored matrix M, corresponding to the example in Figure 1 (here, the number in each cell simply denotes the color of the cell). (Right) A matrix bi-coloring scheme $\mathcal{A}$ of M, where $\mathcal{A} = (1, 3, 1, 2, 2, 1, 2)$. As the maximum number of $\mathfrak{red}$ entries in a row is 4, we have $\mathsf{MRI}(\mathcal{A}(M)) = 4$.

Inapproximability of the closest string with wildcards. The matrix bi-coloring problem mentioned earlier is a generalization of the closest string with wildcards problem for a fixed alphabet Σ. To see this, restrict the number of colors per column for the matrix bi-coloring to the alphabet size. (In the matrix bi-coloring, the number of colors per column could be as large as the number of rows, and also, the number of colors in two different columns could be different.) In a simpler version where wildcards are not allowed, we know of a PTAS [29]. So it is quite natural to ask whether we can get a better than 2-factor approximation (ideally, a PTAS) for the closest string with wildcards problem in polynomial time. In this paper, we refute the possibility of having one, assuming $\mathsf{P} \neq \mathsf{NP}$. We show that there is no polynomial-time $(2 - \epsilon)$-approximation algorithm for this problem.

To show our result, we provide a reduction from a variant of the satisfiability (SAT) problem, namely $(1, k, 2k + 1)$-SAT introduced by Austrin, Guruswami and Håstad [5]. In this problem, for any fixed integer $k \geq 1$, given a $(2k + 1)$-CNF formula F, the objective is to distinguish whether there is a satisfying assignment that satisfies at least k literals per clause or F is unsatisfiable. For every fixed integer $k \geq 1$, this problem was shown to be NP-hard [5].

We provide a simple polynomial-time reduction from $(1,k,2k+1)$-SAT to the problem of approximating closest string with wildcards. For an $0<\epsilon<1$, fix $k=\lceil 1/\epsilon \rceil$. Given an instance ($(2k+1)$-CNF formula) F of $(1,k,2k+1)$-SAT with n variables and m clauses, for each clause we create an n-length binary string with wildcards ($\{0,1\}\cup\{*\}$). Each bit position of these strings corresponds to a variable. In a clause, if a variable appears as a positive literal, we set the corresponding bit position of the corresponding string to be 1; and if it appears as a negative literal, we set the corresponding bit position of the corresponding string to be 0. If a variable does not appear in a particular clause, we set the corresponding bit position to be a wildcard ($*$).

Thus in the reduced instance, each string contains at most $2k+1$ non-wildcard entries. It is quite straightforward to see that for a YES instance of the $(1,k,2k+1)$-SAT formula, there is a satisfying assignment that leads to a center string with objective value at most $k+1$. On the other hand, any center string with an objective value $<2k+1$ gives a satisfying assignment. (See Section 5 for the details.)

2 Preliminaries

Notations. Let $[n]$ denote the set $\{1,2,\dots,n\}$. We refer to the set of all permutations over $[n]$ by $\mathcal{S}_n$. Throughout this paper we consider any permutation x as a sequence of numbers $(a_1,a_2,\dots,a_n)$ such that $x(i)=a_i$.

The Ulam metric and the problem of finding a center. Given two permutations $x,y\in\mathcal{S}_n$, the *Ulam distance* between them, denoted by $d(x,y)$, is the minimum number of character move operations[3] that is needed to transform x into y. Alternatively, it can be defined as $n-|\mathsf{LCS}(x,y)|$, where $\mathsf{LCS}(x,y)$ denotes a *longest common subsequence* between x and y.

Given two strings (permutations) x and y of lengths n_x and n_y respectively, an *alignment* g is a function from $[n_x]$ to $[n_y]\cup\{\perp\}$ which satisfies:

- $\forall i\in[n_x]$, if $g(i)\neq\perp$, then $x(i)=y(g(i))$;
- Let $i\in[n_x], j\in[n_x]$ such that $i\neq j$, $g(i)\neq\perp$ and $g(j)\neq\perp$. Then $i<j \Leftrightarrow g(i)<g(j)$.

For an alignment g between two strings (permutations) x and y, we say g *aligns* a character $x(i)$ with some character $y(j)$ if and only if $j=g(i)$. Thus the alignment g is essentially a common subsequence between x and y (see Figure 1 for an example).

Given a set $S\subseteq\mathcal{S}_n$ and another permutation $y\in\mathcal{S}_n$, we refer to the quantity $\max_{x\in S} d(y,x)$ by the *center objective value* of S with respect to y, denoted by $\mathtt{Obj}(S,y)$.

Given a set $S\subseteq\mathcal{S}_n$, a *center* of S is a permutation $x_{\text{cen}}\in\mathcal{S}_n$ (not necessarily from S) such that $\mathtt{Obj}(S,x_{\text{cen}})$ is minimized, i.e., $x_{\text{cen}}=\arg\min_{y\in\mathcal{S}_n}\mathtt{Obj}(S,y)$. We denote $\mathtt{Obj}(S,x_{\text{cen}})$ by $\mathsf{OPT}(S)$. We call a permutation $\tilde{x}$ a *c-approximate center* (for some $c>0$) of S if and only if $\mathtt{Obj}(S,\tilde{x})\leq\mathsf{OPT}(S)\leq c\cdot\mathtt{Obj}(S,\tilde{x})$.

3 Matrix Bi-coloring

In this section, we introduce a problem called the *matrix bi-coloring* problem. Let us start by defining a colored matrix. We use positive integers to identify a color (except a special color $\mathfrak{red}$). An $m\times\ell$ dimensional matrix M is said to be *colored* if each entry of each column

[3] A single move operation in a permutation can be thought of as "picking up" a character from its position and then "inserting" that character in a different position.

$j \in [\ell]$ is either assigned a color from the set $[f_j]$ (for some positive integer f_j), or no-color (which is to say it is uncolored). (Multiple entries in the same column may have the same color.) In other words, the number of *distinct* colors (other than no-color) that can be seen in column j is f_j. See Figure 2 for a visual depiction of matrix bi-coloring.

Given a colored matrix, our goal is to pick exactly one color c_j for each column $j \in [\ell]$ and recolor the matrix. All entries in column j which are colored c_j retain their color. We recolor the remaining entries (except the no-color entries) of column j to red. This is called a *matrix bi-coloring scheme.* We now define this formally.

▶ **Definition 3** (Matrix Bi-coloring Scheme)**.** *Given an $m \times \ell$ colored matrix M, a* matrix bi-coloring scheme *$\mathcal{A}$ for M is an ℓ-tuple $(c_1, c_2, \ldots, c_\ell) \in [f_1] \times [f_2] \times \cdots \times [f_\ell]$.*

The ℓ-tuple $(c_1, c_2, \ldots, c_\ell)$ produced by the matrix bi-coloring scheme $\mathcal{A}$ is used to recolor the matrix M to produce a final $m \times \ell$ colored matrix $\mathcal{A}(M)$, computed as follows. For every $(i, j) \in [m] \times [\ell]$,

- if $M[i][j] = \mathfrak{no\text{-}color}$, then $\mathcal{A}(M)[i][j] = \mathfrak{no\text{-}color}$;
- else, if $M[i][j] = c_j$, then $\mathcal{A}(M)[i][j] = c_j$;
- else, $\mathcal{A}(M)[i][j] = \mathfrak{red}$.

For each row $i \in [m]$ of $\mathcal{A}(M)$, the *red index* of the row, denoted as $\mathsf{RI}_{\mathcal{A},M}(i)$, is the number of red entries in the i-th row of $\mathcal{A}(M)$. (We will drop the subscript $\mathcal{A}, M$ when they are clear from the context.) The *maximum red index* of $\mathcal{A}(M)$ is defined as

$$\mathsf{MRI}(\mathcal{A}(M)) := \max_{i \in [m]} \mathsf{RI}(i).$$

Next, consider the following optimization problem.

▶ **Definition 4** (Matrix Bi-coloring Problem)**.** *Given an $m \times \ell$ colored matrix M, find a matrix bi-coloring scheme $\mathcal{A}$ of M with the minimum MRI. This minimum MRI is called the* bi-coloring number *of the matrix, denoted by BCN. Formally,*

$$\mathsf{BCN}(M) = \min_{\substack{\mathcal{A}:\ \mathcal{A} \text{ is a matrix bi-} \\ \text{coloring scheme for } M}} \mathsf{MRI}(\mathcal{A}(M)).$$

Thus, designing a matrix bi-coloring scheme essentially means coming up with a color for each column. We would like to emphasize that the above problem is a generalization of a certain variant of the center problem under the Hamming metric known as *closest string with wildcards* [22] (see Section 5).

We first show an upper bound on $\mathsf{BCN}(M)$. Then we provide a dynamic programming algorithm that, given a colored matrix M, finds $\mathsf{BCN}(M)$ exactly.

3.1 An upper bound on the bi-coloring number

We show that for every colored matrix M, there always exists a bi-coloring scheme $\mathcal{A}$ such that $\mathsf{MRI}(\mathcal{A}(M))$ is not "too large".

▶ **Theorem 5.** *Let M be an $m \times \ell$ colored matrix such that $m^4 \leq e^\mu$, where*

$$\mu = \sum_{j \in [\ell]} \left(1 - \frac{1}{f_j}\right).$$

Recall that f_j is the number of distinct colors in column $j \in [\ell]$ (not counting no-color*). Then*

$$\mathsf{BCN}(M) \leq \mu + 2\sqrt{\mu \ln m}.$$

An interesting fact about the above theorem is that μ depends only on the *number* of different colors in each column, regardless of how those colors are placed in M. We prove this theorem using the probabilistic method.

Proof of Theorem 5. Given a colored matrix M, we randomly pick an ℓ-tuple $(c_1, c_2, \ldots, c_\ell)$, by selecting each c_j independently uniformly at random from $[f_j]$. This leads to a random bi-coloring scheme $\mathcal{A}$. We then show that the expected MRI of $\mathcal{A}(M)$ is at most μ. Then by a simple Chernoff bound, we conclude that there exists a choice of the ℓ-tuple for which the MRI is at most $\mu + 2\sqrt{\mu \ln m}$, proving Theorem 5.

More precisely, for each column $j \in [\ell]$, we pick a color c_j independently uniformly at random from $[f_j]$. So,

$$\Pr_{c_j \sim [f_j]}[c_j = c] = \frac{1}{f_j} \qquad\qquad \forall\ j \in [\ell], c \in [f_j]. \tag{2}$$

Recall, in each column $j \in [\ell]$ of $\mathcal{A}(M)$, each entry $\mathcal{A}(M)[i][j]$ is either $\mathfrak{no\text{-}color}$, c_j or $\mathfrak{red}$. For all $i \in [m]$ and $j \in [\ell]$, let $X_{i,j}$ be an indicator random variable denoting whether $\mathcal{A}(M)[i][j] = \mathfrak{red}$ or not, i.e.,

$$X_{i,j} = \begin{cases} 1 & \text{if } \mathcal{A}(M)[i][j] = \mathfrak{red}; \\ 0 & \text{otherwise.} \end{cases}$$

Note, $X_{i,j} = 1$ if and only if $M[i][j] \in [f_j]$ and $c_j \neq M[i][j]$. For all $i \in [m]$, let

$$X_i = \sum_{j \in [\ell]} X_{i,j}.$$

Thus, the random variable X_i denotes the number of $\mathfrak{red}$-entries in row $i \in [m]$. The expected number of $\mathfrak{red}$-entries in row $i \in [m]$ is given by

$$\begin{aligned}
\mathbb{E}[X_i] &= \mathbb{E}\left[\sum_{j \in [\ell]} X_{i,j}\right] = \sum_{j \in [\ell]} \mathbb{E}[X_{i,j}] && \text{(Linearity of expectation)} \\
&= \sum_{j \in [\ell]} \Pr[X_{i,j} = 1] \\
&= \sum_{\substack{j \in [\ell]: \\ M[i][j] \in [f_j]}} \Pr[X_{i,j} = 1] + \sum_{\substack{j \in [\ell]: \\ M[i][j] = \mathfrak{no\text{-}color}}} \Pr[X_{i,j} = 1] \\
&= \sum_{\substack{j \in [\ell]: \\ M[i][j] \in [f_j]}} \Pr[X_{i,j} = 1] + 0 && \scriptstyle{(\mathcal{A}(M)[i][j] = \mathfrak{no\text{-}color} \\ \scriptstyle{\iff M[i][j] = \mathfrak{no\text{-}color})}} \\
&= \sum_{\substack{j \in [\ell]: \\ M[i][j] \in [f_j]}} \Pr\left[c_j \neq M[i][j]\right] && \text{(By definition)} \\
&= \sum_{\substack{j \in [\ell]: \\ M[i][j] \in [f_j]}} \left(1 - \frac{1}{f_j}\right) && \text{(By Equation 2)} \\
&\leq \sum_{j \in [\ell]} \left(1 - \frac{1}{f_j}\right) = \mu.
\end{aligned}$$

Thus for every row $i \in [m]$, the expected number of $\mathfrak{red}$-entries in row i is at most μ. We will now show that the event that all of the rows simultaneously have at most $\mu + 2\sqrt{\mu \ln m}$ many $\mathfrak{red}$-entries occurs with non-zero probability.

Note that for each fixed row $i \in [m]$, the indicator random variables $X_{i,1}, X_{i,2}, \ldots, X_{i,\ell}$ are independent. We set

$$\delta = 2\sqrt{(\ln m)/\mu}.$$

Since we are given that $m^4 \leq e^{\mu}$, taking ln on both sides and square rooting, we get $2\sqrt{(\ln m)/\mu} \leq 1$. Thus $0 < \delta \leq 1$. Then it follows from a standard application Chernoff bound that

$$\Pr[X_i \geq (1+\delta)\cdot\mu] \leq \exp\left(\frac{-\delta^2 \cdot \mu}{3}\right).$$

By a union bound,

$$\Pr[\exists\ i \in [m] : X_i \geq (1+\delta)\cdot\mu] \leq m \cdot \exp\left(\frac{-\delta^2 \cdot \mu}{3}\right).$$

Thus we get,

$$\begin{aligned}\Pr[\forall\ i \in [m] : X_i < (1{+}\delta)\cdot\mu] &\geq 1 - \left(m \cdot \exp\left(\frac{-\delta^2 \cdot \mu}{3}\right)\right)\\ &= 1 - \left(m \cdot \exp\left(\frac{-\left(2\sqrt{(\ln m)/\mu}\right)^2 \cdot \mu}{3}\right)\right) \qquad \text{(Substituting } \delta\text{)}\\ &= 1 - \left(m \cdot \exp\left(\frac{-4\ln m}{3}\right)\right)\\ &= 1 - m^{-1/3} > 0.\end{aligned}$$

Thus, $\Pr[\forall\ i \in [m] : X_i < (1+\delta)\cdot\mu] > 0$. In other words, there is a non-zero probability that the number of $\mathfrak{red}$-entries in every row $i \in [m]$ is at most $(1+\delta)\cdot\mu$. Therefore, there exists a bi-coloring scheme $\mathcal{A}^*$ such that the red index $\mathsf{RI}(i)$ of every row $i \in [m]$ satisfies $\mathsf{RI}(i) \leq (1+\delta)\cdot\mu$. Hence,

$$\mathsf{MRI}(\mathcal{A}^*(M)) = \max_{i\in[m]} \mathsf{RI}(i) \leq (1+\delta)\cdot\mu = \mu + \delta\mu = \mu + 2\sqrt{\mu \ln m}.$$

Since $\mathsf{BCN}(M) \leq \mathsf{MRI}(\mathcal{A}^*(M))$, this completes the proof. ◀

We can also compute an optimal bi-coloring using a dynamic programming algorithm. We defer the algorithm to Appendix B.

▶ **Theorem 6.** *There is a deterministic algorithm* FindBiColoring *that, given an $m \times \ell$ colored matrix M, finds a matrix bi-coloring scheme $\mathcal{A}$ of M with the minimum* MRI*, in $O(m\ell^{m+1})$ time.*

4 Approximation Algorithm for the Ulam Center

In this section, we provide a 3/2-approximation algorithm for the Ulam center problem. In particular, we prove Theorem 1.

We are given a set of permutations $S = \{s_1, s_2, \cdots, s_m\} \subseteq \mathcal{S}_n$ as input. Our algorithm runs two procedures, each producing a permutation (candidate center), and returns the better of the two (that has smaller value). For any positive integer k and a permutation $s \in \mathcal{S}_n$, let us use the notation $\mathcal{B}_k(s)$ to denote the set of all the permutations at distance at most k from s, i.e.,

$$\mathcal{B}_k(s) := \{x \in \mathcal{S}_n \mid d(s, x) \leq k\}.$$

The first procedure BoundedSearch performs an exhaustive search up to distance k, for $k = 8m^2 \ln m$. More specifically, it considers an input permutation, say s_1, and enumerates over all $x \in \mathcal{B}_k(s_1)$, and finally returns a permutation $x \in \mathcal{B}_k(s_1)$ that minimizes $\max_{s_i \in S} d(x, s_i)$. Note, $|\mathcal{B}_k(s_1)| = O(n^{2k})$. Thus the running time of this procedure is $O(mn^{2k+1} \ln n)$. (Computing the Ulam distance between two permutations in $\mathcal{S}_n$ takes $O(n \ln n)$ time.) Clearly, if the optimum center objective $\mathsf{OPT}(S) \leq k$, the procedure BoundedSearch outputs an optimum center. So from now, we assume

$$\mathsf{OPT}(S) \geq 8m^2 \ln m. \tag{3}$$

The second procedure, referred to as ApproxCenter, has two main steps. Firstly, it computes the best center string (not necessarily a permutation) x_n^* of length at most n using a procedure FindStringCenter. And secondly, it converts x_n^* to a permutation $s^* \in \mathcal{S}_n$ using a procedure StringToPermutation.

We will show that assuming (3), s^* output by ApproxCenter is a 3/2-approximate center of S. Below we first describe each of the two main steps of ApproxCenter in detail.

4.1 Finding a length-restricted center string

This subsection provides a dynamic programming algorithm that given any set of n length strings computes a center string of length n. More specifically, we design an algorithm that computes a string (over the alphabet $[n]$) of length n, which maximizes the minimum longest common subsequence (LCS) with the input strings. We defer the algorithm to Appendix C.

▶ **Theorem 7.** *There is a deterministic algorithm* FindStringCenter *that, given m strings $s_1, s_2, \ldots, s_m$, each of length n, computes a string $x_n^* = \arg\max_{x \in [n]^n} (\min_i |\mathsf{LCS}(x, s_i)|)$, also of length n, in $O(n^{2m+1} 2^m)$ time.*

Now let us apply this algorithm to our problem. Recall that we are given a set of permutations $S = \{s_1, s_2, \ldots, s_m\} \subseteq \mathcal{S}_n$. We apply the procedure FindStringCenter on the input set S to get an n-length string x_n^*. Note that x_n^* need not be a permutation. In the next subsection, we describe how to transform x_n^* into a permutation.

4.2 Converting a length-restricted center string to a permutation

Let x_n^* be the n-length string obtained in the previous subsection. If x_n^* is a permutation, then we are done, as we have an *exact* solution to the Ulam center problem. Otherwise, let $\mathcal{R} = \{a_1, a_2, \ldots, a_\ell\}$ be the set of "repeated symbols", i.e., the symbols that appear at least twice in x_n^*. For each $a_j \in \mathcal{R}$, let freq_j denote the number of occurrences of a_j in x_n^*. Also, let $\mathcal{M}$ be the set of "missing symbols", i.e., the symbols that do not appear in x_n^*. To transform x_n^* into a permutation, we need to remove duplicate occurrences of the repeated symbols ($\mathcal{R}$), and insert all the missing symbols ($\mathcal{M}$).

Our transformation procedure StringToPermutation consists of following two steps (the pseudocodes for these can be found in Appendix D):

1. Use a procedure RemoveDuplicate (Algorithm 1) to remove all the duplicate occurrences of the symbols in $\mathcal{R}$. RemoveDuplicate first computes an (arbitrary) optimal alignment α_i between x_n^* and s_i, for each $i \in [m]$. Next, construct a colored matrix M of dimension $m \times \ell$ as follows: For each $i \in [m]$ and $j \in [\ell]$, if α_i aligns the r-th occurrence (for some $r \in [\mathsf{freq}_j]$) of the symbol a_j in x_n^*, set $M[i][j] = r$; else set $M[i][j] = \texttt{no-color}$. (As s_i is a permutation, at most one occurrence of a_j in x_n^* can be aligned with the a_j in s_i.)
   Then we use this colored matrix M as an instance of the matrix bi-coloring problem (defined in Section 3) and find an optimum bi-coloring scheme $\mathcal{A}$ for M (using Theorem 6). (For an illustration, see Figure 1 and Figure 2.) Let the scheme $\mathcal{A}$ be the tuple $(c_1, \dots, c_\ell)$. Then, for each symbol $a_j \in \mathcal{R}$, we keep the c_j-th occurrence of it in x_n^* and delete all the remaining occurrences of it. Let $\bar{x}$ denote the output string. (Note, no symbol in $\bar{x}$ appears more than once, and therefore the length of $\bar{x}$ might be less than n.)
2. Use a procedure InsertMissing (Algorithm 2) to insert all the symbols in $\mathcal{M}$ in $\bar{x}$ in a "balanced" manner. Compute an (arbitrary) optimal alignment β_i between $\bar{x}$ and s_i, for each $i \in [m]$. (Note, β_i's can easily be obtained by updating the α_i's computed before.) Consider s_1 and a symbol $b \in \mathcal{M}$. Suppose $s_1[p] = b$, for $p \in [n]$. Let $q \in [n]$ be the largest index $< p$ such that $s_1[q] = a$ is aligned by β_1.
   Then place b just after a in $\bar{x}$, and also update β_1 (by aligning the symbol b). Then remove b from the set $\mathcal{M}$. Next, consider s_2 and another symbol from $\mathcal{M}$, and insert that symbol in $\bar{x}$ in a similar way. Loop through the input permutations one by one in a cyclic manner (after s_m, again take s_1) and perform the above process of inserting symbols in $\mathcal{M}$, until there is no symbol left in $\mathcal{M}$. Let us denote the final transformed string $\bar{x}$ by z.

It is straightforward to see that the final output string z is a permutation in $\mathcal{S}_n$. We claim that z is a 3/2-approximate center. Recall, we only need to argue for $\mathsf{OPT}(S) \geq 8m^2 \ln m$ (by Assumption 3).

▶ **Lemma 8.** *Assuming 3, the final string z output by the procedure* StringToPermutation *is a $\left(\frac{3}{2} - \frac{1}{3m}\right)$-approximate center of S.*

Before commencing the proof of Lemma 8, we need a simple observation on the size of $\mathcal{M}$, the set of missing symbols. Since x_n^* is of length n, the number of missing symbols is equal to the number of repeated occurrences of the symbols in $\mathcal{R}$. More specifically,

$$|\mathcal{M}| = \sum_{a_j \in \mathcal{R}} (\mathsf{freq}_j - 1). \tag{4}$$

Let $\mu := \sum_{j=1}^{\ell}(1 - 1/\mathsf{freq}_j)$. Note that

$$\begin{aligned} \mu = \sum_{j=1}^{\ell}\big((\mathsf{freq}_j - 1)/\mathsf{freq}_j\big) &\leq \frac{1}{2}\sum_j(\mathsf{freq}_j - 1) && (\text{Since } \min_j \mathsf{freq}_j \geq 2) \\ &\leq |\mathcal{M}|/2. && (\text{By Equation 4}) \end{aligned} \tag{5}$$

▷ Claim 9. If $\mu \geq 4 \ln m$, then for all $i \in [m]$,

$$|\mathsf{LCS}(\bar{x}, s_i)| \geq |\mathsf{LCS}(x_n^*, s_i)| - \frac{|\mathcal{M}|}{2} - \sqrt{2|\mathcal{M}| \ln m}.$$

Proof. Recall, $\mathcal{A}$ is an optimum matrix bi-coloring scheme for M (constructed by the procedure RemoveDuplicate). $\bar{x}$ is obtained from x_n^* by removing all repeated occurrences of the symbols in $\mathcal{R}$ according to the tuple $(c_1, \dots, c_\ell)$, corresponding to $\mathcal{A}$. Note, $\mu \geq 4 \ln m$ implies $m^4 \leq e^\mu$. By Theorem 5, $\mathsf{MRI}(\mathcal{A}(M)) \leq \mu + 2\sqrt{\mu \ln m}$, where $\mu = \sum_{j=1}^{\ell}(1 - 1/\mathsf{freq}_j)$.

Consider any $s_i \in S$. Note, M was constructed using the alignment α_i between s_i and x_n^*. Observe, the number of symbols (initially) aligned by α_i that are deleted from x_n^* to obtain $\bar{x}$ is at most $\mathsf{RI}(i)$ (see Section 3 for the definition of RI). Thus, we get

$$\begin{aligned} |\mathsf{LCS}(\bar{x}, s_i)| &\geq |\mathsf{LCS}(x_n^*, s_i)| - (\mu + 2\sqrt{\mu \ln m}) \\ &\geq |\mathsf{LCS}(x_n^*, s_i)| - \frac{|\mathcal{M}|}{2} - \sqrt{2|\mathcal{M}| \ln m}. && \text{(By Equation 5)} \end{aligned}$$

This completes the proof of Claim 9. ◁

Proof of Lemma 8. We will argue that for all $s_i \in S$, $d(z, s_i) \leq \left(\frac{3}{2} - \frac{1}{3m}\right) \mathsf{OPT}(S)$. Let s^* be an optimum center of S under the Ulam metric. So, for all $s_i \in S$, $d(s^*, s_i) \leq \mathsf{OPT}(S)$. Recall, by definition, $d(s^*, s_i) = n - |\mathsf{LCS}(s^*, s_i)|$. Thus,

$$\forall s_i \in S, \quad |\mathsf{LCS}(s^*, s_i)| \geq n - \mathsf{OPT}(S). \tag{6}$$

Since x_n^* maximizes the minimum LCS between an n-length string and $s_i \in S$, by (6),

$$\forall s_i \in S, \quad |\mathsf{LCS}(x_n^*, s_i)| \geq n - \mathsf{OPT}(S). \tag{7}$$

Also, observe that

$$\begin{aligned} |\mathcal{M}| &\leq n - \min_{s_i \in S} (|\mathsf{LCS}(x_n^*, s_i)|) \\ &\leq \mathsf{OPT}(S). && (\min_{s_i \in S} (|\mathsf{LCS}(x_n^*, s_i)|) \geq n - \mathsf{OPT}(S) \text{ by Equation 7}) \end{aligned} \tag{8}$$

Consider an $s_i \in S$. By the procedure InsertMissing, among the inserted symbols at least $\lfloor |\mathcal{M}|/m \rfloor$ symbols will be aligned between the final string z and s_i by the alignment function β_i. In particular,

$$|\mathsf{LCS}(z, s_i)| \geq |\mathsf{LCS}(\bar{x}, s_i)| + \left\lfloor \frac{|\mathcal{M}|}{m} \right\rfloor. \tag{9}$$

Next, we proceed by considering the two cases depending on the value of μ separately. Let us first argue for $\mu < 4 \ln m$. In fact, in this case, we get a solution that is much closer to the optimum. More specifically, we claim that $d(z, s_i) \leq (1 + 1/m^2)\mathsf{OPT}(S)$. As $\bar{x}$ is obtained from x_n^* by deleting repeated occurrences of the symbols in $\mathcal{R}$ and $s_i \in \mathcal{S}_n$,

$$|\mathsf{LCS}(\bar{x}, s_i)| \geq |\mathsf{LCS}(x_n^*, s_i)| - \ell. \qquad \text{(Recall, } \ell = |\mathcal{R}|) \tag{10}$$

Note, the above inequality holds irrespective of the value of μ.

Since $\mathsf{freq}_j \geq 2$ for all $j \in [\ell]$, we have $\mu = \sum_{j=1}^{\ell} (1 - 1/\mathsf{freq}_j) \geq \ell/2$. This implies that

$$\ell \leq 2\mu < 8 \ln m \leq \frac{\mathsf{OPT}(S)}{m^2} \tag{11}$$

where the last inequality follows from Assumption (3). Thus,

$$\begin{aligned} d(z, s_i) &= n - |\mathsf{LCS}(z, s_i)| && \text{(By definition)} \\ &\leq n - |\mathsf{LCS}(\bar{x}, s_i)| && \text{(By Equation 9)} \\ &\leq n - |\mathsf{LCS}(x_n^*, s_i)| + \ell && \text{(By Equation 10)} \\ &\leq \mathsf{OPT}(S) + \frac{\mathsf{OPT}(S)}{m^2} && \text{(By Equation 7, 11)} \\ &\leq \left(1 + \frac{1}{m^2}\right) \mathsf{OPT}(S). \end{aligned}$$

So, for $\mu < 4\ln m$, we have $d(z, s_i) \leq \left(1 + \frac{1}{m^2}\right) \mathsf{OPT}(S)$, which is at most $\left(\frac{3}{2} - \frac{1}{3m}\right) \mathsf{OPT}(S)$ for $m \geq 2$.

Now, the only remaining case is $\mu \geq 4\ln m$. We will use an argument similar to the previous case. The only difference is that now to lower bound $|\mathsf{LCS}(\bar{x}, s_i)|$, we will apply Claim 9 (instead of Equation 10).

$$\begin{aligned}|\mathsf{LCS}(z, s_i)| &\geq |\mathsf{LCS}(\bar{x}, s_i)| + \left\lfloor \frac{|\mathcal{M}|}{m} \right\rfloor && \text{(By Equation 9)}\\ &\geq |\mathsf{LCS}(x_n^*, s_i)| - \frac{|\mathcal{M}|}{2} - \sqrt{2|\mathcal{M}|\ln m} + \left\lfloor \frac{|\mathcal{M}|}{m} \right\rfloor && \text{(By Claim 9)}\\ &\geq |\mathsf{LCS}(x_n^*, s_i)| - |\mathcal{M}|\left(\frac{1}{2} - \frac{1}{m}\right) - \sqrt{2|\mathcal{M}|\ln m} - 1. && (12)\end{aligned}$$

Hence,

$$\begin{aligned}d(z, s_i) &= n - |\mathsf{LCS}(z, s_i)| && \text{(By definition)}\\ &\leq n - |\mathsf{LCS}(x_n^*, s_i)| + |\mathcal{M}|\left(\frac{1}{2} - \frac{1}{m}\right) + \sqrt{2|\mathcal{M}|\ln m} + 1 && \text{(By Equation 12)}\\ &\leq \mathsf{OPT}(S) + \left(\frac{1}{2} - \frac{1}{m} + \sqrt{\frac{2\ln m}{\mathsf{OPT}(S)}}\right) \mathsf{OPT}(S) + 1 && \text{(By Equation 7, 8)}\\ &\leq \left(\frac{3}{2} - \frac{1}{m} + \sqrt{\frac{2\ln m}{8m^2 \ln m}} + \frac{1}{\mathsf{OPT}(S)}\right) \mathsf{OPT}(S) && \text{(By Assumption (3))}\\ &\leq \left(\frac{3}{2} - \frac{1}{3m}\right) \mathsf{OPT}(S). && \text{(By Assumption (3))}\end{aligned}$$

This completes the proof of Lemma 8. ◀

Running time analysis. We now analyze the running time of the overall algorithm. The procedure BoundedSearch takes $O(mn^{2k+1}\ln n)$ time (to perform an exhaustive search up to distance k from an input permutation). Next, we analyze the running time of ApproxCenter. The first step of it uses Theorem 7 taking $O(2^m n^{2m+1})$ time. The second step consists of two procedures: RemoveDuplicate and InsertMissing. Constructing the colored matrix M in the procedure RemoveDuplicate takes $O(mn\ln n)$ time. (Note, it only involves m LCS computations. In each of them, one of the strings is a permutation and thus takes $O(n\ln n)$ time per string by using a standard LCS algorithm.) Next, RemoveDuplicate invokes the algorithm from Theorem 6 to find an optimal bi-coloring scheme, which requires $O(mn^{m+1})$ time (the number of missing symbols is at most n). Once we get the bi-coloring scheme, generating $\bar{x}$ takes only $O(n)$ time. The next procedure, InsertMissing, again involves m LCS computations and then updating those alignments according to the insertion of missing symbols. This takes $O(mn\ln n)$ time. So the overall running time is $O(mn^{2k+1}\ln n + 2^m n^{2m+1} + mn^{m+1}) = n^{O(m^2 \ln m)}$ (by replacing $k = 8m^2 \ln m$).

▶ Remark 10. Let us now comment on how to reduce the running time by increasing the approximation factor slightly. Recall, after performing the exhaustive search BoundedSearch up to distance k, we remain with the case when $\mathsf{OPT}(S) \geq k$. When we analyze the approximation factor of our algorithm (in particular, Lemma 8), to attain $\left(\frac{3}{2} - \frac{1}{3m}\right)$ we need to assume that $\mathsf{OPT}(S)$ is at least $\Omega(m^2 \ln m)$. That is why we set $k = 8m^2 \ln m$. Our analysis essentially shows that the approximation factor is $\max\left\{1 + \frac{8\ln m}{k}, \frac{3}{2} - \frac{1}{m} + \sqrt{\frac{2\ln m}{k}} + \frac{1}{k}\right\}$ with the running time $O(mn^{2k+1}\ln n + 2^m n^{2m+1})$. So we get a trade-off between the approximation factor and the running time. For instance, if we set $k = 8m$, we get an approximation factor $\left(\frac{3}{2} + \sqrt{\frac{\ln m}{4m}} - \frac{7}{8m}\right)$ and running time $n^{O(m)}$. In fact, for any $0 < \epsilon < 1$, by setting $k = \frac{8\ln m}{\epsilon^2}$, we get a $\left(\frac{3}{2} + \epsilon - \frac{1}{m}\right)$-approximate center in $O(2^m n^{2m+1}) + n^{O\left(\frac{\ln m}{\epsilon^2}\right)}$ time.

▶ **Theorem 11.** *There is a deterministic algorithm that, given an $0 < \epsilon < 1$ and a set of m permutations $S \subseteq \mathcal{S}_n$, computes a $\left(\frac{3}{2} + \epsilon - \frac{1}{m}\right)$-approximate center of S in time $n^{3m} + n^{O\left(\frac{\ln m}{\epsilon^2}\right)}$.*

4.3 An exact algorithm for three permutations

When the number of input permutations is only three (i.e., $m = 3$), we can get an exact polynomial-time algorithm for the Ulam center problem.

▶ **Theorem 12.** *There is a deterministic polynomial-time algorithm that takes 3 permutations as input, and outputs their Ulam center.*

We will show that given three permutations s_1, s_2, s_3 from $\mathcal{S}_n$, it is possible to remove duplicate symbols and insert missing symbols in a simple and efficient way that converts an n-length center string for s_1, s_2, s_3 to an optimal center permutation for them.

If the x_n^* computed by our dynamic program is already a permutation, then we are done. Otherwise, first fix some optimal alignment between x_n^* and s_i for all $i \in [3]$. We incrementally update the sting x_n^* by processing the repeated symbols one by one. Take an arbitrary repeated symbol a in x_n^* and a missing symbol b. Note, a appears more than once. Consider the first occurrence of a in x_n^*, and do the following:

- If (that particular occurrence of) a does not align with any of s_1, s_2, s_3, then we delete it from x_n^*, and insert b at an arbitrary location in x_n^*. Remove b from $\mathcal{M}$. Clearly, this does not decrease $\mathsf{LCS}(x_n^*, s_i)$ for any of the three s_i's.
- If (that particular occurrence of) a aligns with the symbol a of only one input permutation (say s_1), then delete it from x_n^*, and insert b in x_n^* so that it aligns with the b in s_3. Remove b from $\mathcal{M}$. The length of $\mathsf{LCS}(x_n^*, s_1)$ is decreased by one for by the deletion of a, but it is then increased by one by the insertion of b. So $|\mathsf{LCS}(x_n^*, s_1)|$ remains the same.
- If (that particular occurrence of) a aligns with the symbol a in two of the permutations (say s_1, s_2), we do nothing.

Only except the last case, each time we process a repeated symbol a, we remove an occurrence of it from x_n^*. (Note that the number of times we need to process a symbol a is equal to its number of occurrences in x_n^*, minus one.) Since the number of input permutations is exactly three, at most one occurrence of a can be aligned with that of two input permutations. So for any repeated symbol in x_n^*, at most once we will be in the last case.

Once we are left with no repeated symbols, we stop. (Since x_n^* is always of length n, there will not be any missing symbols left after we are done with processing all the repeated symbols.) So, x_n^* will eventually become a permutation, and $|\mathsf{LCS}(x_n^*, s_i)|$ for none of the three s_i's will decrease. Let us denote the final string by z. By Equation 7, for each $i \in [3]$,

$$n - \mathsf{OPT}(S) \leq |\mathsf{LCS}(x_n^*, s_i)| = |\mathsf{LCS}(z, s_i)|.$$

Thus, $d(z, s_i) \leq \mathsf{OPT}(S)$. Hence, we conclude that z is an (exact) Ulam center for s_1, s_2, s_3. The running time of the algorithm is clearly polynomial in n, concluding Theorem 12.

5 Closest String with Wildcards

The matrix bi-coloring problem defined in Section 3 is a generalization of the well-known *closest string with wildcards* problem. In this problem, any given string may include wildcard characters which can be matched with any character of the other strings. Consider any alphabet Σ. For any two strings $s, s' \in (\Sigma \cup \{*\})^n$, the Hamming distance between them is defined as

$$d_H(s, s') := |\{i \in [n] \mid s[i] \neq s'[i] \text{ and } s[i] \neq * \text{ and } s'[i] \neq *\}|.$$

In the closest string with wildcards problem, given a set of m strings $s_1, s_2 \ldots, s_m \in (\Sigma \cup \{*\})^n$, the objective is to find a string $s \in \Sigma^n$ such that $\max_{i\in[m]} d_H(s, s_i)$ is minimized.

The above problem is a special case of the matrix bi-coloring problem, where the strings are the rows of the matrix and the wildcards ($*$) are the $\mathfrak{no\text{-}color}$ entries in the matrix. If we restrict $f_j = |\Sigma|$ (for all $j \in [n]$) in the matrix bi-coloring problem, we get the closest string with wildcards problem.

So far we do not know of any polynomial-time algorithm for the closest string with wildcards problem that achieves a $(2 - \epsilon)$-factor approximation (for some $0 < \epsilon < 1$). In this section, we refute the possibility of getting such an algorithm unless $\mathsf{P} = \mathsf{NP}$, even when the alphabet Σ is binary. In particular, we prove Theorem 2.

To show the inapproximability result, we start with defining a variant of the satisfiability (SAT) problem, namely $(1, k, 2k + 1)$-SAT introduced by Austrin, Guruswami & Håstad [5].

▶ **Definition 13** ($(1, k, 2k + 1)$-SAT)**.** *Let $k \geq 1$ be a fixed integer constant. Given a $(2k + 1)$-CNF formula F (i.e., each clause of F has exactly $2k+1$ literals), decide between the following two cases:*

- *YES: There is an assignment for the variables in F that satisfies at least k literals in each clause of F.*
- *NO: F is unsatisfiable.*

▶ **Theorem 14** ([5])**.** *For every fixed integer $k \geq 1$, $(1, k, 2k + 1)$-SAT is NP-hard.*

For $k = 1$, $(1, k, 2k+1)$-SAT is simply 3-SAT. We now provide a polynomial-time reduction from $(1, k, 2k + 1)$-SAT to (a gap version of) the closest string with wildcards problem.

▶ **Definition 15** (Approximate closest string with wildcards)**.** *Consider any alphabet Σ and an $\epsilon > 0$. Given a set of m strings $s_1, s_2, \ldots, s_m \in (\Sigma \cup \{*\})^n$ (where $*$ is a wildcard) and a positive integer r, decide between the following two cases.*

- *YES: There is a string $s \in \Sigma^n$ such that for all $i \in [m]$, $d_H(s, s_i) \leq r$.*
- *NO: For all strings $s \in \Sigma^n$, there exists an $i \in [m]$ such that $d_H(s, s_i) > (2 - \epsilon)r$.*

Proof of Theorem 2. Let $k = \lceil 1/\epsilon \rceil$. Consider an instance ($(2k + 1)$-CNF formula) F of the $(1, k, 2k + 1)$-SAT problem with n variables $x_1, x_2, \ldots, x_n$ and m clauses. We create m strings each of length n over the alphabet $\{0, 1\} \cup \{*\}$. For each clause C_i, we create a string s_i as follows: If the literal x_j appears in C_i, set $s_i[j] = 1$; else if the literal $\bar{x}_j$ (negation of x_j) appears in C_i, set $s_i[j] = 0$; else set $s_i[j] = *$. Set $r = k + 1$.

Suppose F is a YES instance of $(1, k, 2k + 1)$-SAT. Take the corresponding satisfying assignment σ (that satisfies at least k literals per clause). Create a string $s \in \{0, 1\}^n$ by setting $s[j] = 1$ if x_j is set to TRUE by σ, and $s[j] = 0$ if x_j is set to FALSE by σ, for all $j \in [n]$. Note that $d_H(s, s_i) \leq (2k + 1) - k = k + 1$ for all $i \in [m]$.

Now, suppose F is a NO instance of $(1, k, 2k+1)$-SAT. Assume to contrary that there exists a string $s \in \{0,1\}^n$ such that for all $i \in [m]$, $d_H(s, s_i) \leq (2-\epsilon)(k+1) < 2k+1$ (since $k \geq 1/\epsilon$). Then create an assignment σ' by setting x_j to TRUE if $s[j] = 1$, and FALSE if $s[j] = 0$, for all $j \in [n]$. Note that σ' satisfies F, contradicting the fact that F is unsatisfiable.

The proof follows from Theorem 14. ◀

6 Conclusion

In this paper, we study the problem of computing a center rank/permutation under the Ulam metric, which is known to be NP-complete. There is a folklore 2-approximation algorithm that works for every metric space. No better (polynomial-time) algorithm is known for the Ulam metric, even when the number of input permutations is constant. Our main result breaks below the 3/2-approximation for constantly many inputs. An exciting open direction is to beat the 2-approximation for arbitrarily many inputs (i.e., an algorithm whose running time is polynomial in both n and m).

In proving our result, we establish a connection between the Ulam center problem and the closest string with wildcards problem (the center problem under the Hamming metric in the presence of wildcards). We further show that the latter problem is $(2-\epsilon)$-inapproximable unless P = NP. This result is in sharp contrast with the PTAS known for the closest string problem without wildcards.

References

1 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. doi:10.1145/1411509.1411513.

2 David Aldous and Persi Diaconis. Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem. *Bulletin of the American Mathematical Society*, 36(4):413–432, 1999. doi:10.1090/S0273-0979-99-00796-X.

3 Alexandr Andoni and Robert Krauthgamer. The computational hardness of estimating edit distance. *SIAM J. Comput.*, 39(6):2398–2429, 2010.

4 Alexandr Andoni and Huy L. Nguyen. Near-optimal sublinear time algorithms for Ulam distance. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 76–86, 2010. doi:10.1137/1.9781611973075.8.

5 Per Austrin, Venkatesan Guruswami, and Johan Håstad. (2+ϵ)-sat is NP-hard. *SIAM J. Comput.*, 46(5):1554–1573, 2017.

6 Christian Bachmaier, Franz J. Brandenburg, Andreas Gleißner, and Andreas Hofmeier. On the hardness of maximum rank aggregation problems. *Journal of Discrete Algorithms*, 31:2–13, 2015. 24th International Workshop on Combinatorial Algorithms (IWOCA 2013).

7 Mihai Bādoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the thiry-fourth annual ACM Symposium on Theory of Computing*, pages 250–257, 2002.

8 Therese Biedl, Franz J Brandenburg, and Xiaotie Deng. On the complexity of crossings in permutations. *Discrete Mathematics*, 309(7):1813–1823, 2009.

9 Mahdi Boroujeni and Saeed Seddighin. Improved MPC algorithms for edit distance and Ulam distance. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019.*, pages 31–40, 2019.

10 Marc Bury and Chris Schwiegelshohn. On finding the jaccard center. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

11 Diptarka Chakraborty, Debarati Das, and Robert Krauthgamer. Approximating the median under the ulam metric. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 761–775. SIAM, 2021.

12 Moses Charikar and Robert Krauthgamer. Embedding the Ulam metric into l_1. *Theory of Computing*, 2(11):207–224, 2006. doi:10.4086/toc.2006.v002a011.

13 Flavio Chierichetti, Ravi Kumar, Sandeep Pandey, and Sergei Vassilvitskii. Finding the Jaccard median. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 293–311. SIAM, 2010. doi:10.1137/1.9781611973075.25.

14 Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A black box for online approximate pattern matching. In *Annual Symposium on Combinatorial Pattern Matching*, pages 143–151. Springer, 2008.

15 Michael B Cohen, Yin Tat Lee, Gary Miller, Jakub Pachocki, and Aaron Sidford. Geometric median in nearly linear time. In *Proceedings of the forty-eighth annual ACM Symposium on Theory of Computing*, pages 9–21, 2016.

16 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings of the thiry-fourth annual ACM Symposium on Theory of Computing*, pages 592–601, 2002.

17 Graham Cormode, Shan Muthukrishnan, and Süleyman Cenk Sahinalp. Permutation editing and matching via embeddings. In *International Colloquium on Automata, Languages, and Programming*, pages 481–492. Springer, 2001. doi:10.1007/3-540-48224-5_40.

18 Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10*, pages 613–622, 2001. doi:10.1145/371920.372165.

19 Moti Frances and Ami Litman. On covering problems of codes. *Theory of Computing Systems*, 30(2):113–119, 1997.

20 Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M. LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77–80, 2013.

21 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

22 Danny Hermelin and Liat Rozenberg. Parameterized complexity analysis for the closest string with wildcards problem. In *Combinatorial Pattern Matching*, pages 140–149, Cham, 2014. Springer International Publishing.

23 John G Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

24 Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the thirty-ninth annual ACM Symposium on Theory of Computing*, pages 95–103, 2007.

25 Tomohiro Koana, Vincent Froese, and Rolf Niedermeier. Parameterized algorithms for matrix completion with radius constraints. In *31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

26 Teuvo Kohonen. Median strings. *Pattern Recognition Letters*, 3(5):309–313, 1985. doi:10.1016/0167-8655(85)90061-3.

27 Joseph B Kruskal. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM review*, 25(2):201–237, 1983. doi:10.1137/1025045.

28 Christina Leslie, Rui Kuang, and Kristin Bennett. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 5(9), 2004.

29 Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *Journal of the ACM (JACM)*, 49(2):157–171, March 2002.

30 Bin Ma and Xiaoming Sun. More efficient algorithms for closest string and substring problems. *SIAM Journal on Computing*, 39(4):1432–1443, 2010.

31 Carlos D. Martínez-Hinarejos, Alfons Juan, and Francisco Casacuberta. Use of median string for classification. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 2, pages 903–906. IEEE, 2000. doi:10.1109/ICPR.2000.906220.

32 Daniel Marx. Closest substring problems with small distances. *SIAM J. Comput.*, 38:1382–1410, 2008.

33 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM (JACM)*, 31(1):114–127, 1984.

34 Stanislav Minsker. Geometric median and robust estimation in banach spaces. *Bernoulli*, 21(4):2308–2335, 2015.

35 Timothy Naumovitz, Michael Saks, and C. Seshadhri. Accurate and nearly optimal sublinear approximations to Ulam distance. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2012–2031, 2017. doi:10.1137/1.9781611974782.131.

36 François Nicolas and Eric Rivals. Complexities of the centre and median string problems. In *Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Morelia, Michocán, Mexico, June 25-27, 2003, Proceedings*, pages 315–327, 2003.

37 Pavel Pevzner. *Computational molecular biology: an algorithmic approach.* MIT press, 2000.

38 V Yu Popov. Multiple genome rearrangement by swaps and by element duplications. *Theoretical computer science*, 385(1-3):115–126, 2007.

39 Cyrus Rashtchian, Konstantin Makarychev, Miklós Z. Rácz, Siena Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. Clustering billions of reads for DNA data storage. In *Advances in Neural Information Processing Systems 30*, pages 3360–3371. Curran Associates, Inc., 2017.

40 David Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, 1975. doi:10.1137/0128004.

41 Warren Schudy. Approximation schemes for inferring rankings and clusterings from pairwise data. *Ph.D. Thesis*, 2012.

42 James Joseph Sylvester. A question in the geometry of situation. *Quarterly Journal of Pure and Applied Mathematics*, 1(1):79–80, 1857.

43 E Alper Yildirim. Two algorithms for the minimum enclosing ball problem. *SIAM Journal on Optimization*, 19(3):1368–1391, 2008.

44 H Peyton Young. Condorcet's theory of voting. *American Political science review*, 82(4):1231–1244, 1988.

45 H Peyton Young and Arthur Levenglick. A consistent extension of condorcet's election principle. *SIAM Journal on Applied Mathematics*, 35(2):285–300, 1978.

A Ulam Median reduces to Ulam Center

The idea of the reduction is the same as [8, Theorem 6]. Here we discuss the idea for four permutations, which can be generalised to m permutations easily. Given a set $P = \{s_1, s_2, s_3, s_4\}$ of 4 permutations on $[n]$, we construct a new set $Q = (z_1, z_2, z_3, z_4)$ of 4 permutations on $[4n]$ by applying s_1, s_2, s_3, s_4 to four partitions of $[4n]$ as follows, such that the Ulam median for P can be obtained from the Ulam center for Q.

- $z_1 = (s_1([1, \dots, n]), s_2([n+1, \dots, 2n]), s_3([2n+1, 3n]), s_4([3n+1, 4n]))$
- $z_2 = (s_2([1, \dots, n]), s_3([n+1, \dots, 2n]), s_4([2n+1, 3n]), s_1([3n+1, 4n]))$
- $z_3 = (s_3([1, \dots, n]), s_4([n+1, \dots, 2n]), s_1([2n+1, 3n]), s_2([3n+1, 4n]))$
- $z_4 = (s_4([1, \dots, n]), s_1([n+1, \dots, 2n]), s_2([2n+1, 3n]), s_3([3n+1, 4n]))$

Let $c_Q \in \mathcal{S}_{4n}$ be an Ulam center of Q. The following are easy to see.

- $c_Q[1, \dots, n]$ does not contain any symbols from $[n+1, \dots, 4n]$;
- $c_Q[n+1, \dots, 2n]$ does not contain any symbols from $[1, \dots, n] \cup [2n+1, \dots, 4n]$;
- $c_Q[2n+1, \dots, 3n]$ does not contain any symbols from $[1, \dots, 2n] \cup [3n+1, \dots, 4n]$;
- $c_Q[3n+1, \dots, 4n]$ does not contain any symbols from $[1, \dots, 3n]$.

It is also easy to see that the four permutations are equidistant from c_Q. That is,

$$d(c_Q, z_1) = d(c_Q, z_2) = d(c_Q, z_3) = d(c_Q, z_4).$$

Finally, these facts are sufficient to claim that $c_Q[1, \ldots, n]$ (or rather any one of the four partitions of $[4n]$) is an Ulam median for P. See [8, Theorem 6] for a comprehensive proof. (Although [8, Theorem 6] talks about the Kendall's tau distance, the argument could easily be extended for the Ulam metric.)

B Computing the Bi-coloring Number of a Colored Matrix

Here, we provide a dynamic programming algorithm that given any colored matrix M of dimension $m \times \ell$, computes $\mathsf{BCN}(M)$ in $O(m\ell^{m+1})$ time. More specifically, we prove Theorem 6.

For a clean description, we provide the dynamic program for $m = 4$, although it works for every positive integer m. We use C to denote our dynamic programming table. The cells of C store a Boolean value if the value is 1. C has 5 dimensions.

1. Subproblem: Let $C[i_1, i_2, i_3, i_4, k]$ denotes whether it is possible to leave at most i_1, i_2, i_3 and i_4 unpicked in rows 1 to 4 respectively by picking colors till column k. If it's not possible, the cell will contain a 0(False value). Otherwise it'll contain a 1(True value) along with the picked color. We denote the number of colors we can leave unpicked per row as the picking requirements for the cell in the dynamic program.
2. Computing C: Consider any column $k \geq 2$ and values $i_1 \geq 1, i_2 \geq 1, i_3 \geq 1$ and $i_4 \geq 1$. Picking any color in a row indicates, that the number of unpicked colors in all other rows increase by 1. Thus, (without loss of generality) M_{1j} could only be a feasibly choice if $C[i_1 - 1, i_2, i_3, i_4, k - 1]$ is true. If no choice of color in column k satisfies this property, then clearly we can't satisfy the picking requirements of the cell.
3. Recurrence:

$$\begin{aligned} C[i_1, i_2, i_3, i_4, k] &= C[i_1 - 1, i_2, i_3, i_4, k - 1] \vee D[i_1, i_2 - 1, i_3, i_4, k - 1] \\ &\quad \vee C[i_1, i_2, i_3 - 1, i_4, k - 1] \vee C[i_1, i_2, i_3, i_4 - 1, k - 1] \\ C[1, 0, 0, 0, 1] &= 1 \\ C[0, 1, 0, 0, 1] &= 1 \\ C[0, 0, 1, 0, 1] &= 1 \\ C[0, 0, 0, 1, 1] &= 1 \end{aligned}$$

4. Order of evaluation: We iterate over k one by one, and then evaluate over the first 4 indices lexicographically. Each cell queries lexicographically smaller cells.
5. Final Answer: Look at all the cells $C[i_1, i_2, i_3, i_4, \ell]$ for all $0 \leq i_1 \leq \ell, 0 \leq i_2 \leq \ell, 0 \leq i_3 \leq \ell, 0 \leq i_4 \leq \ell$ for which the cell contains a 1(True value). For each of these cells, let's denote the maximum of the quantity i_1, i_2, i_3 and i_4 as the i_{max} value for this cell. Output the cell with the minimum i_{max} value.

There are $(\ell + 1)^4 \times \ell$ sub-problems and for each cell, we look at 4 different sub-problems. Generalising to m strings: our dynamic program table has $(\ell + 1)^m \times \ell$ cells, and we look at m different cells to compute the answer for each cell. Thus our dynamic program runs in $m \times (\ell + 1)^m \times \ell$ time.

C Computing a Length-restricted Center String

For simplicity of exposition, we describe the dynamic programming algorithm FindString-Center only for three strings s_1, s_2, s_3. However, it can easily be extended to any number of strings in a natural way. We use D to denote our dynamic programming table. D stores a string and has 7 dimensions.

1. Subproblem: $D[i_1, i_2, i_3, k_1, k_2, k_3, \ell] = x_\ell$, where x_ℓ is an ℓ-length string with the following properties.
   a. $\mathsf{LCS}(s_1[1, 2, \ldots, i_1], x_\ell) \geq k_1$
   b. $\mathsf{LCS}(s_2[1, 2, \ldots, i_2], x_\ell) \geq k_2$
   c. $\mathsf{LCS}(s_3[1, 2, \ldots, i_3], x_\ell) \geq k_3$
   If such a string does not exist, then $D[i_1, i_2, i_3, k_1, k_2, k_3, \ell] = \emptyset$.
2. Computing D: Consider the substrings $s_1[1, 2, \ldots, i_1], s_2[1, 2, \ldots, i_2], s_3[1, 2, \ldots, i_3]$. The cases when there exists a string of length at most ℓ satisfying the above three conditions are listed below.
   a. At least one of the following is true.
      (i) $D[i_1, i_2, i_3, k_1, k_2, k_3, \ell - 1] \neq \emptyset$.
      (ii) $D[i_1 - 1, i_2, i_3, k_1 - 1, k_2, k_3, \ell] \neq \emptyset$.
      (iii) $D[i_1, i_2 - 1, i_3, k_1, k_2 - 1, k_3, \ell] \neq \emptyset$.
      (iv) $D[i_1, i_2, i_3 - 1, k_1, k_2, k_3 - 1, \ell] \neq \emptyset$.
      If the first of these four cases is true, then we can extend $x_{\ell-1}$ by putting an arbitrary symbol at the ℓ-th position in x_ℓ. In the other three cases, x_ℓ remains the same.
   b. At least one of the following is true.
      (v) If $D[i_1 - 1, i_2, i_3, k_1 - 1, k_2, k_3, \ell - 1] \neq \emptyset$, then $x_\ell \leftarrow x_{\ell-1} \circ s_1[i_1]$.
      (vi) If $D[i_1, i_2 - 1, i_3, k_1, k_2 - 1, k_3, \ell - 1] \neq \emptyset$, then $x_\ell \leftarrow x_{\ell-1} \circ s_2[i_2]$.
      (vii) If $D[i_1, i_2, i_3 - 1, k_1, k_2, k_3 - 1, \ell - 1] \neq \emptyset$, then $x_\ell \leftarrow x_{\ell-1} \circ s_3[i_3]$.
      (viii) If $D[i_1 - 1, i_2 - 1, i_3, k_1 - 1, k_2 - 1, k_3, \ell - 1] \neq \emptyset$ and $s_1[i_1] = s_2[i_2]$, then $x_\ell \leftarrow x_{\ell-1} \circ s_1[i_1]$.
      (ix) If $D[i_1, i_2 - 1, i_3 - 1, k_1, k_2 - 1, k_3 - 1, \ell - 1] \neq \emptyset$ and $s_2[i_2] = s_3[i_3]$, then $x_\ell \leftarrow x_{\ell-1} \circ s_2[i_2]$.
      (x) If $D[i_1 - 1, i_2, i_3 - 1, k_1 - 1, k_2, k_3 - 1, \ell - 1] \neq \emptyset$ and $s_1[i_1] = s_3[i_3]$, then $x_\ell \leftarrow x_{\ell-1} \circ s_1[i_1]$.
      (xi) If $D[i_1 - 1, i_2 - 1, i_3 - 1, k_1 - 1, k_2 - 1, k_3 - 1, \ell - 1] \neq \emptyset$ and $s_1[i_1] = s_2[i_2] = s_3[i_3]$, then $x_\ell \leftarrow x_{\ell-1} \circ s_1[i_1]$.
   c. None of the above are true.
      (xii) $D[i_1, i_2, i_3, k_1, k_2, k_3, \ell] = \emptyset$.
3. Recurrence: The cell $D[i_1, i_2, i_3, k_1, k_2, \ell]$ looks at all the possible 12 cases described above and does as mentioned in the points. The base case is $D[0, 0, 0, 0, 0, 0, 0] = \varepsilon$ where ε denotes the empty string.
4. Order of evaluation: We initialize by setting the cell $D[0, 0, 0, 0, 0, 0, 0] = \emptyset$ and proceed in lexicographic order. Note that each cell only queries lexicographically smaller cells.
5. Final Answer: Consider only those cells for which $D \neq \emptyset$. Let the string stored in each such cell σ be denoted by x^*_σ. Let $\mathsf{LCS}^{\min}_\sigma$ be one of the three strings $\{\mathsf{LCS}(x^*_\sigma, s_1), \mathsf{LCS}(x^*_\sigma, s_2), \mathsf{LCS}(x^*_\sigma, s_3)\}$, whichever has the minimum length. Compute $\mathsf{LCS}^{\min}_\sigma$ for the cell $D[n, n, n, k_1, k_2, k_3, n]$ for all $0 \leq k_1 \leq n, 0 \leq k_2 \leq n, 0 \leq k_3 \leq n$ (whenever $D[n, n, n, k_1, k_2, k_3, n] \neq \emptyset$). Among all these $\mathsf{LCS}^{\min}_\sigma$ strings, output the longest string as the final answer, denoted by x^*. (Note that x^* might not be of length n.)

If the final string x^* is of length less than n, then we fill in missing symbols from $[n]$ arbitrarily and make x^* an n-length string (denoted by x^*_n). Clearly adding more symbols to x^* cannot decrease $\mathsf{LCS}(s_i, x^*)$ for any of the s_i's.

D Pseudocodes from Section 4.2

Algorithm 1 RemoveDuplicate.

```
Result: Removes duplicate characters from $x_n^*$
Initialise $\alpha_i$ as an arbitrary occurrence of $\mathsf{LCS}(x_n^*, s_i)$ in $s_i$ $\forall i \in [m]$;
Initialise $M$ as an empty $m \times \ell$ matrix;
For $j \in [\ell]$, $a_j^k$ denotes the $k^{th}$ occurrence of $a_j$ in $x_n^*$ $\forall k \in \{1, 2, \ldots, \mathsf{freq}_j\}$;
for $(i, j) \in [m] \times [\ell]$ do
    if $a_j \in s_i$ then
        $M[i][j] \leftarrow k$, where $k$ is the unique index such that $a_j^k \in \alpha_i$
    end
    else
        $M[i][j] \leftarrow$ no-color
    end
end
$\mathcal{A} \leftarrow$ FindBiColoring$(M)$
$\bar{x} \leftarrow \varepsilon$
for $i \in [n]$ do
    if $\exists j \in [\ell], k \in [\mathsf{freq}_j]$ such that $x_n^*[i] = a_j^k$ then
        if $k = \mathcal{A}[j]$ then
            $\bar{x} \leftarrow \bar{x} \circ x_n^*[i]$
        end
    end
    else
        $\bar{x} \leftarrow \bar{x} \circ x_n^*[i]$
    end
end
```

Algorithm 2 InsertMissing.

```
Result: Inserts missing characters into $\bar{x}$ so that $\bar{x} \in \mathcal{S}_n$
Initialise $\beta_i$ as an arbitrary occurrence of $\mathsf{LCS}(\bar{x}_n, s_i)$ in $s_i$ $\forall i \in [m]$;
$i \leftarrow 1$
while $\mathcal{M} \neq \emptyset$ do
    Pick any $b \in \mathcal{M}$
    $p \leftarrow s_i^{-1}(b)$
    if $\exists r \in [n]$ such that $r < p$, $s_1[r] \in \beta_i$ then
        $q \leftarrow \max\{r \in [n] \mid r < p, \ s_1[r] \in \beta_i\}$
        $j \leftarrow \bar{x}^{-1}(a)$
        $\bar{x} \leftarrow \bar{x}[1 \ldots j] \circ p \circ \bar{x}[j + 1 \ldots \mathsf{len}(\bar{x})]$
    end
    else
        $\bar{x} \leftarrow b \circ \bar{x}$
    end
    $\mathcal{M} \leftarrow \mathcal{M} \setminus \{b\}$
    $i \leftarrow i \mod m + 1$
end
```

Towards Stronger Counterexamples to the Log-Approximate-Rank Conjecture

Arkadev Chattopadhyay ✉

Tata Institute of Fundamental Research, Mumbai, India

Ankit Garg ✉

Microsoft Research India, Bengaluru, India

Suhail Sherif ✉

Vector Institute, Toronto, Canada

Abstract

We give improved separations for the query complexity analogue of the log-approximate-rank conjecture i.e. we show that there are a plethora of total Boolean functions on n input bits, each of which has approximate Fourier sparsity at most $O(n^3)$ and randomized parity decision tree complexity $\Theta(n)$. This improves upon the recent work of Chattopadhyay, Mande and Sherif [6] both qualitatively (in terms of designing a large number of examples) and quantitatively (shrinking the gap from quartic to cubic). We leave open the problem of proving a randomized communication complexity lower bound for XOR compositions of our examples. A linear lower bound would lead to new and improved refutations of the log-approximate-rank conjecture. Moreover, if any of these compositions had even a sub-linear cost randomized communication protocol, it would demonstrate that randomized parity decision tree complexity does not lift to randomized communication complexity in general (with the XOR gadget).

2012 ACM Subject Classification Theory of computation → Oracles and decision trees; Theory of computation → Communication complexity

Keywords and phrases Approximate Rank, Randomized Parity Decision Trees, Randomized Communication Complexity, XOR functions, Subspace Designs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.13

Funding *Suhail Sherif*: This work was mainly done while the author was at Microsoft Research India, Bengaluru.

1 Introduction

The Log-Rank Conjecture (LRC) of Lovasz and Saks asserts that two very seemingly different quantities, one the deterministic communication complexity of a total function f (denoted by $D(f)$) and the other the log of the rank of its communication matrix (denoted by (M_f)) over the field of reals, are essentially the same, i.e. within a fixed polynomial of each other. While this thirty year old conjecture remains wide open, it's natural to try upper-bounding the communication complexity of f by *some* function of the rank of M_f. The best such known bound was obtained by Lovett [21], rather recently, which showed that $D(f)$ is at most the square-root of the rank of M_f, ignoring log factors.

A tempting analog of the LRC for randomized communication complexity appears in a book by Lee and Shraibman [20] where it was named as the Log-Approximate-Rank Conjecture (LARC). Informally, this is LRC with deterministic communication complexity replaced by bounded-error randomized complexity of f, and rank replaced by the *approximate rank* of M_f, where the approximation is uniform point-wise. The LARC is important for several reasons. First, it implies the LRC itself [10]. Second, it implies several other central conjectures, like the polynomial equivalence of quantum and classical communication complexity of total functions [3]. Third, every known lower bound, until very recently,

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 13; pp. 13:1–13:16
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

was no larger than a small polynomial of the log of the approximate rank. Very recently, Chattopadhyay, Mande and Sherif [6] provided a surprisingly simple counterexample to the LARC, that exponentially separated randomized communication complexity from the log of the approximate rank. In particular, their function f has Alice and Bob holding n bits each, the approximate rank of its $2^n \times 2^n$ communication matrix M_f is merely $O(n^2)$ and yet the randomized communication complexity is $\Theta(\sqrt{n})$.

Some questions immediately arise from the above refutation of the LARC. First, is the refutation optimal? There are two ways to measure optimality. The approximate rank and communication complexity are separated by a 4th power. Is this separation true for all functions i.e. is randomized communication complexity always upper bounded by fourth-root of the approximate rank? Interestingly, Gál and Syed [9] recently showed that quantum communication complexity is upper bounded by at most square-root of the approximate rank but for randomized communication, the best upper bound is still linear in the approximate rank. The second way to view optimality is the extent of the gap achieved between log of the approximate rank and communication complexity. This is $O(\log n)$ vs. $\sqrt{n}$ for the current refutation. Can this gap be widened via other functions? This leads us to, of course, the related problem of finding other counter-examples to LARC. Finding a richer set of counter-examples, besides being interesting in their own right, could prove useful for understanding other central conjectures. A concrete example is the question of relative power of quantum and classical protocols to solve total functions, a major open problem. If we have to find a total function with an exponential gap between the quantum communication and randomized communication complexities (if one exists at all), then the function should also have an exponential separation between log of approximate rank and randomized communication complexity.[1] However, it was shown by Anshu et al. [1] and Sinha and de Wolf [25] that the function of [6] has large quantum communication complexity (hence refuting the quantum version of LARC as well). This already motivates the search for other examples refuting the LARC.

Very recently, it has been realized that counter-examples to LARC may find use in an apparently different model of computation to make progress on long-standing problems. In general, the class of log-rank conjectures imply that negation does not offer much advantage: The log of (approximate) non-negative rank of Boolean matrices would have to be polynomially bounded by the log of (approximate) ordinary rank. Separations of (approximate) non-negative rank from (approximate) rank provide evidences of the significant power of negation or cancellation. Using this intuition and the LARC counter-example provided in [6], Chattopadhyay, Datta and Mukhopadhyay [4] constructed a monotone polynomial over the reals that can be computed very efficiently by constant-depth general arithmetic formulas but need exponential size to be computed by monotone arithmetic circuits of unrestricted depth. This provides the first improvement to a classical result of Valiant [26]. No analogue of this improvement is known in the world of Boolean circuits. More generally, in several natural models of computation the precise power of negation or cancellation remains undetermined. We believe that the study of more potential counter-examples to LARC holds the promise of enabling progress on this general problem as well.

In this work, we come up with a rich set of functions that leaves us with the following win-win situation: either every one of these functions gives a stronger refutation of the LARC than what is known or there is no *lifting theorem* for randomized communication complexity of XOR functions. Lifting theorems, in the setting of communication complexity, lift the

[1] Since log of the approximate rank lower bounds quantum communication as well.

complexity of a function f in an appropriate query model to the communication complexity of a problem crafted out of f naturally by block composition with a gadget g, denoted by $f \circ g$. Starting with the celebrated work of Raz and McKenzie [24], they have enabled major progress recently in communication complexity and adjoining areas [12, 8, 11, 5]. In all these theorems, the size of the gadget g is at least logarithmic in the input length of the query function f. A challenging open problem is to prove lifting theorems for a constant size gadget[2]. A natural one is the one-bit[3] XOR gadget. It is not hard to verify that a (randomized) parity decision tree (R)PDT algorithm for f of cost c readily translates into a communication protocol of cost $2c$ for $f \circ \text{XOR}$. A lifting theorem for XOR functions would assert the converse. In other words, a communication protocol cannot be more efficient than naively simulating the optimal RPDT. The strongest evidence for such an assertion is the result of Hatami, Hosseini and Lovett [17] who show that if f has deterministic PDT cost c, then $f \circ \text{XOR}$ has deterministic communication complexity $c^{\Omega(1)}$. While no general result exists for the randomized model, the community believes it to be plausible. We state our main result informally.

▶ **Theorem 1** (Informal). *Assuming XOR lifting theorems for randomized communication complexity, there exists a rich class of functions* $f : \{0,1\}^n \to \{0,1\}$*, such that* $M_{f \circ XOR}$ *has approximate rank* $O(n^3)$ *and* $R(f \circ XOR) = \Theta(n)$*.*

Thus, conditionally, we get the following improvements over the results in [6]: (1) We narrow the gap between approximate rank and randomized communication complexity from quartic to cubic. (2) We expand the gap between log-approximate-rank and randomized complexity from $O(\log n)$ vs. $\sqrt{n}$ to $O(\log n)$ vs. n, thus yielding essentially the strongest possible refutation of the LARC, under plausible assumptions. While this is a nice conceptual way to view our results, it seems proving communication lower bounds for these functions will require new tools and techniques. On the other hand, coming up with non-trivial communication protocol for any of these functions will rule out a PDT to communication lifting theorem for XOR functions in the randomized model.

1.1 Main Ideas

The starting point of our work is to pursue the idea in [6] of looking for functions with small (approximate) spectral norm, i.e. functions whose sum of the magnitude of Fourier coefficients is a small polynomial in n. The previous counterexample to the LARC used the concept of disjoint subcubes to achieve this as every subcube has spectral norm one. This implied that a function f whose set of ones form a union of polynomially many disjoint subcubes will have polynomial spectral norm. The fact that polynomial spectral norm implies polynomial approximate Fourier sparsity, yields that the approximate rank of every such f lifted by XOR is guaranteed to be small. The randomized communication complexity of one such function, $\mathsf{SINK} \circ \text{XOR}$, was shown to be large via a Corruption Bound, the proof of which utilized Shearer's Lemma. The randomized parity decision tree lower bound used a robust subspace-hitting property of the subcubes instead.

In this work, we study a broader class of functions based on disjoint subspaces. The approximate rank of their lifts by XOR is again guaranteed to be small. The main conceptual contribution of our work is to identify a property that is sufficient for every such union

[2] Interesting recent progress on this front, using the AND gadget, was made in the work of Knop, Lovett, McGuire and Yuan [18]

[3] the gadget size here means the number of bits held by each of the two players.

of subspaces to have large RPDT complexity. Remarkably, this property is quite well encapsulated in the concept of Subspace Designs, a notion introduced by Guruswami and Xing [15] and for which explicit constructions were given in [14, 16]. We show that Subspace Designs are hard for RPDTs. The general philosophy of LARC-like conjectures is that randomized complexity of total functions is well captured/characterized by algebraic or analytical measures of the function like (approximate) rank. For instance, a classical result of Nisan and Szegedy [22] confirms this idea in the world of randomized (and quantum) query complexity where the relevant algebraic measure is approximate degree. In the world of PDTs, the natural algebraic notion is approximate Fourier sparsity. The work of [6] refuted this philosophy for parity decision trees via the SINK function, whose approximate Fourier sparsity is $O(n^2)$ and RPDT complexity is $\Theta(\sqrt{n})$. Our lower bounds for functions based on subspace designs yields unconditionally a stronger refutation of this philosophy for the model of parity decision trees. We state here our result in terms of random subspaces because this yields the cleanest formulation.

▶ **Theorem 2** (Main Result). *Let $m = 100n$. Let $\mathcal{V} = \{V_1, V_2, \ldots, V_m\}$ be a set of subspaces of $\{0,1\}^n$ chosen independently and uniformly at random from the set of subspaces of dimension $2n/5$. Let f be the function that outputs 1 on the set $\bigcup_{V\in\mathcal{V}} V$. With probability $1 - o(1)$ the following two statements are true.*

- *Randomized parity decision tree complexity of f is at least $\Omega(n)$.*
- *The spectral norm of f (sum of absolute values of its Fourier coefficients) is upper bounded by $O(n)$ and its approximate Fourier sparsity is upper bounded by $O(n^3)$.*

Hence there exist functions which have a merely cubic gap between approximate Fourier sparsity and RPDT complexity.

The two properties of random subspaces appearing in such a collection that we use are the following: each pair of them have no non-trivial intersection. They also form a (dual) subspace design. We are not able to prove non-trivial lower bounds for the communication problems arising out of Subspace Designs composed with the XOR gadget. However, in Section 3.2, we state concrete conjectures, that seem to be interesting from a Fourier analytic and additive combinatorics point of view, which imply linear lower bounds for such communication problems.

1.2 Organization and plan of the paper

Section 2 contains some basic preliminaries. In Section 3, we prove our main result, a lower bound on the RPDT complexity of a natural class of functions arising out of subspace designs. In Section 3.2, we state a few plausible conjectures and show that they imply a lower bound on the communication complexity of functions arising out of subspace designs composed with the XOR gadget. Finally, we end up with some open problems in Section 4.

2 Preliminaries

In this section, we provide some basic preliminaries needed for the paper. Section 2.1 starts off with some notation. Then in Section 2.2, we present some basic facts about subspaces. Then we introduce the basics of our models of computations, parity decision trees and communication protocols in Section 2.3. Finally, in Section 2.4, we present some basic concepts from Fourier analysis.

2.1 Notation

Given a subspace $S \subseteq \mathbb{F}_2^n$, we use $\mathsf{dim}(S)$ to denote its dimension and $\mathsf{codim}(S)$ to denote its codimension i.e. $n - \mathsf{dim}(S)$. Given the standard bilinear form $\langle \cdot, \cdot \rangle$ on $\mathbb{F}_2^n$, we can define the dual space of S as the set $\{\ell \in \mathbb{F}_2^n \mid \forall x \in S \ \langle \ell, x \rangle = 0\}$. It is a subspace of dimension $n - \mathsf{dim}(S)$ and its dual space is S.

Given a subspace S of dimension k, fix a basis $L = \{\ell_1, \ldots, \ell_{n-k}\}$ of its dual space. For every point $a \in \mathbb{F}_2^{n-k}$, we can define the set $S_a^L = \{x \in \mathbb{F}_2^n \mid \forall i \in [n-k] \ \langle \ell_i, x \rangle = a_i\}$. These are called affine shifts, or cosets, of S. Sets of the kind S_a^L are also called affine subspaces. Each coset of S also has size 2^k. We can also define a coset map of S with respect to a basis of its dual space as

$$\mathsf{coset}_S^L(x) = (\langle \ell_1, x \rangle, \ldots, \langle \ell_{n-k}, x \rangle).$$

It is easy to see that the choice of basis for the dual space does not affect the set of cosets of S. It merely affects the string $a \in \mathbb{F}_2^{n-k}$ that is used to refer to a specific coset. Hence we will refer to the coset map as coset_S, and we may choose an arbitrary basis of the dual space of S in order to interpret the coset map.

From here on, we will use $\{0,1\}$ to refer to $\mathbb{F}_2$. The values 0 and 1 represent the additive and multiplicative identity of $\mathbb{F}_2$.

2.2 Basic facts about subspaces

Here we mention two facts about subspaces that will be useful.

▶ **Lemma 3** (Disjoint Subspaces). *Let S be a subspace of $\{0,1\}^n$ of dimension d_1. Let T be a subspace of $\{0,1\}^n$ of dimension d_2 chosen uniformly at random. Then $\Pr_T[S \cap T = \{0\}] \geq 1 - n2^{d_1+d_2-n}$.*

Proof. Let us generate T by choosing d_2 vectors $\{v_1, \ldots, v_{d_2}\}$, each vector independent of the previous ones, in order to form a basis for T. The subspace S intersects T trivially if and only if for all $i \in [d_2]$, $v_i \notin \text{span}(\{v_j\}_{j<i} \cup S)$. We call these events $E_1, \ldots, E_{d_2}$. When choosing v_i to add to the basis for T, there are $2^n - 2^{i-1}$ choices, since $|\text{span}(\{v_j\}_{j<i})| = 2^{i-1}$. Conditioned on $E_1, \ldots, E_{i-1}$, we also know that $|\text{span}(\{v_j\}_{j<i} \cup S)| = 2^{i-1+d_1}$. The probability of E_i occurring is

$$\frac{|(\{0,1\}^n \setminus \text{span}(\{v_j\}_{j<i})) \setminus \text{span}(\{v_j\}_{j<i} \cup S)|}{|\{0,1\}^n \setminus \text{span}(\{v_j\}_{j<i})|} = \frac{|\{0,1\}^n \setminus \text{span}(\{v_j\}_{j<i} \cup S)|}{|\{0,1\}^n \setminus \text{span}(\{v_j\}_{j<i})|}.$$

We can then calculate the probability of $S \cap T = \emptyset$ as

$$\begin{aligned}\Pr\left[\bigcap_{i\in[d_2]} E_i\right] &= \prod_{i=1}^{d_2} \Pr\left[E_i \mid E_1, \cdots, E_{i-1}\right] = \prod_{i=1}^{d_2} \frac{2^n - 2^{d_1+i-1}}{2^n - 2^{i-1}} \\ &\geq \left(1 - \frac{2^{d_1+d_2}}{2^n}\right)^{d_2} \geq 1 - \frac{d_2}{2^{n-d_1-d_2}}.\end{aligned}$$ ◀

▶ **Lemma 4.** *Let V and W be affine subspaces of $\{0,1\}^n$ satisfying*

$$\frac{|V \cap W|}{|W|} < \frac{|V|}{2^n}.$$

Then $V \cap W = \emptyset$.

Proof. Let $\{\langle v_i, x\rangle = a_i\}_{i\in[k]}$ be the constraints defining the affine subspace W. Let $W_0, W_1, \cdots, W_k$ be the affine spaces defined as follows. The constraints for W_j are $\{\langle v_i, x\rangle = a_i\}_{i\in[j]}$. Clearly $W_0 = \{0,1\}^n$ and $W_k = W$.

Now let us assume that $|V \cap W_i| \neq 0$ and is hence an affine subspace. The set $V \cap W_{i+1}$ is the same affine subspace with the added constraint $\langle v_{i+1}, x\rangle = a_{i+1}$.

- If this constraint was already implied by the constraints in $V \cap W_i$, then $|V \cap W_{i+1}| = |V \cap W_i|$.
- If this constraint is incompatible with the constraints in $V \cap W_i$, then $|V \cap W_{i+1}| = 0$.
- If this constraint was independent of the constraints in $V \cap W_i$, then $|V \cap W_{i+1}| = |V \cap W_i|/2$.

Hence $|V \cap W_k|$ is either 0 or is at least $|V \cap W_0|/2^k$. On the other hand, $|W|/2^n = 1/2^k$. Since $V \cap W_k = V \cap W$ and $V \cap W_0 = V$, we can rewrite this as

$$V \cap W \neq \emptyset \implies \frac{|V \cap W|}{|V|} \geq \frac{|W|}{2^n}. \qquad ◀$$

2.3 Parity decision trees, communication complexity and the corruption bound

We now define parity decision trees, aimed at computing functions of the form $f : \{0,1\}^n \to \{0,1\}$.

▶ **Definition 5** (Parity Decision Tree). *A parity decision tree T is a binary tree rooted at a node r satisfying the following properties.*

- *Each internal node is labelled with a set $S \subseteq [n]$.*
- *Each internal node has two children, with one of the edges labelled with a 0 and the other labelled with a 1.*
- *Each leaf has a label from $\{0,1\}$.*

A parity decision tree outputs a value $a \in \{0,1\}$ on given an input $x \in \{0,1\}^n$ as follows. The "current node" below is initialized to the root node r.

- *The tree computes $b = \oplus_{i\in S} x_i$, where S is the label on the current node.*
- *The tree moves to the child that is reached by taking the edge labelled b. If the child is a leaf, output the label of the leaf. Else, repeat the previous step with the child as the current node.*

The cost of the parity decision tree is defined as the height of the tree.

▶ **Definition 6** (Randomized Parity Decision Tree). *A randomized parity decision tree (RPDT) of cost c is a distribution over deterministic parity decision trees of cost c. The output of the RPDT on an input x is the random variable defined as the output of T on x, where T is a parity decision tree sampled as per the distribution specified by the RPDT.*

The ϵ-error RPDT complexity of a function f, denoted $\mathsf{R}^{\oplus}_{\epsilon}(f)$, is the minimum cost of an RPDT T such that $\forall x, \Pr[f(x) = T(x)] \geq 1 - \epsilon$.

▶ **Lemma 7** (Corruption, RPDT version). *Let $f : \{0,1\}^n \to \{0,1\}$. Let μ be a distribution on $\{0,1\}^n$ such that $\mu(f^{-1}(0)) = 1/2$. Let $\epsilon \leq 1/8$. Then an ϵ-error cost-c RPDT computing f implies the existence of an affine subspace W such that*

- *$\mu(W \cap f^{-1}(1)) \leq 4\epsilon\mu(W)$ and*
- $\mathsf{codim}(W) \leq c$.

Proof. Note that an ϵ-error cost-c RPDT T computing f implies that for any distribution μ over the inputs of f, there is an RPDT whose expected error, $\mathbb{E}_{T,x\sim\mu}[|T(x)-f(x)|]$, is at most ϵ. Since T is a distribution over deterministic parity decision trees, there is a deterministic parity decision tree whose expected error is also at most ϵ.

Suppose that a subspace such as the one posited in the lemma statement did not exist. Then for any cost-c parity decision tree T, we may compute the error made as follows. Note that the set of inputs that reach any specific leaf forms an affine subspace of codimension at most c, with each pair of such affine subspaces being disjoint. Let $\mathcal{L}$ be the set of these affine subspaces corresponding to the leaves of T that are labelled 0. Then $\sum_{V\in\mathcal{L}}\mu(V) \geq 1/2-\epsilon$, since otherwise T would be outputting 1 on more than an ϵ mass of 0-inputs. But then $\sum_{V\in\mathcal{L}}\mu(V\cap f^{-1}(1)) \geq \sum_{V\in\mathcal{L}} 4\epsilon\mu(V) \geq 4\epsilon(1/2-\epsilon) \geq 2\epsilon-4\epsilon^2 > \epsilon$. So on more than an ϵ mass of 1-inputs, T outputs 0. Hence the tree T is erring on a larger than ϵ mass of inputs and we have a contradiction. ◀

We now move to communication complexity. We are concerned with the number of bits that two parties Alice and Bob need to communicate in order to compute a function $F:\mathcal{X}\times\mathcal{Y}\to\{0,1\}$. See [19] for a thorough introduction to the topic. We will use that a deterministic communication protocol of cost c partitions the input space of F into at most 2^c rectangles (sets of the form $A\times B$ for $A\subseteq\mathcal{X}, B\subseteq\mathcal{Y}$), and it outputs the same value on all inputs in a rectangle. Randomized communication is defined akin to randomized parity decision trees.

▶ **Definition 8** (Randomized Communication Protocol). *A randomized communication protocol of cost c is a distribution over deterministic communication protocols of cost c. The output of the randomized communication protocol on an input x is the random variable defined as the output of T on (x,y), where T is a communication protocol sampled as per the distribution specified by the randomized communication protocol.*

The ϵ-error randomized communication complexity of a function F is the minimum cost of an randomized communication protocol T such that $\forall x,y, \Pr[F(x,y)=T(x,y)]\geq 1-\epsilon$.

The following is a lower-bound technique for randomized communication complexity akin to the lower bound for RPDTs given previously. This technique is well-known with roots in [27].

▶ **Lemma 9** (Corruption). *Let $F:\{0,1\}^n\to\{0,1\}$. Let ν be a distribution on $\{0,1\}^n$ such that $\nu(F^{-1}(0))=1/2$. Let $\epsilon<1/8$. Then an ϵ-error cost-c randomized communication protocol computing F implies the existence of a rectangle R such that*

- *$\nu(R\cap F^{-1}(1))\leq 4\epsilon\nu(R)$ and*
- *$\nu(R)\geq 2^{-c-3}$.*

2.4 Basic notions from Fourier analysis

We now move to Fourier analysis, a particularly useful tool in analyzing Boolean functions. We define the parity functions as follows. For each $S\subseteq[n]$, we define a parity function $\chi_S:\{0,1\}^n\to\{-1,1\}$ as $\chi_S(x)=(-1)^{\sum_{i\in S}x_i}$. These form an orthonormal basis for the class of functions from $\{0,1\}^n$ to $\mathbb{R}$ under the inner product $\langle f,g\rangle=\frac{1}{2^n}\sum_{x\in\{0,1\}^n}f(x)g(x)$. Hence every such function f can be written as $\sum_S\hat{f}(S)\chi_S$. The values $\hat{f}(S)$ are referred to as Fourier coefficients and can be computed as $\langle f,\chi_S\rangle$. Let $\hat{f}$ denote the vector $(\hat{f}(S))_{S\subseteq[n]}\in\mathbb{R}^{2^n}$, known as the Fourier spectrum. We define the following measures of f.

- The sparsity of f is $\|\hat{f}\|_0$.
- The spectral norm of f is $\left\|\widehat{f}\right\|_1$.
- The ϵ-approximate sparsity of f, $\|\hat{f}\|_{0,\epsilon}$, is $\min_{g:\forall x\ |g(x)-f(x)|\leq\epsilon}\|\hat{g}\|_0$.
- The ϵ-approximate spectral norm of f, $\left\|\widehat{f}\right\|_{1,\epsilon}$, is $\min_{g:\forall x\ |g(x)-f(x)|\leq\epsilon}\|\widehat{g}\|_1$.

The Fourier spectrum of a subspace is easy to compute. (See, for instance, [23].) It follows from the spectrum that any subspace $V \subseteq \{0,1\}^n$, the function $\mathbb{1}_V$ satisfies $\left\|\widehat{\mathbb{1}_V}\right\|_1 = 1$.

For a function $f : \{0,1\}^n \to \mathbb{R}$ its composition with XOR, denoted $f \circ \mathsf{XOR}$, is a function $F : \{0,1\}^n \times \{0,1\}^n \to \mathbb{R}$ defined as $F(x,y) = f(x \oplus y)$ where $x \oplus y$ is the bitwise XOR of x and y.

It is a well known fact that for a function $F := f \circ \mathsf{XOR}$, the rank of the communication matrix of F, denoted $\mathsf{rank}(F)$, is equal to $\|\hat{f}\|_0$. The ϵ-approximate rank of F is at most the ϵ-approximate sparsity of f.

We note a theorem useful in showing that a function has small approximate sparsity.

▶ **Theorem 10** (Grolmusz's Theorem [2, 13, 28, 6]). *For any $f : \{0,1\}^n \to \{0,1\}$ and $\delta > \epsilon \geq 0$,*

$$\|\hat{f}\|_{0,\delta} \leq O\left(\left\|\widehat{f}\right\|_{1,\epsilon}^2 n/(\delta-\epsilon)^2\right).$$

We conclude the preliminaries with the useful notion of entropy.

▶ **Definition 11** (Entropy). *Let X be a discrete random variable. The entropy $H(X)$ is defined as*

$$H(X) := \sum_{s\in\mathrm{supp(X)}} \Pr[X=s]\log\left(\frac{1}{\Pr[X=s]}\right).$$

▶ **Fact 12** (Folklore). $|\mathrm{supp}(X)| = k \implies H(X) \leq \log k$*, with equality if and only if X is uniform.*

3 The RPDT Complexity of Dual Subspace Designs

In this section, we prove a lower bound on the RPDT complexity of a natural class of functions arising from subspace designs. A subspace design is a set of subspaces such that any small dimensional subspace non-trivially intersects only a few members of the set. (These are referred to as weak subspace designs in [14].)

▶ **Definition 13** (Subspace Design). *An n-dimensional (s,h)-subspace design is a set of subspaces $\{S_1, S_2, \cdots, S_m\}$ of $\{0,1\}^n$ such that for all subspaces T of dimension at most s, at most h of the m subspaces intersect T non-trivially.*

We call a set of subspaces $\{V_1, V_2, \cdots, V_m\}$ of $\{0,1\}^n$ an n-dimensional (s,h)-dual subspace design if their duals form an (s,h)-subspace design. Dual subspace designs have an alternate characterization based on the notion of independent subspaces.

▶ **Definition 14** (Independent Subspaces). *Subspaces $S, T \subseteq \{0,1\}^n$ are independent if their coset maps are independent. That is, let L_S and L_T be arbitrary bases for the dual spaces of S and T. For a variable x chosen uniformly at random from $\{0,1\}^n$, consider the random variables $\mathsf{coset}_S(x)$ and $\mathsf{coset}_T(x)$. For every $a \in \mathbb{F}_2^{\mathsf{codim}(S)}, b \in \mathbb{F}_2^{\mathsf{codim}(T)}$, we want that* $\Pr[\mathsf{coset}_S(x) = a | \mathsf{coset}_T(x) = b] = \Pr[\mathsf{coset}_S(x) = a] = 2^{-\mathsf{codim}(S)}$.

In particular this implies that every coset of S intersects with every coset of T.

We now state the alternate characterization of dual subspace designs.

▷ Claim 15. The set $\{V_1, V_2, \cdots, V_m\}$ of $\{0,1\}^n$ is an n-dimensional (s,h)-dual subspace design if and only if for all subspaces W of *codimension* at most s, at least $m-h$ of the m subspaces are *independent* from W.

This claim follows from the following lemma relating trivial subspace intersections and independent subspaces.

▶ **Lemma 16** (Independent Subspaces). *Subspaces S and T of $\mathbb{F}_2^n$ are independent if and only if the dual space of S and the dual space of T intersect trivially (i.e. only at the point $0 \in \mathbb{F}_2^n$).*

Proof. Let V and W be the dual spaces of S and T respectively. If V and W intersected at a non-zero point $\ell \in \mathbb{F}_2^n$, then consider bases L_S and L_T for V and W respectively, wherein ℓ is the first element of L_S and also the first element of L_T. The coset maps of S and T with this choice of L_S and L_T cannot be independent since for all $x \in \mathbb{F}_2^n$, the first entries of $\mathsf{coset}_S^{L_S}(x)$ and $\mathsf{coset}_T^{L_T}(x)$ will always agree.

For the other direction, let L_S and L_T be arbitrary bases for V and W respectively. We will show that if V and W intersect trivially, then the coset maps are independent. Assuming V and W intersect trivially, this means that $\mathrm{span}(L_S) \cap \mathrm{span}(L_T) = \{0\}$. Hence $L = L_S \cup L_T$ is an independent set of size $\mathsf{dim}(V) + \mathsf{dim}(W)$. Consider the subspace X with basis L, and let R be its dual subspace. The cosets of R each have size $2^{n-\mathsf{dim}(V)-\mathsf{dim}(W)}$. For any $a \in \mathbb{F}_2^{\mathsf{codim}(S)}, b \in \mathbb{F}_2^{\mathsf{codim}(T)}$, the set $\{x \mid \mathsf{coset}_S^{L_S}(x) = a \wedge \mathsf{coset}_T^{L_T}(x) = b\}$ is a coset of R. Hence $\Pr[\mathsf{coset}_S^{L_S}(x) = a | \mathsf{coset}_T^{L_T}(x) = b] = 2^{-\mathsf{dim}(V)-\mathsf{dim}(W)}/2^{-\mathsf{dim}(W)} = 2^{-\mathsf{dim}(V)}$ ◀

A useful corollary of Claim 15 is that an (s,h)-dual subspace design also forms a hitting set for the set of all affine subspaces of codimension at most s. We will use this fact to lower bound the randomized parity decision tree complexity of unions of subspaces.

▶ **Corollary 17.** *Let $\{V_1, V_2, \cdots, V_m\}$ be an n-dimensional (s,h)-dual subspace design. For all affine subspaces W of* codimension *at most s, at least $m-h$ of the m subspaces intersect with W.*

Proof. This follows from Claim 15 and the fact that if two subspaces S and T are independent, then S will intersect any affine shift of T non-trivially. ◀

We are now ready to prove the main theorem of the section.

▶ **Theorem 18.** *Let $\mathcal{V}$ be an n-dimensional (s,h)-dual subspace design of size m.*

Let f be the function defined as $f^{-1}(1) = \bigcup_{V \in \mathcal{V}} V$. We now show that $\mathsf{R}_\epsilon^\oplus(f) \geq s$ as long as $\epsilon < \frac{m-h}{8m} \frac{|f^{-1}(0)|}{2^n}$.

Proof. Consider the distribution μ defined over the inputs of f as follows.

- Sample $z \sim_{\mathsf{unif}} \{0,1\}$.
- If $z = 0$, output a uniformly random input from $f^{-1}(0)$.
- Otherwise, sample $V \sim_{\mathsf{unif}} \mathcal{V}$.
- Output a uniformly random input from V.

Assuming that f is computed by an ϵ-error cost c RPDT, Lemma 7 implies the existence of a subspace W such that

- $\mu(W \cap f^{-1}(1)) \leq 4\epsilon\mu(W)$ and
- $\mathsf{codim}(W) \leq c$.

Assume we have a W such that $\mu(W \cap f^{-1}(1)) \leq 4\epsilon\mu(W)$. This means that $\mu(W \cap f^{-1}(1)) \leq \frac{4\epsilon}{1-4\epsilon}\mu(W \cap f^{-1}(0))$. We also know the following from the definition of μ.

$$\mu(W \cap f^{-1}(1)) = 1/2 \cdot \frac{1}{|\mathcal{V}|}\sum_{V \in \mathcal{V}} \frac{|W \cap V|}{|V|}$$

$$\mu(W \cap f^{-1}(0)) = 1/2 \cdot \frac{|W \cap f^{-1}(0)|}{|f^{-1}(0)|} \leq 1/2 \cdot \frac{|W|}{|f^{-1}(0)|}$$

Putting these together, we get that

$$\frac{1}{|\mathcal{V}|}\sum_{V \in \mathcal{V}} \frac{|W \cap V|}{|V|} \leq \frac{4\epsilon}{1-4\epsilon}\frac{|W|}{|f^{-1}(0)|}.$$

Now if $\epsilon < \frac{m-h}{8m}\frac{|f^{-1}(0)|}{2^n} \leq \frac{1}{8}$, then $\frac{4\epsilon}{1-4\epsilon} < \frac{m-h}{m}\frac{|f^{-1}(0)|}{2^n}$. This implies that less than $m-h$ subspaces of $\mathcal{V}$ can satisfy $\frac{|W \cap V|}{|V|} \geq \frac{|W|}{2^n}$, and hence more than h of them *must* satisfy $\frac{|W \cap V|}{|V|} < \frac{|W|}{2^n}$. This means that $W \cap V = \emptyset$ (Lemma 4). In other words, W is an affine subspace that managed to evade more than h subspaces of $\mathcal{V}$. But by Corollary 17, if W is of codimension at most s, then it is disjoint from at most h subspaces of $\mathcal{V}$. So W must be of codimension more than s.

Hence the cost of the RPDT must also be more than s. ◀

▶ Remark 19. The above proof would more generally work for the union of any set of affine subspaces that forms a hitting set for the set of all large affine subspaces.

3.1 Narrowing the gap between RPDT complexity and approximate sparsity to cubic

In this section, we instantiate Theorem 18 with random subspaces to get a mere cubic gap between RPDT complexity and approximate sparsity. It is known that there are efficient probabilistic constructions of subspace designs. We go through such a construction here, and use it to show our main theorem.

▶ **Theorem 20.** *Let $m = 100n$. Let $V_1, V_2, \ldots, V_m$ be subspaces of $\{0,1\}^n$ chosen independently and uniformly at random from the set of subspaces of dimension $2n/5$. With probability $1 - o(1)$ the following two statements are true.*

- *$\mathcal{V} = \{V_1, \ldots, V_m\}$ forms an $(n/5, m/10)$-dual subspace design.*
- *Every pair of subspaces in $\mathcal{V}$ intersects trivially.*

Proof. Let W be a fixed affine subspace of $\{0,1\}^n$ of dimension $4n/5$. Let $\mathcal{V} = \{V_1, V_2, \cdots, V_m\}$ be subspaces of $\{0,1\}^n$ chosen independently and uniformly at random from the set of subspaces of dimension $2n/5$.

Since the duals of W and V_1 have dimension $3n/5$ and $n/5$ respectively, the probability that W and V_1 are independent is at least $1 - n2^{-n/5}$ (Lemma 3). This is independently true of W and each $V \in \mathcal{V}$. The probability that W is not independent with *at least* $m/10$ of the m subspaces is at most $\binom{m}{m/10}(n2^{-n/5})^{m/10}$.

Since the number of subspaces of dimension $4n/5$ is at most $(2^n)^{4n/5} = 2^{4n^2/5}$, the probability that there exists such a subspace W that is not independent with at least $m/10$ of the subspaces in $\mathcal{V}$ is at most $2^{4n^2/5}\binom{m}{m/10}(n2^{-n/5})^{m/10}$.

Setting $m = 100n$, this upper bound is at most $2^{.8n^2+100n+10n\log n-2n^2} = o(1)$.

Hence with high probability, $\mathcal{V}$ is an $(n/5, m/10)$-dual subspace design.

Let f be defined as in the theorem statement. Note that since V_1 and V_2 are random subspaces of dimension $2n/5$, the probability that they intersect only at 0 is at least $1 - n2^{-n/5}$. The probability that any two subspaces in $\mathcal{V}$ intersect at more than just 0 is at most $\binom{m}{2} n2^{-n/5} = o(1)$. ◀

▶ **Theorem 2** (Main Result)**.** *Let $m = 100n$. Let $\mathcal{V} = \{V_1, V_2, \ldots, V_m\}$ be a set of subspaces of $\{0,1\}^n$ chosen independently and uniformly at random from the set of subspaces of dimension $2n/5$. Let f be the function that outputs 1 on the set $\bigcup_{V \in \mathcal{V}} V$. With probability $1 - o(1)$ the following two statements are true.*

- *Randomized parity decision tree complexity of f is at least $\Omega(n)$.*
- *The spectral norm of f (sum of absolute values of its Fourier coefficients) is upper bounded by $O(n)$ and its approximate Fourier sparsity is upper bounded by $O(n^3)$.*

Hence there exist functions which have a merely cubic gap between approximate Fourier sparsity and RPDT complexity.

Proof. We know from Theorem 20 that with probability $1 - o(1)$ the set $\mathcal{V}$ forms an $(n/5, m/10)$-dual subspace design. We also can trivially lower bound $|f^{-1}(0)|/2^n$ by $1 - m2^{-3n/5}$. Since $\mathcal{V}$ is an $(n/5, m/10)$-dual subspace design, we can conclude from Theorem 18 that for $\epsilon \leq 1/10$, $\mathsf{R}^{\oplus}_{\epsilon}(f) \geq n/5$.

We also know from Theorem 20 that with probability $1-o(1)$, every pair of subspaces from $\mathcal{V}$ intersects trivially. When this event holds, f can be represented as $\sum_{V \in \mathcal{V}} \mathbb{1}_V - (m-1)\mathbb{1}_{V_0}$ where $V_0 = \{0\}$ is the trivial subspace of dimension 0. Since the spectral norm of a subspace is equal to 1, the spectral norm of f is upper bounded by $m + m - 1 < 2m$. Using Theorem 10, this also implies that $\|\hat{f}\|_{0,\epsilon} \leq O(m^2 n/\epsilon^2) = O(n^3)$ for any constant ϵ.

This concludes the proof of the merely cubic gap. ◀

3.2 On Extending this to Communication

In this section, we state a plausible conjecture that would imply a lower bound on the randomized communication complexity of XOR compositions of our functions.

In the RPDT lower bound, we showed that in order for an affine subspace to avoid most of the subspaces of a dual subspace design, the codimension of the affine subspace needs to be large. We could hope for a similar statement in the communication world: For a rectangle to put very little mass on most of the subspaces making up a dual subspace design (i.e., puts very little mass on inputs (x, y) such that $x \oplus y$ lies in the subspaces), the mass of the rectangle must be $2^{-\Omega(n)}$. One particularly neat conjecture that would imply that statement is the following, in which $\mathcal{U}_k$ denotes the uniform distribution over k elements. (See the proof of Theorem 23 for an implicit proof of the implication.)

▶ **Conjecture 21.** *There exist constants $0 < \alpha < 1$, $\beta > 0$ and $k \geq 1$ such that the following holds. Let $\mathcal{V} = \{V_1, \ldots, V_m\}$ be an n-dimensional (s,h)-dual subspace design. Let B_i be the coset map of V_i. Let X be a random variable over $\{0,1\}^n$ such that $\|B_i(X) - \mathcal{U}_{2^{\mathsf{codim}(V_i)}}\|_1 \geq \alpha$ for more than kh values of $i \in [m]$. Then $H(X) \leq n - \beta s$.*

The merely cubic gap in the RPDT world used random subspaces. So for extending it to communication, it would be okay for us to bypass dual subspace designs and prove the theorem for random subspaces instead.

▶ **Conjecture 22.** *There exists a constant $0 < \alpha < 1, \beta > 0$ such that the following holds. Let $m = 100n$. Let $V_1, V_2, \ldots, V_m$ be random subspaces of $\{0,1\}^n$ of dimension $2n/5$, and let $B_1, B_2, \cdots, B_m$ be their coset maps. Let X be a random variable over $\{0,1\}^n$ such that $\|B_i(X) - \mathcal{U}_{2^{3n/5}}\|_1 \geq \alpha$ for at least $m/3$ values of $i \in [m]$. Then with high probability, $H(X) \leq n - \beta n$.*

First of all note that the conjectures are true when X is the uniform distribution over an affine subspace. To see this, suppose X is the uniform distribution over an affine subspace W. $H(X) \geq n - s$ is the same as saying that $\mathsf{codim}(W) \leq s$. Then by Claim 15, for at least $m - h$ of the subspaces $V_1, \ldots, V_m$, V_i and the dual space of W are independent, which implies that $B_i(X)$ will be exactly uniform ($\mathcal{U}_{2^{\mathsf{codim}(V_i)}}$).

We discuss now why the Conjectures 21 and 22 appear to be a bit tricky to prove. While the conjectures are true for affine subspaces, the number of distributions (or even the number of subsets of $\{0,1\}^n$) are much larger (doubly exponential in n), so the conjectures are a leap of faith in this sense. But we haven't been able to come up with counterexamples and it would be very interesting to do so. The conceptual way to view the conjectures, e.g. Conjecture 22 to be concrete, is that if a random variable X has the property that when projected down to $2n/5$ bits in various ways it loses $\Omega(1)$ bits of entropy, then X overall loses $\Omega(n)$ bits of entropy. Shearer's lemma talks about these kind of statements. While in Shearer's lemma, the projections are onto subcubes, there are generalizations called Brascamp-Lieb inequalities which talk about more general projections (e.g. see [7]). However, the Brascamp-Lieb inequalities can at best guarantee an $\Omega(n/k)$-bit entropy loss in X if there is an $\Omega(1)$-bit entropy loss while projecting X to k bits in various ways. What we want is much stronger. This is one difficulty.

The other difficulty is that a Fourier type approach doesn't seem to work either. One can control $\|B_i(X) - \mathcal{U}_{2^{\mathsf{codim}(V_i)}}\|_1$ by bounding the ℓ_2 distance and then trying to bound the Fourier coefficients of the distribution of X on the dual space of V_i. But this doesn't give any meaningful bound (if done in a naive way at least).

We now state the lower bound on the randomized communication complexity of a dual subspace design composed with XOR that we get assuming Conjecture 21. For a set of subspaces in n dimensions $\mathcal{V} = \{V_1, V_2, \ldots, V_m\}$, let $f_{\mathcal{V}}$ be the function on n bits that outputs 1 on inputs in $\cup_{V \in \mathcal{V}} V$.

▶ **Theorem 23.** *Let us assume Conjecture 21 holds with constants α, β and k. Let $\mathcal{V} = \{V_1, V_2, \ldots, V_m\}$ be an n-dimensional (s,h)-dual subspace design and define γ so that $|\cup_{V \in \mathcal{V}} V| = \gamma 2^n$. Let $F = f_{\mathcal{V}} \circ \mathsf{XOR}$. For $\epsilon < \frac{(1-\alpha)^2}{4}\frac{m-2kh}{8m}(1-\gamma)$, the ϵ-error randomized communication complexity of F is at least $\beta s + \log(1-\gamma)$.*

We will prove this with $\alpha = 1/2$ to reduce symbol clutter. After the proof we discuss how the proof would change with a different value of α. Note that the bound on ϵ in the above theorem statement simplifies to $\frac{m-2kh}{128m}(1-\gamma)$ when $\alpha = 1/2$.

Proof when $\alpha = 1/2$. For any $V \in \mathcal{V}$, let $S_V = \{(x,y) \in \{0,1\}^n \times \{0,1\}^n \mid x \oplus y \in V\}$. Note that $|S_V| = 2^n |V|$ and $F^{-1}(1) = \cup_{V \in \mathcal{V}} S_V$. Consider the distribution ν defined over the inputs of F as follows.

- Sample $z \sim_{\mathsf{unif}} \{0,1\}$.
- If $z = 0$, output a uniformly random input from $F^{-1}(0)$.
- Otherwise, sample $V \sim_{\mathsf{unif}} \mathcal{V}$.
- Output a uniformly random input from S_V.

Assuming F is computed by an ϵ-error cost c communication protocol, Lemma 9 implies the existence of a "large biased rectangle" R satisfying

- $\nu(R \cap F^{-1}(1)) \leq 4\epsilon\nu(R)$ and
- $\nu(R) \geq 2^{-c-3}$.

Fix such an R. The condition $\nu(R \cap F^{-1}(1)) \leq 4\epsilon\nu(R)$ is the same as $\nu(R \cap F^{-1}(1)) \leq \frac{4\epsilon}{1-4\epsilon}\nu(R \cap F^{-1}(0))$. We also know the following from the definition of ν.

$$\nu(R \cap F^{-1}(1)) = {}^1\!/\!_2 \cdot \frac{1}{|\mathcal{V}|} \sum_{V \in \mathcal{V}} \frac{|R \cap S_V|}{|S_V|}$$

$$\nu(R \cap F^{-1}(0)) = {}^1\!/\!_2 \cdot \frac{|R \cap F^{-1}(0)|}{|F^{-1}(0)|} \leq {}^1\!/\!_2 \cdot \frac{|R|}{|F^{-1}(0)|}$$

Putting these together, we get that

$$\frac{1}{|\mathcal{V}|} \sum_{V \in \mathcal{V}} \frac{|R \cap S_V|}{|S_V|} \leq \frac{4\epsilon}{1-4\epsilon} \frac{|R|}{|F^{-1}(0)|}.$$

Now if $\epsilon < \frac{m-2kh}{128m} \frac{|F^{-1}(0)|}{2^{2n}} < 1/8$, then $\frac{4\epsilon}{1-4\epsilon} < \frac{m-2kh}{16m} \frac{|F^{-1}(0)|}{2^{2n}}$. This implies that less than $m - 2kh$ subspaces of $\mathcal{V}$ can satisfy $\frac{|R \cap S_V|}{|S_V|} \geq \frac{|R|}{16 \cdot 2^{2n}}$, and hence more than $2kh$ of them *must* satisfy $\frac{|R \cap S_V|}{|S_V|} < \frac{|R|}{16 \cdot 2^{2n}}$. Let us fix such a V.

Let coset_V denote the function $\mathsf{coset}_V^{L_V}$ for some fixed basis L_V of the dual space of V. Let $R = A \times B$. Then $\frac{|R \cap S_V|}{|R|}$ is the probability that, when x and y are sampled uniformly at random from A and B, $\mathsf{coset}_V(x) = \mathsf{coset}_V(y)$. Let A_V be the distribution of $\mathsf{coset}_V(x)$ and B_V be the distribution of $\mathsf{coset}_V(y)$. The condition $\frac{|R \cap S_V|}{|R|} < \frac{|S_V|}{16 \cdot 2^{2n}}$ can be rewritten as

$$\Pr_{x' \sim A_V, y' \sim B_V}[x' = y'] < \frac{|S_V|}{16 \cdot 2^{2n}} = \frac{1}{16 \cdot 2^{\mathsf{codim}(V)}}.$$

It follows that $A_V(S) < 1/4$ where $S = \{y' \mid B_V(y') \geq \frac{1}{4 \cdot 2^{\mathsf{codim} V}}\}$. However, $B_V(S)$ must be at least ${}^3\!/\!_4$, since $B_V(\overline{S}) \leq {}^1\!/\!_4$.

Hence A_V and B_V have total variational distance at least ${}^1\!/\!_2$, and $\|A_V - B_V\|_1 \geq 1$. By the triangle inequality, $\max\{\|A_V - \mathcal{U}_{2^{\mathsf{codim}(V)}}\|_1, \|B_V - \mathcal{U}_{2^{\mathsf{codim}(V)}}\|_1\} \geq {}^1\!/\!_2$.

Hence, either there are more than kh subspaces that satisfy $\|A_V - \mathcal{U}_{2^{\mathsf{codim}(V)}}\| \geq {}^1\!/\!_2$ or there are more than kh subspaces that satisfy $\|B_V - \mathcal{U}\| \geq {}^1\!/\!_2$. Without loss of generality we assume the former. Now we use our conjecture. The conjecture implies that $H(A) \leq n - \beta s$. Hence $\frac{|R|}{2^{2n}} \leq 2^{-\beta s}$.

We now want to move from $|R|$ being small under the uniform distribution to R being small under ν. We know that $\nu(R \cap F^{-1}(1)) \leq 4\epsilon\nu(R) < \nu(R)/2$, so $\nu(R \cap F^{-1}(0)) \geq \nu(R)/2$. We also know from the definition of ν that

$$\nu(R \cap F^{-1}(0)) = \frac{|R \cap F^{-1}(0)|}{2|F^{-1}(0)|} \leq \frac{|R|}{2 \cdot 2^{2n}} \cdot \frac{2^{2n}}{|F^{-1}(0)|} \leq 2^{-\beta s - 1} \cdot \frac{1}{1-\gamma}.$$

So $\nu(R) \leq 2\nu(R \cap F^{-1}(0)) \leq 2^{-\beta s - 1 - \log(1-\gamma)}$. Hence the cost of the protocol is at least $\beta s + \log(1-\gamma) - 3$. ◀

We now explain how to modify the proof assuming the conjecture were true for other values of α. The reasoning involved in the proof does not change, we just need to track the modified values. The following modified values appear when analyzing a large biased rectangle $R = A \times B$.

- We would find more than $2kh$ subspaces V such that $\Pr[A_V = B_V] < \frac{(1-\alpha)^2}{4} \frac{|S_V|}{2^{2n}}$.
- We would then set $S = \{y' \mid B_V(y') \geq \frac{1-\alpha}{2 \cdot 2^{\mathsf{codim}(V)}}\}$. This would mean that $A_V(S) \leq \frac{1-\alpha}{2}$ and $B_V(S) \geq 1 - \frac{1-\alpha}{2}$. Hence $\|A_V - B_V\|_1 \geq 2\alpha$, and one of A or B (wlog, A) satisfies $\|A_V - \mathcal{U}_{2^{\mathsf{codim}(V)}}\|_1 \geq \alpha$ for at least kh subspaces from the dual subspace design.

- The conjecture would then tell us that $\frac{|R|}{2^{2n}} \leq 2^{-\beta s}$ and we would use that to conclude that the cost of the protocol would be at least $\beta s + \log(1-\gamma) - 3$, which is $\Omega(s)$ for constant γ.

Given this lower bound, we would want to apply it to get a merely cubic gap between randomized communication complexity and approximate rank along the lines of Theorem 20.

▶ **Corollary 24.** *Let $\mathcal{V} = \{V_1, V_2, \ldots, V_m\}$ be an $(n/5, m/20k)$-dual subspace design with (1) $m = 200kn$, (2) each subspace having dimension $2n/5$ and (3) every pair of subspaces intersecting trivially. Let $F = f_{\mathcal{V}} \circ \mathsf{XOR}$. Then assuming Conjecture 21,*

- *The $1/10$-error randomized communication complexity of F is $\Omega(n)$.*
- $\mathsf{rank}_{1/10}(F) = O(n^3)$.

Proof. The size of $F^{-1}(1)$ would be at most $2^n \sum_{V\in\mathcal{V}} |V| \leq 2^{n+2n/5}m = o(2^{2n})$. We can then use Theorem 23 to get a lower bound of $\beta n/5$ when $\epsilon < \frac{(1-\alpha)^2}{4}\frac{m-2kh}{8m}\frac{|F^{-1}(0)|}{2^{2n}}$, which is a constant. Since we can use error reduction to go from error $1/10$ to any small constant error with only a constant blow-up in cost, the $1/10$-error randomized communication complexity is also $\Omega(n)$.

The ϵ-approximate rank of $f \circ \mathsf{XOR}$ is known to be at most the ϵ-approximate sparsity of f. As analyzed in Theorem 20, $\left\|\widehat{f_{\mathcal{V}}}\right\|_1 \leq 2m$ and $\|\hat{f_{\mathcal{V}}}\|_{0,1/10} \leq O(m^2 n) = O(n^3)$ and hence $\mathsf{rank}_{1/10}(F) \leq O(n^3)$. ◀

The existence of a dual subspace design as required in the previous corollary follows by changing Theorem 20 to set $m = 200kn$. The proof of the modified statement is syntactically identical to the proof of the original statement.

4 Conclusion and open problems

We come up with new and improved refutations of the query complexity analogue of the log-approximate-rank conjecture, following the work of Chattopadhyay, Mande and Sherif [6]. Our examples are derived from subspace designs, a concept which has previously found applications in coding theory and pseudorandomness [15, 14, 16]. A lot of interesting open problems arise from our work, some of which we mention below.

1. **(Communication complexity of XOR composed subspace designs).** What is the randomized communication complexity of dual subspace designs composed with XOR (as studied in Section 3.2)? A lower bound would follow from Conjecture 21. If Conjecture 21 is false, is there an alternate way to prove the communication lower bound? Since we already have an RPDT lower bound for dual subspace designs, these functions provide a interesting class of functions to study randomized XOR lifting. Currently we cannot even prove that this class of functions do not have large monochromatic rectangles.
2. **(Communication complexity of XOR composed random subspaces).** What is the randomized communication complexity of random subspaces composed with XOR? A lower bound would follow from Conjecture 22 which follows from Conjecture 21. Even if Conjecture 21 is false, Conjecture 22 could still be true or perhaps easier to prove. If even Conjecture 22 is false, is there an alternate way to prove the communication lower bound, perhaps adapting the technique of [17] to the randomized communication setting? Here also we cannot prove that there are no large monochromatic rectangles.

3. **(Quantum communication complexity of XOR composed subspace designs).** What is the quantum communication complexity of dual subspace designs composed with XOR? Is there a function in this class which has polylogarithmic quantum communication complexity?
4. **(RPDT and approximate sparsity).** What is the optimal gap between RPDT complexity and approximate sparsity? We give examples where the RPDT complexity is at least cube root of the approximate sparsity and also RPDT complexity is easily seen to be at most the approximate sparsity.

References

1 Anurag Anshu, Naresh Goud Boddu, and Dave Touchette. Quantum log-approximate-rank conjecture is also false. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 982–994. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00063.

2 Jehoshua Bruck and Roman Smolensky. Polynomial threshold functions, AC^0 functions and spectral norms (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 632–641, 1990.

3 Harry Buhrman and Ronald de Wolf. Communication complexity lower bounds by polynomials. In *Proceedings of the 16th Annual Conference on Computational Complexity*, CCC '01, page 120, USA, 2001. IEEE Computer Society.

4 Arkadev Chattopadhyay, Rajit Datta, and Partha Mukhopadhyay. Lower bounds for monotone arithmetic circuits via communication complexity. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 786–799. ACM, 2021.

5 Arkadev Chattopadhyay, Michal Koucký, Bruno Loff, and Sagnik Mukhopadhyay. Simulation beats richness: new data-structure lower bounds. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1013–1020. ACM, 2018.

6 Arkadev Chattopadhyay, Nikhil S. Mande, and Suhail Sherif. The log-approximate-rank conjecture is false. *J. ACM*, 67(4), June 2020. doi:10.1145/3396695.

7 Michael Christ. The optimal constants in Holder-Brascamp-Lieb inequalities for discrete Abelian groups. *arXiv preprint*, 2013. arXiv:1307.8442.

8 Susanna F. de Rezende, Or Meir, Jakob Nordström, Toniann Pitassi, Robert Robere, and Marc Vinyals. Lifting with simple gadgets and applications to circuit and proof complexity. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:186, 2019.

9 Anna Gál and Ridwan Syed. Upper bounds on communication in terms of approximate rank. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:6, 2019.

10 Dmitry Gavinsky and Shachar Lovett. En route to the log-rank conjecture: New reductions and equivalent formulations. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 514–524, 2014.

11 Mika Göös, Rahul Jain, and Thomas Watson. Extension complexity of independent set polytopes. *SIAM J. Comput.*, 47(1):241–269, 2018. doi:10.1137/16M109884X.

12 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM J. Comput.*, 47(6):2435–2450, 2018. doi:10.1137/16M1059369.

13 Vince Grolmusz. On the power of circuits with gates of low ℓ_1 norms. *Theor. Comput. Sci.*, 188(1-2):117–128, 1997.

14 Venkatesan Guruswami and Swastik Kopparty. Explicit subspace designs. *Combinatorica*, 36(2):161–185, 2016.

15 Venkatesan Guruswami and Chaoping Xing. List decoding reed-solomon, algebraic-geometric, and gabidulin subcodes up to the singleton bound. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 843–852. ACM, 2013.

16 Venkatesan Guruswami, Chaoping Xing, and Chen Yuan. Subspace designs based on algebraic function fields. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 86:1–86:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

17 Hamed Hatami, Kaave Hosseini, and Shachar Lovett. Structure of protocols for XOR functions. *SIAM J. Comput.*, 47(1):208–217, 2018.

18 Alexander Knop, Shachar Lovett, Sam McGuire, and Weiqiang Yuan. Log-rank and lifting for and-functions. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 197–208. ACM, 2021.

19 Eyal Kushilevitz and Noam Nisan. *Communication complexity.* Cambridge University Press, 1997.

20 Troy Lee and Adi Shraibman. Lower bounds in communication complexity. *Foundations and Trends in Theoretical Computer Science*, 3(4):263–398, 2009.

21 Shachar Lovett. Communication is bounded by root of rank. *J. ACM*, 63(1):1:1–1:9, 2016.

22 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 462–467, New York, NY, USA, 1992. Association for Computing Machinery. doi:10.1145/129712.129757.

23 Ryan O'Donnell. *Analysis of Boolean Functions.* Cambridge University Press, 2014.

24 R. Raz and P. McKenzie. Separation of the monotone NC hierarchy. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, page 234, USA, 1997. IEEE Computer Society.

25 Makrand Sinha and Ronald de Wolf. Exponential separation between quantum communication and logarithm of approximate rank. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 966–981. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00062.

26 Leslie G. Valiant. Negation can be exponentially powerful. *Theor. Comput. Sci.*, 12:303–314, 1980.

27 Andrew Chi-Chih Yao. Lower bounds by probabilistic arguments (extended abstract). In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 420–428. IEEE Computer Society, 1983. doi:10.1109/SFCS.1983.30.

28 Shengyu Zhang. Efficient quantum protocols for XOR functions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1878–1885, 2014.

Functional Lower Bounds for Restricted Arithmetic Circuits of Depth Four

Suryajith Chillara
CSTAR, International Institute of Information Technology, Hyderabad, India

Abstract

Recently, Forbes, Kumar and Saptharishi [CCC, 2016] proved that there exists an explicit $d^{O(1)}$-variate and degree d polynomial $P_d \in \mathsf{VNP}$ such that if any depth four circuit C of bounded formal degree d which computes a polynomial of bounded individual degree $O(1)$, that is functionally equivalent to P_d, then C must have size $2^{\Omega(\sqrt{d}\log d)}$.

The motivation for their work comes from Boolean Circuit Complexity. Based on a characterization for ACC^0 circuits by Yao [FOCS, 1985] and Beigel and Tarui [CC, 1994], Forbes, Kumar and Saptharishi [CCC, 2016] observed that functions in ACC^0 can also be computed by algebraic $\Sigma\wedge\Sigma\Pi$ circuits (i.e., circuits of the form – sums of powers of polynomials) of $2^{\log^{O(1)} n}$ size. Thus they argued that a $2^{\omega(\text{poly}\log n)}$ "functional" lower bound for an explicit polynomial Q against $\Sigma\wedge\Sigma\Pi$ circuits would imply a lower bound for the "corresponding Boolean function" of Q against non-uniform ACC^0. In their work, they ask if their lower bound be extended to $\Sigma\wedge\Sigma\Pi$ circuits.

In this paper, for large integers n and d such that $\omega(\log^2 n) \leq d \leq n^{0.01}$, we show that any $\Sigma\wedge\Sigma\Pi$ circuit of bounded individual degree at most $O\left(\frac{d}{k^2}\right)$ that functionally computes Iterated Matrix Multiplication polynomial $\mathsf{IMM}_{n,d}$ ($\in \mathsf{VP}$) over $\{0,1\}^{n^2 d}$ must have size $n^{\Omega(k)}$. Since Iterated Matrix Multiplication $\mathsf{IMM}_{n,d}$ over $\{0,1\}^{n^2 d}$ is functionally in GapL, improvement of the afore mentioned lower bound to hold for quasipolynomially large values of individual degree would imply a fine-grained separation of ACC^0 from GapL.

For the sake of completeness, we also show a syntactic size lower bound against any $\Sigma\wedge\Sigma\Pi$ circuit computing $\mathsf{IMM}_{n,d}$ (for the same regime of d) which is tight over large fields. Like Forbes, Kumar and Saptharishi [CCC, 2016], we too prove lower bounds against circuits of bounded formal degree which functionally compute $\mathsf{IMM}_{n,d}$, for a slightly larger range of individual degree.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Algebraic complexity theory

Keywords and phrases Functional Lower Bounds, Boolean Circuit Lower Bounds, Depth Four, Connections to Boolean Complexity, Iterated Matrix Multiplication

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.14

Related Version *Full Version*: https://eccc.weizmann.ac.il/report/2021/105/

Funding Research done while at University of Haifa and was supported in parts by (1) PBC fellowship from Israeli Council of Higher Education, (2) funding from University of Haifa, and (3) funding from Prof. Noga Ron-Zewi.

Acknowledgements The author is grateful to Nikhil Balaji, Mrinal Kumar, Noga Ron-Zewi, Nitin Saurabh, and Nithin Varma for helpful discussions. The author thanks Nikhil Balaji for telling him more about the Boolean complexity of Iterated Matrix Multiplication. The author thanks Ramprasad Saptharishi for patiently presenting the results in [14] while the author visited Tel Aviv University in 2016, hosted by Amir Shpilka.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 14; pp. 14:1–14:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Owing to the difficulty in proving Boolean circuit size lower bounds, Valiant proposed that we prove lower bounds in an "algebraic setting" as the underlying algebraic structure could help us understand the computations better. Valiant further conjectured that any circuit theoretic proof for $\mathsf{P} \neq \mathsf{NP}$ would have to be preceded by an analogous result in this more constrained arithmetic model [42].

Arithmetic circuits (also called as algebraic circuits) are directed acyclic graphs such that the leaf nodes are labeled by variables or constants from the underlying field, and every non-leaf node is labeled either by a + or ×. Every node computes a polynomial by operating on its inputs with the operation given by its label. The computation flows from the leaves to the output node. Complexity of computation here is quantified by the size of the circuit, which is the number of nodes in it.

It is conjectured that Permanent polynomial does not have polynomial size arithmetic circuits [41]. Bürgisser [6] showed that if Permanent polynomial were to have a polynomial sized arithmetic circuit then this would imply $\#\mathsf{P} \subseteq \mathsf{FNC}^3/\,\mathrm{poly}$ which would further imply that $\mathsf{NP} \subseteq \mathsf{P}/\,\mathrm{poly}$ which leads to (1) $\mathsf{PH} \subseteq \Sigma_p^2$ [20] and (2) $\mathsf{AM} = \mathsf{MA}$ [3], both of which go against widely believed conjectures. Thus, a central question in the field of algebraic complexity theory is to show that Permanent polynomial (or any closely related polynomial of interest) needs superpolynomial sized arithmetic circuits to compute it.

Four decades after the problem was formulated, the best known size lower bound is still super linear [4]. Over the span of last three decades, researchers have considered restricted arithmetic circuits and here we have seen a great progress towards proving lower bounds under these restrictions (see [40, 38] for a detailed survey). In a surprising result, Agrawal and Vinay [1] showed that it is sufficient to prove subexponential size lower bounds against depth four circuits, to prove super polynomial size lower bounds against general arithmetic circuits.

A depth four circuit[1] (denoted by $\Sigma\Pi\Sigma\Pi$) computes polynomials that can also be expressed as a sum of products of polynomials.

$$P(X) = \sum_i \prod_j Q_{i,j} \, .$$

Syntactic lower bounds

We say that a polynomial P has a syntactic circuit size lower bound of s against class $\mathcal{C}$ of circuits if no circuit in $\mathcal{C}$ of size strictly smaller than s syntactically computes P.

Strong syntactic size lower bounds for depth four circuits were proven in restricted settings: Bounded fan-in [17, 24, 15, 12, 27], Homogeneous [21, 26, 22, 28], Multilinear [36, 11], and Multi-r-ic [25, 8, 7]. In a breakthrough, Limaye, Srinivasan and Tavenas recently proved superpolynomial size lower bounds against all constant depth circuits [29]. Prior to that the best known lower bound for depth four circuits was super-quadratic [19] (which improves upon super-linear lower bounds due to Shoup and Smolensky [39] and Raz [34]).

[1] Generally speaking, a depth four circuit can also be of the form $\Pi\Sigma\Pi\Sigma$ but we follow the convention that the root node is a + node. Under such a convention $\Pi\Sigma\Pi\Sigma$ circuit is a depth five circuit.

Functional lower bounds

For a set $B \subseteq \mathbb{F}$, we say that two polynomials $P(x_1, \ldots, x_N)$ and $Q(x_1, \ldots, x_N)$ are functionally equivalent over B^N if $P(\mathbf{a}) = Q(\mathbf{a})$ for all $\mathbf{a} \in B^N$. We say that a circuit C functionally computes a polynomial $P \in \mathbb{F}[x_1, \ldots, x_N]$ over B^N if the output polynomial $f \in F[x_1, \ldots, x_N]$ of C is functionally equivalent to P over B^N.

We say that a polynomial P has a functional size lower bound of s against a class $\mathcal{C}$ of circuits if no polynomial that is computed by circuits in $\mathcal{C}$ of size strictly less than s, is functionally equivalent to P over B^n for some $B \subseteq \mathbb{F}$.

Forbes, Kumar and Saptharishi [14] proved exponential functional lower bounds for a polynomial in VNP against depth four circuits of bounded formal degree and bounded individual degree $O(1)$. Formally, they showed that there is an explicit polynomial P_d of degree d over $\approx d^3$ variables such that no depth four circuit of bounded formal degree d and size smaller than $2^{c(\sqrt{d}\log d)}$ (for a small constant c) that computes a polynomial of bounded individual degree at most $O(1)$ can be functionally equivalent to P_d. Apart from this work, strong functional lower bounds are known against depth three circuits over finite fields [16], multilinear formulas [33, 32, 35, 36, 10, 11], and set-multilinear formulas [31, 29].

The motivation for the work of [14] comes from Boolean circuit complexity. ACC^0 circuits are constant depth Boolean circuits that have AND, OR, NOT and MOD gates. Allender and Gore [2] showed that *uniform* ACC^0 circuits of subexponential size cannot compute Permanent. In a major breakthrough, Williams [44] showed that there exists a function in NEXP such that it cannot be computed by polynomial sized *nonuniform* ACC^0 circuits. Recently Murray and Williams [30] further improved the situation to show that there exists a function in NQP such that it needs superpolynomial size ACC^0 circuits to compute it.

Beigel and Tarui [5] showed that every language L in the class ACC^0 can be recognized by a family of depth two[2] deterministic circuits with a symmetric function gate at the root and $2^{\log^{O(1)} n}$ many AND gates of fan-in $\log^{O(1)} n$ in the second layer. Over large fields, Forbes, Kumar and Saptharishi [14] observed that given this Boolean circuit, there is an algebraic circuit of depth four which computes polynomials of the form – sum of $2^{\log^{O(1)} n}$ many powers of polynomials each of whose monomials are supported on at most $\log^{O(1)} n$ many variables such that outputs of both of these circuits are functionally equivalent.

$\Sigma\wedge\Sigma\Pi$ circuits are depth four circuits that compute polynomials which can be expressed as sums of powers of polynomials. $\Sigma\wedge\Sigma\Pi^{[t]}$ circuits are depth four circuits that compute polynomials which can be expressed as sums of powers of polynomials each of whose monomials are supported on at most t many variables.

We can summarize the afore mentioned discussion formally as follows.

▶ **Lemma 1** (Lemma 3.2, [14]). *Let $\mathbb{F}$ be any field of characteristic zero or at least $\exp(\omega(\mathrm{poly}(\log n)))$. If a function $f : \{0,1\}^n \mapsto \{0,1\}$ is in ACC^0 then there exists a polynomial $P_f \in \mathbb{F}[x_1, \ldots, x_n]$ such that*

- *P_f and f are functionally equivalent over $\{0,1\}^n$, and*
- *P_f can be computed by a $\Sigma\wedge\Sigma\Pi$ circuit of top fan-in at most $2^{\log^{O(1)} n}$ and bottom support at most $\log^{O(1)} n$.*

[2] Here the variables can appear negated at the leaves that feed into the AND gates. Even though it is stated as depth two in the paper, the longest leaf to root path in this circuit is of length 3. Leaf node → AND → root.

Thus, to show a lower bound against ACC^0 circuits in the Boolean setting, it is sufficient to show a functional lower bound of $\exp(\omega(\text{poly}(\log n)))$ for a polynomial P would imply that the Boolean part[3] of P is not in ACC^0.

▶ **Lemma 2** (Lemma 3.3, [14]). *Let $\mathbb{F}$ be any field of characteristic zero or at least $\exp(\omega(\text{poly}(\log n)))$. Then a $\exp(\omega(\text{poly}(\log n)))$ functional size lower bound for a $n^{O(1)}$-variate and $n^{O(1)}$ degree polynomial $P \in \mathbb{F}[X]$ against $\Sigma\wedge\Sigma\Pi^{[\text{poly}(\log(n))]}$ circuits over $\mathbb{F}$ would imply that Boolean part of P is not in ACC^0.*

Forbes, Kumar and Saptharishi [14] through an open question in their paper ask if such functional lower bounds can also be proved for $\Sigma\wedge\Sigma\Pi$ circuits. We in this paper show strong functional lower bounds against all $\Sigma\wedge\Sigma\Pi$ circuits which output polynomials of bounded individual degree.

A circuit C is said to have a bounded individual degree[4] r if the polynomial output by the circuit C has degree at most r with respect to each of its variables.

▶ **Theorem 3** (Functional Lower Bounds for $\Sigma\wedge\Sigma\Pi$ circuits of Bounded Individual Degree). *Let n be a large integer. Let d, k and r be such that $\omega(\log^2 n) \leq d \leq n^{0.01}$ and $r \leq \frac{d}{1201k^2}$. Any depth four $\Sigma\wedge\Sigma\Pi$ circuit of bounded individual degree r computing a function equivalent to $\mathsf{IMM}_{n,d}$ on $\{0,1\}^{n^2d}$, must have size at least $n^{\Omega(k)}$.*

Note that there is a trade-off between the lower bound on the circuit size and the upper bound on the range of r this lower bound can be achieved for.

Since Iterated Matrix Multiplication $\mathsf{IMM}_{n,d}$ over $\{0,1\}^{n^2d}$ is functionally[5] in GapL [43, Section 6], improvement of the afore mentioned lower bound to hold for quasipolynomially large values of individual degree would imply a fine-grained separation of ACC^0 from GapL.

By a divide and conquer construction, we get a depth four $\Sigma\Pi\Sigma\Pi$ circuit of size $n^{O(\sqrt{d})}$ that computes $\mathsf{IMM}_{n,d}$ such that the fan-in of both the product gates is equal to $\sqrt{d}$. Using the identity

$$m! \cdot x_1 x_2 \dots x_m = \sum_{S\subseteq[m]} \left(\sum_{i\in S} x_i\right)^m \cdot (-1)^{m-|S|}$$

(attributed to Fischer [13] and Ryser [37] in [18]), over large fields this circuit can be converted into a $\Sigma\wedge\Sigma\Pi$ circuit of size $n^{O(\sqrt{d})}$. We will now show a lower bound of $n^{\Omega(\sqrt{d})}$ for $\mathsf{IMM}_{n,d}$ against any $\Sigma\wedge\Sigma\Pi$ circuits. From the afore mentioned discussion, this lower bound is optimal up to a constant in the exponent over large fields.

▶ **Theorem 4** (Syntactic Lower Bounds for $\Sigma\wedge\Sigma\Pi$ circuits). *Let n and d be a large integers such that $\omega(\log^2 n) \leq d \leq n^{0.01}$. Any depth four $\Sigma\wedge\Sigma\Pi$ circuit computing $\mathsf{IMM}_{n,d}$ must have size at least $n^{\Omega(\sqrt{d})}$.*

[3] Bürgisser [6] defined the boolean part of a polynomial $P(x_1, \dots, x_n)$ (denoted by BP(P)) to be a function that agrees with P over all evaluations over $\{0,1\}^n$.

[4] Not to be confused with the multi-r-ic circuits dealt with in [23, 25, 8, 7].

[5] Bürgisser [6] showed that boolean part of any polynomial in VP lies in $\mathsf{FNC}^3/\text{poly}$, and in particular $\mathsf{IMM}_{n,d} \in \mathsf{VP}$. On the other hand, Vinay [43] identified that this problem of computing Iterated Matrix Product of integer matrices (denoted by ITMATPROD) is in fact in the class GapL which consists of all problems that are logspace reducible to determinant computation of an integer matrix. This is a better characterization as $\mathsf{GapL} \subseteq NC^2 \subseteq \mathsf{FNC}^3/\text{poly}$.

Proof of this lemma can be found in [9, Section 5]. Recall that Forbes, Kumar and Saptharishi [14] proved functional lower bounds for a polynomial in VNP against depth four circuits of bounded formal degree whose output polynomials are of bounded individual degree $O(1)$. Here shall prove functional lower bounds for a polynomial in VP against depth four circuits of bounded formal degree whose output polynomials are of bounded individual degree $O(\log n)$.

Formal degree of a circuit is the maximum degree of any polynomial that could be computed by this circuit structure sans the constants nor cancellations. Formal degree of a circuit is inductively defined as follows: for a leaf node w, the formal degree 1 if it is labeled by a variable and 0 otherwise. Formal degree of a sum node is the maximum over all the formal degrees of its children, and formal degree of a product node is equal to the sum over all the formal degrees of its children.

▶ **Theorem 5** (Functional Lower Bounds for $\Sigma\Pi\Sigma\Pi$ Circuits of Bounded Formal Degree)**.** *Let n, d and r be integers such that $\omega(\log^2 n) \leq d \leq n^{0.01}$ and $r \leq \frac{\log n}{12}$. Any depth four $\Sigma\Pi\Sigma\Pi$ circuit of formal degree d and bounded individual degree r that computes a function equivalent to $\mathsf{IMM}_{n,d}$ on $\{0,1\}^{n^2 d}$, must have size at least $n^{\Omega\left(\sqrt{\frac{d}{r}}\right)}$.*

Proof of this lemma can be found in [9, Section 6]. We would to remark that the afore mentioned bound and the bound for similar circuits in [14] can be made to work for formal degree that is slightly larger than d.

Related Work

For the sake of brevity, we shall denote the $\Sigma\wedge\Sigma\Pi$ circuits of bounded individual degree r by $(\Sigma\Pi\Sigma\Pi)^{\leq r}$. We in this table summarize our results in comparison to the work of [14].

Circuit model	Work	Hard multilinear polynomial family	Lower Bound	Range of parameters
$(\Sigma\Pi\Sigma\Pi)^{\leq r}$ & formal degree d	[14]	Nisan-Wigderson polynomial $\mathsf{NW}_{m,d} \in \mathsf{VNP}$ with md many variables and degree d	$2^{\Omega\left(\sqrt{d}\log(md)\right)}$	$m = \Theta(d^2)$, and $r \leq O(1)$.
$(\Sigma\Pi\Sigma\Pi)^{\leq r}$ & formal degree d	This work	Iterated Matrix Multiplication polynomial $\mathsf{IMM}_{n,d} \in \mathsf{VP}$ with $n^2 d$ many variables and degree d	$n^{\Omega\left(\sqrt{\frac{d}{r}}\right)}$	$\omega(\log^2 n) \leq d \leq n^{0.01}$, and $r \leq \frac{\log n}{12}$.
$(\Sigma\wedge\Sigma\Pi)^{\leq r}$	This work	$\mathsf{IMM}_{n,d}$	$n^{\Omega(k)}$	$\omega(\log^2 n) \leq d \leq n^{0.01}$, and $r \leq \frac{d}{1201k^2}$.

Our work is inspired by [14]'s line of research and depends on the techniques introduced by them. We take their research a bit further.

Complexity measure and proof overview

Let the variable set X be partitioned into two fixed, disjoint sets Y and Z. Let $\sigma_Y : \mathbb{F}[Y \sqcup Z] \mapsto \mathbb{F}[Z]$ be a linear map such that for any polynomial $P(Y, Z)$, $\sigma_Y(P) \in \mathbb{F}[Z]$ is obtained by setting every variable from Y to zero and leaving the variables from Z untouched.

For a polynomial $P(x_1, \ldots, x_N)$, let $\text{mult}(P)$ be defined to be equal to P mod $\left\{(x_i^2 - x_i) \mid i \in [N]\right\}$. Similarly, let $\text{mult}(V)$ for a subspace V of polynomials in $\subseteq \mathbb{F}[x_1, \ldots, x_N]$, be defined as follows.

$$\text{mult}(V) = \{\text{mult}(P) \mid P \in V\}.$$

For a polynomial $P(Y, Z)$ and a set $S \subseteq \mathbb{F}$, let $\text{Eval}_S^{[Y \cup Z]}(P)$ denote the vector of evaluations of polynomial P over $S^{|Y \cup Z|}$ as follows.

$$\text{Eval}_S^{[Y \cup Z]}(P(Y, Z)) = (P(\mathbf{a}))_{\mathbf{a} \in S^{|Y \cup Z|}}.$$

This definition can be extended to a set V of polynomials over $\mathbb{F}[Y \cup Z]$ as follows.

$$\text{Eval}_S^{[Y \cup Z]}(V) = \left\{\text{Eval}_S^{[Y \cup Z]}(P(Y, Z)) \mid P(Y, Z) \in V\right\}.$$

We use $\partial_Y^{\leq k} P$ to denote the set of all partial derivatives of P of order at most k with respect to monomials over variables just from Y, and $Z^{=\ell} \cdot \sigma_Y(\partial_Y^{=k} P)$ to refer to the set of polynomials obtained by multiplying each polynomial in $\sigma_Y(\partial_Y^{\leq k} P)$ with monomials of degree equal to ℓ in Z variables.

Main measure – Multilinear Shifted Evaluation Dimension ($\mathbf{mSED}_{k,\ell}^{[Y,Z]}$)

Forbes, Kumar and Saptharishi [14] defined Shifted Evaluation Dimension which counts the dimension of space of vectors each of which is a list of evaluations of polynomials $\{0,1\}^{|X|}$ where these polynomials are Z-shifts of partial evaluations.

$$\text{SED}_{k,\ell}^{[Y,Z]}(P(Y, Z)) = \dim\left(\text{Eval}_{\{0,1\}^{|Z|}}\left\{Z^{=\ell} \cdot \mathbb{F}\text{-span}\left\{P(\mathbf{a}, Z) \mid \mathbf{a} \in \{0,1\}_{\leq k}^{|Y|}\right\}\right\}\right)$$

We just make a minor modification to this measure to better relate our measure with the measure of Projected Shifted Skew Partial derivatives ([8, 7]) and this helps us obtain bounds that we could not get before.

$$\text{mSED}_{k,\ell}^{[Y,Z]}(P(Y, Z)) = \dim\left(\text{Eval}_{\{0,1\}^{|Z|}}\left\{\text{mult}\left(Z^{=\ell} \cdot \mathbb{F}\text{-span}\left\{P(\mathbf{a}, Z) \mid \mathbf{a} \in \{0,1\}_{\leq k}^{|Y|}\right\}\right)\right\}\right)$$

In spirit, it is still the measure of [14] and thus we do not consider this to be a new measure. We just make a minor modification to relate this measure with their measure of Projected Shifted Skew Partial derivatives ([8, 7]) and this helps us obtain bounds that we could not get before.

By unfurling the above definition, we can see that if two N-variate polynomials $P_1(Y, Z)$ and $P_2(Y, Z)$ (defined on the same variable sets) are functionally equivalent over $\{0,1\}^N$ then $\text{mSED}_{k,\ell}^{[Y,Z]}(P_1(Y, Z)) = \text{mSED}_{k,\ell}^{[Y,Z]}(P_2(Y, Z))$. Note that two polynomials which are not functionally equivalent over $\mathbb{F}^N$ can end up being functionally equivalent over $\{0,1\}^N$ but to show that two polynomials are not functionally equivalent, it is sufficient to show that they are not functionally equivalent over $\{0,1\}^N$.

The crux of our work henceforth is to show that the polynomial of interest, $\mathsf{IMM}_{n,d}$ is not functionally equivalent over $\{0,1\}^{n^2 d}$ to the polynomials that are output by the $\Sigma\wedge\Sigma\Pi$ circuits of bounded individual degree. That is, we need to show that $\text{mSED}_{k,\ell}^{[Y,Z]}(\mathsf{IMM}_{n,d}(Y, Z))$ is much larger than $\text{mSED}_{k,\ell}^{[Y,Z]}(C(Y, Z))$ where C is a $\Sigma\wedge\Sigma\Pi$ circuit of small size and bounded individual degree.

Though two N-variate polynomials P_1 and P_2 that are functionally equivalent over $\{0,1\}^N$ have the same (multilinear) shifted evaluation dimension, the dimension of their partial derivative spaces can be very different (see [14, Section 1.2.1] for an example). However in certain special cases Forbes, Kumar and Saptharishi [14] do manage to relate the shifted evaluation dimension, and a partial derivate based measure well enough for their proof to work. We shall do something very similar.

Let C be a $\Sigma\wedge\Sigma\Pi$ circuit of bounded individual degree at most r that computes a polynomial that is functionally equivalent to a homogeneous and degree d set-multilinear polynomial $P(X)$ defined over the sets $X = X_1 \sqcup \ldots \sqcup X_d$ such that $Y = X_{i_1} \sqcup \ldots X_{i_k}$ (for a fixed subset $\{i_1, \ldots, i_k\} \subseteq [d]$) and $Z = X \setminus Y$. Similar to [14], we show that we can bound the multilinear shifted evaluation dimension on the above and below by an auxiliary measure that counts the dimension of a space of a specially chosen syntactic polynomials. For every value of k, ℓ and r, we can show that

$$\mathrm{PSSPD}^{[Y,Z]}_{k,\ell}(P(Y,Z)) \leq \mathrm{mSED}^{[Y,Z]}_{k,\ell}(P(Y,Z)) = \mathrm{mSED}^{[Y,Z]}_{k,\ell}(C(Y,Z)) \leq \mathrm{PSSPD}^{[Y,Z]}_{rk,\ell}(C(Y,Z)) \,.$$

Upon instantiating the above expression with explicit homogeneous and set-multilinear polynomial $\mathsf{IMM}_{n,d}(Y,Z)$, and if for a suitable setting of values of k, ℓ and r, we get that $\mathrm{PSSPD}^{[Y,Z]}_{k,\ell}(\mathsf{IMM}_{n,d}(Y,Z))$ is much larger than $\mathrm{PSSPD}^{[Y,Z]}_{rk,\ell}(C(Y,Z))$ where C is a $\Sigma\wedge\Sigma\Pi$ circuit that computes polynomials of bounded individual degree r of size s, then we can infer that $\mathsf{IMM}_{n,d}(Y,Z)$ cannot be functionally computed by this class of circuits, thus giving us a functional size lower bound of s for this explicit polynomial.

Auxiliary measure – Projected Skew Shifted Partial Derivatives ($\mathrm{PSSPD}^{[\boldsymbol{Y},\boldsymbol{Z}]}_{\boldsymbol{k},\boldsymbol{\ell}}$)

The following is a measure[6] borrowed from [7] which was used to prove syntactic lower bounds for multi-r-ic depth four circuits.

$$\mathrm{PSSPD}^{[Y,Z]}_{k,\ell}(P(Y,Z)) = \dim\left(\mathbb{F}\text{-span}\left\{\mathrm{mult}\left(Z^{=\ell} \cdot \sigma_Y\left(\partial^{\leq k}_{Y} P\right)\right)\right\}\right) .$$

We currently do not know how to directly obtain a bound on $\mathrm{PSSPD}^{[Y,Z]}_{rk,\ell}(C(Y,Z))$ to a value that is much smaller than $\mathrm{PSSPD}^{[Y,Z]}_{k,\ell}(\mathsf{IMM}_{n,d}(Y,Z))$. To resolve this issue, we use random restrictions $V \leftarrow D$ to convert our $\Sigma\wedge\Sigma\Pi$ circuit C of size $s \leq n^{\frac{t}{2}}$ that computes a polynomial P of bounded individual degree to a $\Sigma\wedge\Sigma\Pi$ circuit C' of size s and of bottom fan-in at most t that still computes the restricted polynomial P', with a high probability. We can now bound $\mathrm{PSSPD}^{[Y,Z]}_{rk,\ell}(C(Y,Z))$ to a value that is much smaller than $\mathrm{PSSPD}^{[Y,Z]}_{k,\ell}((\mathsf{IMM}_{n,d}(Y,Z))|_V)$. This trick is omnipresent in this line of work [21, 26, 22, 28, 25, 14, 8, 7].

We then borrow the lower bound on $\mathrm{PSSPD}^{[Y,Z]}_{k,\ell}(P'(Y,Z))$ (where P' is the polynomial obtained from $\mathsf{IMM}_{n,d}$ after restrictions) from [7].

We would like to remark that $\mathrm{mult}(P)$ for a polynomial $P(x_1,\ldots,x_N)$ was defined to be $P \mod \left\{x_i^2 : i \in [N]\right\}$ in [8, 7] instead of $P \mod \left\{x_i^2 - x_i : i \in [N]\right\}$ as defined here. We use this new definition of mult because $\mathrm{mSED}^{[Y,Z]}_{k,\ell}(P_1(Y,Z))$ may not be equal to $\mathrm{mSED}^{[Y,Z]}_{k,\ell}(P_2(Y,Z))$ under the older definition of $\mathrm{mult}(P) = P \mod \left\{x_i^2 : i \in [N]\right\}$ even though $P_1(Y,Z)$ and $P_2(Y,Z)$ are functionally equivalent.

The lower bound on $\mathrm{PSSPD}^{[Y,Z]}_{k,\ell}(P'(Y,Z))$ in [7] continues to hold despite this change of definition.

[6] This measure is an amalgamation of measures – dimension of Projected Shifted Partial derivatives of [21] and dimension of Skew Shifted Partial derivatives of [25].

2 Preliminaries

Notation

- We use $[n]$ to refer to the set $\{1, 2, \ldots, n\}$.
- For a polynomial f and a monomial m of degree k, we use $\partial_m^k f$ to refer to the kth partial derivate of the polynomial f with respect to the monomial m.
- For a polynomial f, we use $\partial_Y^{\leq k}(f)$ to refer to the space of partial derivatives of order at most k of f with respect to monomials of degree at most k in variables from Y.
- We use $Z^{=\ell}$ and $Z^{\leq \ell}$ to refer to the set of all the monomials of degree equal to ℓ and at most ℓ, respectively, in variables Z.
- We use $Z_{\mathrm{ML}}^{\leq t}$ to refer to the set of all the multilinear monomials of degree at most t in Z variables.
- For sets A and B of polynomials, we define the product $A \cdot B$ to be the set $\{f \cdot g \mid f \in A \text{ and } g \in B\}$.
- For a monomial m we use $\mathrm{Supp}(m)$ to refer to the set of variables that appear in it.
- We use $Z_{\{\leq t\}}$ to refer to the set of all monomials m in Z variables such that $|\mathrm{Supp}(m)| \leq t$.

▷ Claim 6. Let $W \subseteq \mathbb{F}[X]$ be a subspace of multilinear polynomials. Then $\dim(W) = \dim(\mathrm{Eval}_{\{0,1\}}^{[X]}(W))$.

Proof. Proof of this claim follows from the facts that every multilinear polynomial in W has a unique evaluation vector, and access to evaluations of a multilinear polynomial over all of $\{0,1\}^{|X|}$ uniquely determines it. ◁

▶ **Proposition 7.** *For two sets A and B of polynomials,*

1. $\mathrm{mult}(A \cdot B) = \mathrm{mult}(\mathrm{mult}(A) \cdot \mathrm{mult}(B))$*, and*
2. $\dim(\mathrm{mult}(\mathrm{mult}(A) \cdot \mathrm{mult}(B))) \leq \dim(\mathrm{mult}(A) \cdot \mathrm{mult}(B))$.

The proof of this proposition easily follows from the fact that mult is a many to one map and not one to many.

▶ **Definition 8** (Homogeneous polynomials). *A polynomial P of degree d is said to be homogeneous if it can be expressed as a linear combination of just the monomials of degree equal to d.*

▶ **Definition 9** (Set-multilinear polynomials). *A polynomial P is said to be set-multilinear with respect to a set of variables X, under the partition $X = X_1 \sqcup X_2 \sqcup \ldots X_d$ if every monomial m in the monomial support of P is such that $|\mathrm{MonSupp}(m) \cap X_i| \leq 1$ for all $i \in [d]$.*

▶ **Definition 10** (Multi-r-ic polynomials). *A polynomial P is said to be multi-r-ic polynomial if the degree of the polynomial with respect to each of its variables is at most r.*

The following lemma (from [17]) is key to the asymptotic estimates required for the lower bound analyses.

▶ **Lemma 11** (Lemma 6, [17]). *Let $a(n), f(n), g(n) : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be integer valued functions such that $(f+g) = o(a)$. Then,*

$$\ln \frac{(a+f)!}{(a-g)!} = (f+g)\ln a \pm O\left(\frac{(f+g)^2}{a}\right)$$

We shall now state a few lemmas that help us relate both the complexity measures introduced above.

▶ **Lemma 12** (Observation 4.5 in [14]). *Let $X = X_1 \sqcup \ldots \sqcup X_d$ and $|X| = N$. Let $Y = X_1 \sqcup \ldots \sqcup X_k$ for some $k \ll d$. Let P be a homogeneous set multilinear polynomial of degree d with respect to the partition $X_1 \sqcup \ldots \sqcup X_d$. Let $m = Y^{\mathbf{e}}$ be a set multilinear monomial*[7] *of degree k over Y. Then,*

$$\frac{\partial^k P}{\partial Y^{\mathbf{e}}} = P(\mathbf{e}, Z).$$

The following corollary can be obtained from Lemma 12 and proof of this corollary can be found in [9].

▶ **Corollary 13** (Similar to Corollary 4.6 in [14]). *For a homogeneous and set multilinear polynomial $P(Y, Z)$ which is as defined as in Lemma 12, and for all values of parameters k and ℓ,*

$$\mathrm{PSSPD}_{k,\ell}^{[Y,Z]}(P(Y,Z)) \leq \mathrm{mSED}_{k,\ell}^{[Y,Z]}(P(Y,Z))\,.$$

▶ **Lemma 14** (Lemma 4.7 in [14]). *Let $P(Y, Z)$ be a multi-r-ic polynomial. Then for every choice of parameters k and ℓ, we have*

$$\left\{P(\mathbf{e}, Z) \mid \mathbf{e} \in \{0,1\}^{|Y|}_{\leq k}\right\} \subseteq \mathbb{F}\text{-}span\left\{\sigma_Y(\partial_Y^{\leq rk} P)\right\}.$$

The following corollary can be obtained from Lemma 14. and proof of this corollary can be found in [9].

▶ **Corollary 15** (Similar to Lemma 4.8 in [14]). *For a multi-r-ic polynomial $P(Y, Z)$,*

$$\mathrm{mSED}_{k,\ell}^{[Y,Z]}(P(Y,Z)) \leq \mathrm{PSSPD}_{rk,\ell}^{[Y,Z]}(P(Y,Z)).$$

Complexity measure for the $\Sigma\wedge\Sigma\Pi$ circuits of low bottom support

The arguments from [8] can be adapted to get the following lemma. and its proof can be found in [9].

▶ **Lemma 16.** *Let m, k, ℓ and t be positive integers such that $\ell + kt < \frac{m}{2}$. Let Y and Z be disjoint sets of variables such that $|Z| = m$. Let $C(Y, Z)$ be a depth four $\Sigma\wedge\Sigma\Pi$ circuit of bottom support at most t with respect to variables from Z, and size s. Then, $\mathrm{PSSPD}_{k,\ell}^{[Y,Z]}(C)$ is at most $s \cdot (k+1) \cdot \binom{m}{\ell+kt} \cdot (\ell + kt)$.*

3 Hard Polynomial and Restrictions

In this section we recall the definition of the polynomial family and the set of deterministic and random restrictions imposed on the polynomial family, from [7].

3.1 Polynomial Family: Iterated Matrix Multiplication polynomial

Let $X^{(1)}, X^{(2)}, \ldots, X^{(d)}$ be d generic $n \times n$ matrices defined over disjoint set of variables. For any $k \in [d]$, let $x_{i,j}^{(k)}$ be the variable in the matrix $X^{(k)}$ indexed by $(i, j) \in [n] \times [n]$. The Iterated Matrix Multiplication polynomial, denoted by the family $\{\mathsf{IMM}_{n,d}\}$, is defined as follows.

$$\mathsf{IMM}_{n,d}(X) = \sum_{i_1, i_2, \ldots, i_{d-1} \in [n]} x_{1,i_1}^{(1)} x_{i_1,i_2}^{(2)} \ldots x_{i_{(d-2)},i_{(d-1)}}^{(d-1)} x_{i_{(d-1)},1}^{(d)}.$$

[7] Here $\mathbf{e}$ is a $|Y|$-long vector that indicates the support of multilinear monomials. $Y^{\mathbf{e}}$ is a shorthand representation of $y_1^{e_1} y_2^{e_2} \ldots y_{|Y|}^{e_{|Y|}}$.

3.2 Deterministic and Random Restrictions

Let k and α be a parameters such that $d = (2\alpha + 3) \cdot k$. Let the d matrices be divided into k contiguous blocks of matrices $B_1, B_2, \ldots, B_k$ such that each block B_i contains $2\alpha + 3$ matrices. By suitable renaming, let us assume that each block B_i contains the following matrices.

$$X^{(i,L,\alpha+1)}, \cdots, X^{(i,L,2)}, X^{(i,L,1)}, X^{(i)}, X^{(i,R,1)}, X^{(i,R,2)}, \cdots, X^{(i,R,\alpha+1)}.$$

Let us first consider the following set of restrictions, first deterministic and then randomized.

Deterministic Restrictions

Let $V_0 : X \mapsto Y_0 \sqcup Z_0 \sqcup \{0,1\}$ be a deterministic restriction of the variables X in to disjoint variable sets Y_0, Z_0, and $\{0,1\}$ as follows. For all $i \in [k]$,

- The variables in matrix in $X^{(i)}$ are each set to a distinct Y_0 variable. Henceforth, we shall refer to this as $Y^{(i)}$ matrix.
- The entries of the first row of matrix $X^{(i,L,\alpha+1)}$ are all set to 1 and the rest of the matrix to 0.
- The entries of the first column of matrix $X^{(i,R,\alpha+1)}$ are all set to 1 and the rest of the matrix to 0.
- The rest of the variables are all set to distinct Z_0 variables. Henceforth, for all $b \in \{L, R\}$ and $j \in [\alpha]$, we shall refer to the matrix $X^{(i,b,j)}$ as $Z^{(i,b,j)}$ matrix.

Random Restrictions

Let η and ε' be two fixed constants in $(0,1)$. Let $V_1 : Y_0 \sqcup Z_0 \mapsto Y \sqcup Z \sqcup \{0,1\}$ be a random restriction of the variables $Y_0 \sqcup Z_0$ as follows.

- Matrix $Z^{(i,L,1)}$: For every column, pick n^η distinct elements uniformly at random and keep these elements alive. Set the other entries in this matrix to zero.
- Matrix $Z^{(i,R,1)}$: For every row, pick n^η distinct elements uniformly at random and keep these elements alive. Set the other entries in this matrix to zero.
- Matrices $Z^{(i,L,j)}$ for all $j \in [2, \alpha - \varepsilon' \log n]$: For every column, pick 2 distinct elements uniformly at random and set all the other entries to zero.
- Matrices $Z^{(i,R,j)}$ for all $j \in [2, \alpha - \varepsilon' \log n]$: For every row, pick 2 distinct elements uniformly at random and set all the other entries to zero.
- Matrices $Z^{(i,L,j)}$ for all $j > \alpha - \varepsilon' \log n$: For every column, pick 1 element uniformly at random and set the other elements in that row to zero.
- Matrices $Z^{(i,R,j)}$ for all $j > \alpha - \varepsilon' \log n$: For every row, pick 1 element uniformly at random and set the other elements in that row to zero.

Let D be the distribution of all the restrictions $V : X \mapsto Y \sqcup Z \sqcup \{0,1\}$ such that $V = V_1 \circ V_0$ where V_0 and V_1 are deterministic and random restrictions respectively, as described above. Let m be used to denote the number of Z variables left after the restriction and $m = 2kn(n^\eta + 2(\alpha - \varepsilon' \log n - 1) + \varepsilon' \log n) = O(n^{1+\eta} k)$ when $\alpha \leq O(n^\eta)$.

Effect of Restrictions on $\mathsf{IMM}_{n,d}$

Let $g^{(i,L)}_{1,a}(Z)$ be the $(1,a)$th entry in product of matrices $\prod_{j=0}^{\alpha} X^{(i,L,\alpha+1-j)}|_V$. Let $g^{(i,R)}_{b,1}(Z)$ be the $(b,1)$th entry in product of matrices $\prod_{j=1}^{\alpha+1} X^{(i,R,j)}|_V$. Let $g^{(i)}$ the $(1,1)$th entry in the product of all the matrices in the block B_i. Then we can express $g^{(i)}$ as follows.

$$g^{(i)}(Y,Z) = \sum_{a,b\in[n]} g^{(i,L)}_{1,a}(Z) \cdot y^{(i)}_{a,b} \cdot g^{(i,R)}_{b,1}(Z).$$

Let $P|_V(Y,Z)$ obtained by restricting $\mathsf{IMM}_{n,d}(X)$ with the restriction $V \leftarrow D$. Thus,

$$P|_V(Y,Z) = \prod_{i=1}^{k} g^{(i)}(Y,Z)\,.$$

To summarize, for some parameters α, k, η and m, $P|_V$ is polynomial in $\mathbb{F}[Y \sqcup Z]$ such that its degree is $d = (2\alpha+3)\cdot k$, and has $m = O(n^{1+\eta}k)$ many Z variables. Here the definition of the polynomial $P|_V$ is heavily dependent on $V \leftarrow D$ and the choice of parameters α, k, ε' and η.

Effect on random restrictions

▶ **Lemma 17** (Lemma 8, [7]). *Let t be a parameter. Let C be any depth four circuit of size at most $s \leq n^{\frac{t}{2}}$ that computes $\mathsf{IMM}_{n,d}$. Then with a probability of at least $1-o(1)$, over $V \leftarrow D$ (where $V : X \mapsto Y \sqcup Z \sqcup \{0,1\}$), $C|_V$ is a depth four circuit of bottom support at most t in Z variables that computes the polynomial $P|_V(Y,Z)$.*

3.3 Complexity of $P|_V$

Choice of parameters

We borrow the setting of the parameters involved directly from [7][8].

- $\varepsilon' = 0.34$,
- $\eta = 0.05$,
- $\varepsilon = \varepsilon' - \eta = 0.29$,
- $\tau = 0.08$,
- $\omega(\log n) \leq d \leq n^{0.01}$,
- $d = (2\alpha+3)k$,
- $m = \Theta(n^{1+\eta}k) = \Theta(n^{1.05}k)$,
- $\ell = \frac{m}{2}(1-\Gamma)$,
- $(1+\Gamma)^{\alpha} = 2n^{\varepsilon}$ such that $\Gamma = O_{\varepsilon}\left(\frac{\ln n}{\alpha}\right)$,

We shall now recall the following from [7].

▶ **Theorem 18** (Discussion above Theorem 17, [7]). *Let n be a large enough integer. Let $m, d, \ell, \alpha, k, \varepsilon$ and τ be as described above.*

$$\mathrm{PSSPD}^{[Y,Z]}_{k,\ell}(P|_V) \geq \frac{\left(\frac{m}{m-\ell}\right)^{2\alpha k} \cdot \binom{m-2\alpha k}{\ell}}{2^{O(k)} \cdot \left(\frac{\ell}{m-\ell}\right)^{2\alpha k(1-\tau)}}.$$

[8] In an attempt to have a clean up the notation in comparison to [7], we make the following notational changes – the parameter α here corresponds to k' in [7], the parameter k here corresponds to r' in [7]. Further the parameter $k = d - 3r' = 2k'r'$ in [7] translates to $2\alpha k$ here. The rest of the parameters $\varepsilon, \varepsilon', \eta$ and τ are the same in both the papers.

Note that for a N-variate polynomial $P(X,Y)$, the measure in [7] was defined to be equal to $\dim\left(\mathbb{F}\text{-span}\left\{\text{mult}_0\left(Z^{=\ell}\cdot\sigma_Y\left(\partial_Y^{=k}P\right)\right)\right\}\right)$ where $\text{mult}_0(P) = P \mod \left\{x_i^2 \mid i\in[N]\right\}$ compared to the measure here which is equal to $\dim\left(\mathbb{F}\text{-span}\left\{\text{mult}\left(Z^{=\ell}\cdot\sigma_Y\left(\partial_Y^{\leq k}P\right)\right)\right\}\right)$ where $\text{mult}(P) = P \mod \left\{x_i^2 - x_i \mid i\in[N]\right\}$. This change of definition would not affect the bound as the lower bound in [7] counts the leading monomials of support size and degree both equal to $d-k+\ell$, and $\sigma_Y(\partial^{<k}P|_V) = \emptyset$ for the polynomial $P|_V$ described above.

4 Functional Lower Bounds against restricted $\Sigma\wedge\Sigma\Pi$ Circuits

As mentioned in the proof overview, we first prove a lower bound against bounded bottom support depth four circuits and then escalate this lower bound to circuits without the restriction on bottom support.

▶ **Lemma 19.** *Let n and d be large integers such that $\omega(\log^2 n)\leq d\leq n^{0.01}$. Let α,k,r and t be parameters such that $d=(2\alpha+3)k$ and $r\leq\frac{\alpha}{200t}$. Any depth four $\Sigma\wedge\Sigma\Pi$ circuit of bounded individual degree r and bounded bottom fan-in at most t, computing a function equivalent to $P|_V(X_V)$ (for $V\leftarrow D$) on $\{0,1\}^{|X_V|}$, must have size at least $n^{\Omega(k)}$.*

Proof of this lemma can be found in [9]. Using this lemma, we shall prove Theorem 3.

Proof of Theorem 3

For a large integer n, let d be such that $\omega(\log^2 n)\leq d\leq n^{0.01}$. Let t be a parameter that we shall soon fix. Let C be a $\Sigma\wedge\Sigma\Pi$ circuit of bounded individual degree at most r, and size $s\leq n^{\frac{t}{2}}$ that computes a polynomial $Q(X)$ that is functionally equivalent to $\mathsf{IMM}_{n,d}(X)$ (over $\{0,1\}^{n^2d}$). Let α and k be parameters such that $d=(2\alpha+3)k$. Recall that a restriction $V\leftarrow D$ fixes a subset of variables to values in $\{0,1\}$ and maps the rest to distinct Y and Z variables. For any such restriction $V\leftarrow D$, let $X_V = Y\sqcup Z$ be the set of variables in X that are not set to values in $\{0,1\}$ by V. From Lemma 17 we know that with a probability of at least $1-o(1)$, the circuit C_V obtained by applying the restriction V to C is a $\Sigma\wedge\Sigma\Pi$ circuit of bounded individual degree at most r, size s and bottom support at most t. Let Q_V be the polynomial computed by C_V, over X_V variables. We shall now show that Q_V is functionally equivalent to $P|_V$ over $\{0,1\}^{|X_V|}$.

Let the set $S_V\subset\{0,1\}^{n^2d}$ be the subset of points such that for all $\mathbf{a}\in S_V$, if $x_i\in X\setminus X_V$ and V sets x_i to $b\in\{0,1\}$, then the value at the i'th location of $\mathbf{a}$, $\mathbf{a}_i=b$. Since $Q(X)$ and $\mathsf{IMM}_{n,d}(X)$ are functionally equivalent over all of $\{0,1\}^{n^2d}$, they are functionally equivalent over S_V as well. Thus, $Q_V(\mathbf{a}|_{X_V}) = Q(\mathbf{a}) = \mathsf{IMM}(\mathbf{a}) = P|_V(\mathbf{a}|_{X_V})$ for all $\mathbf{a}\in S_V$. Here $\mathbf{a}|_{X_V}\in\{0,1\}^{|X_V|}$ corresponds to projection of $\mathbf{a}\in\{0,1\}^{n^2d}$ to locations corresponding to the variables in X_V.

This implies that $Q_V(X_V)$ and $P|_V(X_V)$ are functionally equivalent over $\{0,1\}^{|X_V|}$ and thus, there is a $\Sigma\wedge\Sigma\Pi$ circuit of bounded individual degree at most r, size $s\leq n^{\frac{t}{2}}$ and bottom support at most t that functionally computes $P|_V(Y,Z)$. On the other hand if r is at most $\frac{\alpha}{200t}$ then from Lemma 19 we know that any $\Sigma\wedge\Sigma\Pi$ circuit of bounded individual degree at most r and bottom support at most t that functionally computes $P|_V$ must have size $n^{\Omega(k)}$. Putting these together by fixing the value of t to $3k$ we get that s must at least be $n^{\Omega(k)}$. Since r is at most $\frac{\alpha}{200t}$, under this substitution of t, this value computes to $\frac{1}{200\cdot 3k}\cdot\left(\frac{d}{2k}-\frac{3}{2}\right) = \frac{d}{1200k^2}-\frac{1}{400k}$. ◀

References

1 Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 67–75. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.32.

2 Eric Allender and Vivek Gore. A uniform circuit lower bound for the permanent. *SIAM J. Comput.*, 23(5):1026–1049, 1994. doi:10.1137/S0097539792233907.

3 Vikraman Arvind, Johannes Köbler, Uwe Schöning, and Rainer Schuler. If NP has polynomial-size circuits, then MA=AM. *Theor. Comput. Sci.*, 137(2):279–282, 1995. doi:10.1016/0304-3975(95)91133-B.

4 Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983. doi:10.1016/0304-3975(83)90110-X.

5 Richard Beigel and Jun Tarui. On ACC. *Comput. Complex.*, 4:350–366, 1994. doi:10.1007/BF01263423.

6 Peter Bürgisser. Cook's versus valiant's hypothesis. *Theoretical Computer Science*, 235(1):71 – 88, 2000. URL: http://www.sciencedirect.com/science/article/pii/S0304397599001838, doi:https://doi.org/10.1016/S0304-3975(99)00183-8.

7 Suryajith Chillara. New exponential size lower bounds against depth four circuits of bounded individual degree. *Electronic Colloquium on Computational Complexity (ECCC)*, 27:33, 2020. URL: https://eccc.weizmann.ac.il/report/2020/033.

8 Suryajith Chillara. On Computing Multilinear Polynomials Using Multi-r-ic Depth Four Circuits. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2020.47.

9 Suryajith Chillara. Functional lower bounds for restricted arithmetic circuits of depth four. *Electron. Colloquium Comput. Complex.*, page 105, 2021. URL: https://eccc.weizmann.ac.il/report/2021/105.

10 Suryajith Chillara, Christian Engels, Nutan Limaye, and Srikanth Srinivasan. A near-optimal depth-hierarchy theorem for small-depth multilinear circuits. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 934–945. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00092.

11 Suryajith Chillara, Nutan Limaye, and Srikanth Srinivasan. Small-depth multilinear formula lower bounds for iterated matrix multiplication with applications. *SIAM J. Comput.*, 48(1):70–92, 2019. doi:10.1137/18M1191567.

12 Suryajith Chillara and Partha Mukhopadhyay. Depth-4 lower bounds, determinantal complexity: A unified approach. *computational complexity*, May 2019. doi:10.1007/s00037-019-00185-4.

13 Ismor Fischer. Sums of like powers of multivariate linear forms. *Mathematics Magazine*, 67(1):59–61, 1994. doi:10.1080/0025570X.1994.11996185.

14 Michael A. Forbes, Mrinal Kumar, and Ramprasad Saptharishi. Functional lower bounds for arithmetic circuits and connections to boolean circuit complexity. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 33:1–33:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CCC.2016.33.

15 Hervé Fournier, Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower bounds for depth-4 formulas computing iterated matrix multiplication. *SIAM J. Comput.*, 44(5):1173–1201, 2015. doi:10.1137/140990280.

16 Dima Grigoriev and Alexander A. Razborov. Exponential lower bounds for depth 3 arithmetic circuits in algebras of functions over finite fields. *Appl. Algebra Eng. Commun. Comput.*, 10(6):465–487, 2000. doi:10.1007/s002009900021.

17 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Approaching the chasm at depth four. *Journal of the ACM (JACM)*, 61(6):33, 2014. doi:10.1145/2629541.

18 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth 3. *SIAM Journal of Computing*, 45(3):1064–1079, 2016. doi:10.1137/140957123.

19 Nikhil Gupta, Chandan Saha, and Bhargav Thankey. A super-quadratic lower bound for depth four arithmetic circuits. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 23:1–23:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CCC.2020.23.

20 Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 302–309, New York, NY, USA, 1980. Association for Computing Machinery. doi:10.1145/800141.804678.

21 Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. Super-polynomial lower bounds for depth-4 homogeneous arithmetic formulas. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 119–127. ACM, 2014. doi:10.1145/2591796.2591823.

22 Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. An exponential lower bound for homogeneous depth four arithmetic formulas. *SIAM J. Comput.*, 46(1):307–335, 2017. doi:10.1137/151002423.

23 Neeraj Kayal and Chandan Saha. Multi-k-ic depth three circuit lower bound. *Theory Comput. Syst.*, 61(4):1237–1251, 2017. doi:10.1007/s00224-016-9742-9.

24 Neeraj Kayal, Chandan Saha, and Ramprasad Saptharishi. A super-polynomial lower bound for regular arithmetic formulas. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 146–153. ACM, 2014. doi:10.1145/2591796.2591847.

25 Neeraj Kayal, Chandan Saha, and Sébastien Tavenas. On the size of homogeneous and of depth-four formulas with low individual degree. *Theory of Computing*, 14(16):1–46, 2018. doi:10.4086/toc.2018.v014a016.

26 Mrinal Kumar and Shubhangi Saraf. Superpolynomial lower bounds for general homogeneous depth 4 arithmetic circuits. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 751–762. Springer, 2014. doi:10.1007/978-3-662-43948-7_62.

27 Mrinal Kumar and Shubhangi Saraf. The limits of depth reduction for arithmetic formulas: It's all about the top fan-in. *SIAM J. Comput.*, 44(6):1601–1625, 2015. doi:10.1137/140999220.

28 Mrinal Kumar and Shubhangi Saraf. On the power of homogeneous depth 4 arithmetic circuits. *SIAM J. Comput.*, 46(1):336–387, 2017. doi:10.1137/140999335.

29 Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. *Electron. Colloquium Comput. Complex.*, 28:81, 2021. URL: https://eccc.weizmann.ac.il/report/2021/081.

30 Cody D. Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime from a new easy witness lemma. *SIAM J. Comput.*, 49(5), 2020. doi:10.1137/18M1195887.

31 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. doi:10.1007/BF01294256.

32 Ran Raz. Multilinear-$\mathsf{NC}^2 \neq$ multilinear-NC^1. In *proceedings of Foundations of Computer Science (FOCS)*, pages 344–351, 2004. doi:10.1109/FOCS.2004.42.

33 Ran Raz. Separation of multilinear circuit and formula size. *Theory of Computing*, 2(1):121–135, 2006. doi:10.4086/toc.2006.v002a006.

34 Ran Raz. Elusive functions and lower bounds for arithmetic circuits. *Theory Comput.*, 6(1):135–177, 2010. doi:10.4086/toc.2010.v006a007.

35 Ran Raz and Amir Yehudayoff. Balancing syntactically multilinear arithmetic circuits. *Computational Complexity*, 17(4):515–535, 2008. doi:10.1007/s00037-008-0254-0.

36 Ran Raz and Amir Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity*, 18(2):171–207, 2009. doi:10.1007/s00037-009-0270-8.

37 Herbert John Ryser. *Combinatorial mathematics*, volume 14. American Mathematical Soc., 1963. URL: https://www.jstor.org/stable/10.4169/j.ctt5hh8v6.

38 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity version 8.0.4. Github survey, 2019. URL: https://github.com/dasarpmar/lowerbounds-survey/releases/.

39 Victor Shoup and Roman Smolensky. Lower bounds for polynomial evaluation and interpolation problems. *Computational Complexity*, 6(4):301–311, 1997. doi:10.1007/BF01270384.

40 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.

41 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 11h 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.

42 Leslie G. Valiant. *Why is Boolean Complexity Theory so Difficult?*, pages 84–94. London Mathematical Society Lecture Note Series. Cambridge University Press, 1992. doi:10.1017/CBO9780511526633.008.

43 V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proceedings of the Sixth Annual Structure in Complexity Theory Conference, Chicago, Illinois, USA, June 30 - July 3, 1991*, pages 270–284. IEEE Computer Society, 1991. doi:10.1109/SCT.1991.160269.

44 Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014. doi:10.1145/2559903.

On (Simple) Decision Tree Rank

Yogesh Dahiya
The Institute of Mathematical Sciences (HBNI), Chennai, India

Meena Mahajan
The Institute of Mathematical Sciences (HBNI), Chennai, India

Abstract

In the decision tree computation model for Boolean functions, the depth corresponds to query complexity, and size corresponds to storage space. The depth measure is the most well-studied one, and is known to be polynomially related to several non-computational complexity measures of functions such as certificate complexity. The size measure is also studied, but to a lesser extent. Another decision tree measure that has received very little attention is the minimal rank of the decision tree, first introduced by Ehrenfeucht and Haussler in 1989. This measure is not polynomially related to depth, and hence it can reveal additional information about the complexity of a function. It is characterised by the value of a Prover-Delayer game first proposed by Pudlák and Impagliazzo in the context of tree-like resolution proofs. In this paper we study this measure further. We obtain upper and lower bounds on rank in terms of (variants of) certificate complexity. We also obtain upper and lower bounds on the rank for composed functions in terms of the depth of the outer function and the rank of the inner function. We compute the rank exactly for several natural functions and use them to show that all the bounds we have obtained are tight. We also observe that the size-rank relationship for decision trees, obtained by Ehrenfeucht and Haussler, is tight upto constant factors.

2012 ACM Subject Classification Theory of computation → Oracles and decision trees

Keywords and phrases Boolean functions, Decision trees, certificate complexity, rank

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.15

1 Introduction

The central problem in Boolean function complexity is to understand exactly how hard it is to compute explicit functions. The hardness naturally depends on the computation model to be used, and depending on the model, several complexity measures for functions have been studied extensively in the literature. To name a few – size and depth for circuits and formulas, size and width for branching programs, query complexity, communication complexity, length for span programs, and so on. All of these are measures of the computational hardness of a function. There are also several ways to understand hardness of a function intrinsically, independent of a computational model. For instance, the sensitivity of a function, its certificate complexity, the sparsity of its Fourier spectrum, its degree and approximate degree, stability, and so on. Many bounds on computational measures are obtained by directly relating them to appropriate intrinsic complexity measures. See [10] for a wonderful overview of this area. Formal definitions of relevant measures appear in Section 2.

Every Boolean function f can be computed by a simple decision tree (simple in the sense that each node queries a single variable), which is one of the simplest computation models for Boolean functions. The most interesting and well-studied complexity measure in the decision tree model is the minimal depth $\mathrm{Depth}(f)$, measuring the query complexity of the function. This measure is known to be polynomially related to several intrinsic measures: sensitivity, block sensitivity, certificate complexity. But there are also other measures which reveal information about the function. The minimal size of a decision tree, $\mathrm{DTSize}(f)$, is one such measure, which measures the storage space required to store the function as a tree, and has received some attention in the past.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 15; pp. 15:1–15:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A measure which has received relatively less attention is the minimal rank of a decision tree computing the function, first defined and studied in [6]; see also [1]. In general, the rank of a rooted tree (also known as its Strahler number, or Horton-Strahler number, or tree dimension) measures its branching complexity, and is a tree measure that arises naturally in a wide array of applications; see for instance [7]. The rank of a Boolean function f, denoted $\mathrm{Rank}(f)$, is the minimal rank of a decision tree computing it. The original motivation for considering rank of decision trees was from learning theory – an algorithm, proposed in [6], and later simplified in [4], shows that constant-rank decision trees are efficiently learnable in Valiant's PAC learning framework [18]. Subsequently, the rank measure has played an important role in understanding the decision tree complexity of search problems over relations [14, 8, 11] – see more in the Related Work part below. The special case when the relation corresponds to a Boolean function is exactly the rank of the function. However, there is very little work focussing on the context of, and exploiting the additional information from, this special case. This is precisely the topic of this paper.

In this paper, we study how the rank of boolean functions relates to other measures. In contrast with $\mathrm{Depth}(f)$, $\mathrm{Rank}(f)$ is not polynomially related with sensitivity or to certificate complexity $\mathrm{C}(f)$, although it is bounded above by $\mathrm{Depth}(f)$. Hence it can reveal additional information about the complexity of a function over and above that provided by Depth. For instance, from several viewpoints, the $\textsc{Parity}_n$ function is significantly harder than the $\textsc{And}_n$ function. But both of them have the same Depth, n. However, Rank does reflect this difference in hardness, with $\mathrm{Rank}(\textsc{And}_n) = 1$ and $\mathrm{Rank}(\textsc{Parity}_n) = n$. On the other hand, rank is also already known to characterise the logarithm of decision tree size (DTSize), upto a $\log n$ multiplicative factor. Thus lower bounds on rank give lower bounds on the space required to store a decision tree explicitly. (However, the $\log n$ factor is crucial; there is no dimension-free characterisation. Consider e.g. $\log \mathrm{DTSize}(\textsc{And}_n) = \Theta(\log n)$.)

Our main findings can be summarised as follows:

1. $\mathrm{Rank}(f)$ is equal to the value of the Prover-Delayer game of Pudlák and Impagliazzo [14] played on the corresponding relation R_f. (This is implicit in earlier literature [11, 8].)
2. $\mathrm{Rank}(f)$ is bounded between the minimum certificate complexity of f at any point, and $(\mathrm{C}(f) - 1)^2 + 1$; Theorem 5.6. The upper bound (Lemma 5.2) is an improvement on the bound inherited from $\mathrm{Depth}(f)$, and is obtained by adapting that construction.
3. For a composed function $f \circ g$, $\mathrm{Rank}(f \circ g)$ is bounded above and below by functions of $\mathrm{Depth}(f)$ and $\mathrm{Rank}(g)$; Theorem 6.6. The main technique in both bounds (Theorems 6.3 and 6.5) is to use weighted decision trees, as was used in the context of depth [13].
4. The relation between $\mathrm{Rank}(f)$ and $\mathrm{DTSize}(f)$ from [6] is tight, Section 7. In particular, for the $\textsc{Tribes}$ function, the $\log n$ multiplicative factor is necessary.

By calculating the exact rank for specific functions, we show that all the bounds we obtain on rank are tight.

Related work

In [1], a model called k^+-decision trees is considered, and the complexity is related to both simple decision tree rank and to communication complexity. In particular, Theorems 7 and 8 from [1] imply that communication complexity lower bounds with respect to any variable partition (see [12]) translate to decision tree rank lower bounds, and hence by [6] to decision tree size lower bounds.

In [16], the model of linear decision trees is considered (here each node queries not a single variable but a linear threshold function of the variables), and for such trees of bounded rank computing the inner product function, a lower bound on depth is obtained. Thus for this function, in this model, there is a trade-off between rank and depth. In [17], rank of linear decision trees is used in obtaining non-trivial upper bounds on depth-2 threshold circuit size.

In [14], a 2-player game is described, on an unsatisfiable formula F in conjunctive normal form, that constructs a partial assignment falsifying some clause. The players are referred to in subsequent literature as the Prover and the Delayer. The value of the game, $\text{Value}(F)$, is the maximum r such that the Delayer can score at least r points no matter how the Prover plays. It was shown in [14] that the size of any tree-like resolution refutation of F is at least $2^{\text{Value}(F)}$. Subsequently, the results of [11, 8] yield the equivalence $\text{Value}(F) = \text{Rank}(F)$, where $\text{Rank}(F)$ is defined to be the minimal rank of the tree underlying a tree-like resolution refutation of F. (Establishing this equivalence uses refutation-space and tree pebbling as intermediaries.) The relevance here is because there is an immediate, and well-known, connection to decision trees for search problems over relations: tree-like resolution refutations are decision trees for the corresponding search CNF problem. (See Lemma 7 in [2]). Note that the size lower bound from [14], and the rank-value equivalence from [11, 8], hold for the search problem over arbitrary relations, not just searchCNF. (See e.g. Exercise 14.16 in Jukna for the size bound.) In particular, for Boolean function f, it holds for the corresponding canonical relation R_f defined in Section 2. Similarly, the value of an asymmetric variant of this game is known to characterise the size of a decision tree for the search CNF problem [3], and this too holds for general relations and Boolean functions.

Organisation of the paper

After presenting basic definitions and known results in Section 2, we describe the Prover-Delayer game from [14] in Section 3, and observe that its value equals the rank of the function. We also describe the asymmetric game from [3]. We compute the rank of some simple functions in Section 4. In Section 5, we describe the relation between rank and certificate complexity. In Section 6, we present results concerning composed functions. Section 7 examines the size-rank relationship for the TRIBES function. The bounds in Sections 4–7 are all obtained by direct inductive arguments/decision tree constructions. They can also be stated using the equivalence of the game value and rank – while this does not particularly simplify the proofs, it changes the language of the proofs and may be more accessible to the reader already familiar with that setting. Hence we illustrate such game-based arguments for some of our results in Section 8.

2 Preliminaries

Decision trees

For a Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, a decision tree computing f is a binary tree with internal nodes labeled by the variables and the leaves labelled by $\{0,1\}$. To evaluate a function on an unknown input, the process starts at the root of the decision tree and works down the tree, querying the variables at the internal nodes. If the value of the query is 0, the process continues in the the left subtree, otherwise it proceeds in the right subtree. The label of the leaf so reached is the value of the function on that particular input. A decision tree is said to be reduced if no variable is queried more than once on any root-to-leaf path. Without loss of generality, any decision tree can be reduced, so in our discussion, we will only consider reduced decision trees. The depth $\text{Depth}(T)$ of a decision tree T is the length of the longest root-to-leaf path, and its size $\text{DTSize}(T)$ is the number of leaves. The decision tree complexity or the depth of f, denoted by $\text{Depth}(f)$, is defined to be the minimum depth of a decision tree computing f. Equivalently, $\text{Depth}(f)$ can also be seen as the minimum number of worst-case queries required to evaluate f. The size of a function f, denoted by $\text{DTSize}(f)$, is defined similarly i.e. the minimum size of a decision tree computing f. Since decision trees can be reduced, $\mathrm{Depth}(f) \leq n$ and $\mathrm{DTSize}(f) \leq 2^n$ for every n-variate function f. A function is said to be evasive if its depth is maximal, $\mathrm{Depth}(f) = n$.

Weighted decision trees

Weighted decision trees describe query complexity in settings where querying different input bits can have differing cost, and arises naturally in the recursive construction. Formally, these are defined as follows: Let w_i be the cost of querying variable x_i. For a decision tree T, its weighted depth with respect to the weight vector $[w_1, \ldots, w_n]$, denoted by $\mathrm{Depth}_w(T, [w_1, w_2, ..., w_n])$, is the maximal sum of weights of the variables specified by the labels of nodes of T on any root-to-leaf path. The weighted decision tree complexity of f, denoted by $\mathrm{Depth}_w(f, [w_1, w_2, ..., w_n])$, is the minimum weighted depth of a decision tree computing f. Note that $\mathrm{Depth}(f)$ is exactly $\mathrm{Depth}_w(f, [1, 1, \ldots, 1])$. The following fact is immediate from the definitions.

▶ **Fact 2.1.** *For any reduced decision tree T computing an n-variate function, weights $w_1, \ldots, w_n$, and $i \in [n]$,*

$$\mathit{Depth}_w(T, [w_1, \ldots, w_{i-1}, w_i + 1, w_{i+1}, \ldots, w_n]) \leq \mathit{Depth}_w(T, [w_1, w_2, ..., w_n]) + 1.$$

Certificate Complexity

The certificate complexity of a function f, denoted $\mathrm{C}(f)$, measures the number of variables that need to be assigned in the worst case to fix the value of f. More precisely, for a Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ and an input $a \in \{0,1\}^n$, an f-certificate of a is a subset $S \subseteq \{1, ..., n\}$ such that the value of $f(a)$ can be determined by just looking at the bits of a in set S. Such a certificate need not be unique. Let $\mathrm{C}(f, a)$ denote the minimum size of an f-certificate for the input a. That is,

$$\mathrm{C}(f, a) = \min \left\{ |S| \mid S \subseteq [n]; \forall a' \in \{0,1\}^n, \left[\left(a'_j = a_j \forall j \in S \right) \implies f(a') = f(a) \right] \right\}.$$

Using this definition, we can define several measures.

$$\begin{aligned} \text{For} \quad b \in \{0,1\}, \quad \mathrm{C}_b(f) &= \max\{\mathrm{C}(f, a) \mid a \in f^{-1}(b)\} \\ \mathrm{C}(f) &= \max\{\mathrm{C}(f, a) \mid a \in \{0,1\}^n\} = \max\{\mathrm{C}_0(f), \mathrm{C}_1(f)\} \\ \mathrm{C}_{avg}(f) &= 2^{-n} \sum_{a \in \{0,1\}^n} \mathrm{C}(f, a) \\ \mathrm{C}_{\min}(f) &= \min\{\mathrm{C}(f, a) \mid a \in \{0,1\}^n\} \end{aligned}$$

Composed functions

For boolean functions $f, g_1, g_2, \ldots, g_n$ of arity $n, m_1, m_2, \ldots, m_n$ respectively, the composed function $f \circ (g_1, g_2, ..., g_n)$ is a function of arity $\sum_i m_i$, and is defined as follows: for $a^i \in \{0,1\}^{m_i}$ for each $i \in n$, $f \circ (g_1, g_2, ..., g_n)(a^1, a^2, ..., a^n) = f(g_1(a^1), g_2(a^2), \ldots, g_n(a^n))$. We call f the outer function and $g_1, \ldots, g_n$ the inner functions. For functions $f : \{0,1\}^n \longrightarrow \{0,1\}$ and $g : \{0,1\}^m \longrightarrow \{0,1\}$, the composed function $f \circ g$ is the function $f \circ (g, g, \ldots, g) : \{0,1\}^{mn} \longrightarrow \{0,1\}$. The composed function $\textsc{Or}_n \circ \textsc{And}_m$ has a special name, $\textsc{Tribes}_{n,m}$, and when $n = m$, we simply write $\textsc{Tribes}_n$. Its dual is the function $\textsc{And}_n \circ \textsc{Or}_m$ that we denote $\textsc{Tribes}^d_{n,m}$. (The dual of $f(x_1, \ldots, x_n)$ is the function $\neg f(\neg x_1, \ldots, \neg x_n)$.)

Symmetric functions

A Boolean function is symmetric if its value depends only on the number of ones in the input, and not on the positions of the ones.

▶ **Proposition 2.2.** *For every non-constant symmetric boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$,*

1. *f is evasive (has $Depth(f) = n$). (See eg. Lemma 14.19 [10].)*
2. *Hence, for any weights w_i, $Depth_w(f, [w_1, w_2, ..., w_n])) = \sum_i w_i$.*

For a symmetric Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, let $f_0, f_1, ..., f_n \in \{0,1\}$ denote the values of the function f on inputs of Hamming weight $0, 1, ..., n$ respectively. The Gap of f is defined as the length of the longest interval (minus one) where f_i is constant. That is,

$$\text{Gap}(f) = \max_{0\le a\le b\le n} \{b - a : f_a = f_{a+1} = ... = f_b\}.$$

Analogously, $\text{Gap}_{\text{min}}(f)$ is the length of the shortest constant interval (minus one); that is, setting $f_{-1} \neq f_0$ and $f_{n+1} \neq f_n$ for boundary conditions,

$$\text{Gap}_{\text{min}}(f) = \min_{0\le a\le b\le n} \{b - a : f_{a-1} \neq f_a = f_{a+1} = ... = f_b \neq f_{b+1}\}.$$

Decision Tree Rank

For a rooted binary tree T, the rank of the tree is the rank of the root node, where the rank of each node of the tree is defined recursively as follows: For a leaf node u, $\text{Rank}(u) = 0$. For an internal node u with children v, w,

$$\text{Rank}(u) = \begin{cases} \text{Rank}(v) + 1 & \text{if Rank}(v) = \text{Rank}(w) \\ \max\{\text{Rank}(v), \text{Rank}(w)\} & \text{if Rank}(v) \neq \text{Rank}(w) \end{cases}$$

The following proposition lists some known properties of the rank function for binary trees.

▶ **Proposition 2.3.** *For any binary tree T,*

1. *(Rank and Size relationship): $Rank(T) \le \log(DTSize(T)) \le Depth(T)$.*
2. *(Monotonicity of the Rank): Let T' be any subtree of T, and let T'' be an arbitrary binary tree of higher rank than T'. If T' is replaced by T'' in T, then the rank of the resulting tree is not less than the rank of T.*
3. *(Leaf Depth and Rank): If all leaves in T have depth at least r, then $Rank(T) \ge r$.*

For a Boolean function f, the rank of f, denoted $\text{Rank}(f)$, is the minimum rank of a decision tree computing f.

From Proposition 2.3(2), we see that the rank of a subfunction of f (a function obtained by assigning values to some variables of f) cannot exceed the rank of the function itself.

▶ **Proposition 2.4.** *(Rank of a subfunction): Let f_S be a subfunction obtained by fixing the values of variables in some set $S \subseteq [n]$ of f. Then $Rank(f_S) \le Rank(f)$.*

The following rank and size relationship is known for boolean functions.

▶ **Proposition 2.5** (Lemma 1 [6]). *For a non-constant Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$,*

$$Rank(f) \le \log DTSize(f) \le Rank(f) \log\left(\frac{en}{Rank(f)}\right).$$

For symmetric functions, Rank is completely characterized in terms of Gap.

▶ **Proposition 2.6** (Lemma C.6 [1]). *For symmetric Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, $\mathit{Rank}(f) = n - \mathit{Gap}(f)$.*

▶ Remark 2.7. For (simple) deterministic possibly weighted decision trees, each of the measures DTSize, Depth, and Rank, is the same for a Boolean function f, its complement $\neg f$, and its dual f^d.

Relations and Search problems

A relation $R \subseteq X \times W$ is said to be X-complete, or just complete, if its projection ot X equals X. That is, for every $x \in X$, there is a $w \in W$ with $(x, w) \in R$. For an X-complete relation R, where X is of the form $\{0,1\}^n$ for some n, the search problem SearchR is as follows: given an $x \in X$, find a $w \in W$ with $(x, w) \in R$. A decision tree for SearchR is defined exactly as for Boolean functions; the only diference is that leaves are labeled with elements of W, and we require that for each input x, if the unique leaf reached on x is labeled w, then $(x, w) \in R$. The rank of the relation, Rank(R), is the minimum rank of a decision tree solving the SearchR problem.

A Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ naturally defines a complete relation R_f over $X = \{0,1\}^n$ and $W = \{0,1\}$, with $R_f = \{(x, f(x)) \mid x \in X\}$, and $\mathrm{Rank}(f) = \mathrm{Rank}(R_f)$.

3 Game Characterisation for Rank

In this section we observe that the rank of a Boolean function is characterised by the value of a Prover-Delayer game introduced by Pudlák and Impagliazzo in [14]. As mentioned in Section 1, the game was originally described for searchCNF problems on unsatisfiable clause sets. The appropriate analog for a Boolean function f, or its relation R_f, and even for arbitrary X-complete relations $R \subseteq X \times W$, is as follows:

The game is played by two players, the Prover and the Delayer, who construct a (partial) assignment ρ in rounds. Initially, ρ is empty. In each round, the Prover queries a variable x_i not set by ρ. The Delayer responds with a bit value 0 or 1 for x_i, or defers the choice to the Prover. In the later case, Prover can choose the value for the queried variable, and the Delayer scores one point. The game ends when there is a $w \in W$ such that for all x consistent with ρ, $(x, w) \in R$. (Thus, for a Boolean function f, the game ends when $f|_\rho$ is a constant function.) The value of the game, Value(R), is the maximum k such that the Delayer can always score at least k points, no matter how the Prover plays.

▶ **Theorem 3.1** (implied from [14, 11, 8]). *For any X-complete relation $R \subseteq X \times W$, where $X = \{0,1\}^n$, $\mathit{Rank}(R) = \mathit{Value}(R)$. In particular, for a boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, $\mathit{Rank}(f) = \mathit{Value}(R_f)$.*

In [3], an aysmmmetric version of this game is defined. In each round, the Prover queries a variable x, the Delayer specifies values $p_0, p_1 \in [0,1]$ adding up to 1, the Prover picks a value b, the Delayer adds $\log \frac{1}{p_b}$ to his score. Let ASym-Value denote the maximum score the Delayer can always achieve, independent of the Prover moves. Note that ASym-Value(R) $\geq$ Value(R); an asymmetric-game Delayer can mimic a symmetric-game Delayer by using $p_b = 1$ for choice b and $p_0 = p_1 = 1/2$ for deferring. As shown in [3], for the search CNF problem, the value of this asymmetric game is exactly the optimal leaf-size of a decision tree. We note below that this holds for the SearchR problem more generally.

▶ **Proposition 3.2** (implicit in [3]). *For any X-complete relation $R \subseteq X \times W$, where $X = \{0,1\}^n$, $\log \mathit{DTSize}(R) = \mathit{ASym\text{-}Value}(R)$. In particular, for a boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, $\log \mathit{DTSize}(f) = \mathit{ASym\text{-}Value}(R_f)$.*

(In [3], the bounds have $\log\lceil S/2\rceil$; this is because S there counts all nodes in the decision tree, while here we count only leaves.)

Thus we have the relationship

$$\mathrm{Rank}(f) = \mathrm{Value}(R_f) \leq \mathrm{ASym\text{-}Value}(R_f) = \log \mathrm{DTSize}(f).$$

4 The Rank of some natural functions

For symmetric functions, rank can be easily calculated using Proposition 2.6. In Table 1 we tabulate various measures for some standard symmetric functions. As can be seen from the OR_n and AND_n functions, the $\mathrm{Rank}(f)$ measure is not polynomially related with the measures $\mathrm{Depth}(f)$ or certificate complexity $\mathrm{C}(f)$.

Table 1 Some simple symmetric functions and their associated complexity measures.

f	Depth	C_0	C_1	C	Gap	Rank
0 or 1	0	0	0	0	n	0
AND_n	n	1	n	n	$n-1$	1
OR_n	n	n	1	n	$n-1$	1
PARITY_n	n	n	n	n	0	n
MAJ_{2k}	$2k$	k	$k+1$	$k+1$	k	k
MAJ_{2k+1}	$2k+1$	$k+1$	$k+1$	$k+1$	k	$k+1$
THR_n^k $(k \geq 1)$	n	$n-k+1$	k	$\max\left\{\begin{matrix} n-k+1, \\ k \end{matrix}\right\}$	$\max\left\{\begin{matrix} k-1, \\ n-k \end{matrix}\right\}$	$n - \mathrm{Gap}$

For two composed functions that will be crucial in our later discussions, we can directly calculate the rank as described below.

▶ **Theorem 4.1.** *For every $n \geq 1$,*
1. *$\mathrm{Rank}(\mathrm{TRIBES}_{n,m}) = \mathrm{Rank}(\mathrm{TRIBES}^d_{n,m}) = n$ for $m \geq 2$.*
2. *$\mathrm{Rank}(\mathrm{AND}_n \circ \mathrm{PARITY}_m) = n(m-1)+1$ for $m \geq 1$.*

We prove each of the lower and upper bounds separately in a series of lemmas below. The lemmas use the following properties about the rank function which can be easily verified.

▶ **Proposition 4.2** (Composition of Rank). *Let T be a rooted binary tree with depth ≥ 1, rank r, and with leaves labelled by 0 and 1. Let T_0, T_1 be arbitrary rooted binary trees of ranks r_0, r_1 respectively. For $b \in \{0,1\}$, attach T_b to each leaf of T labeled b, to obtain rooted binary tree T' of rank r'.*
1. *$r' \leq r + \max\{r_0, r_1\}$. Furthermore, if T is a complete binary tree, and if $r_0 = r_1$, then this is an equality; $r' = r + r_0$.*
2. *If every non-trivial subtree (more than one leaf) of T has both a 0 leaf and a 1 leaf, then $r' \geq r + \max\{r_0, r_1\} - 1$. If, furthermore, T is a complete binary tree, then this is an equality when $r_0 \neq r_1$,*

We first establish the bounds for $\mathrm{TRIBES}^d_{n,m} = \bigwedge_{i\in[n]} \bigvee_{j\in[m]} x_{i,j}$.

▶ **Lemma 4.3.** *For every $n, m \geq 1$, $\mathrm{Rank}(\mathrm{TRIBES}^d_{n,m}) \leq n$.*

Proof. We show the bound by giving a recursive construction and bounding the rank by induction on n. In the base case, $n = 1$. $\textsc{Tribes}^d_{1,m} = \textsc{Or}_m$, which has rank 1. For the inductive step, $n \geq 1$. For $j < n$, let $T_{j,m}$ denote the recursively constructed trees for $\textsc{Tribes}^d_{j,m}$. Take the tree T which is $T_{1,m}$ on variables $x_{n,j}$, $j \in [m]$. Attach the tree $T_{n-1,m}$ on variables $x_{i,j}$ for $i \in [n-1]$, $j \in [m]$, to all the 1-leaves of T, to obtain $T_{n,m}$. It is straightforward to see that this tree computes $\textsc{Tribes}^d_{n,m}$. Using Proposition 4.2 and induction, we obtain $\text{Rank}(T_{n,m}) \leq \text{Rank}(T_{1,m}) + \text{Rank}(T_{n-1,m}) \leq 1 + (n-1) = n$. ◀

▶ Remark 4.4. More generally, this construction shows that $\text{Rank}(\textsc{And}_n \circ f) \leq n\text{Rank}(f)$.

▶ **Lemma 4.5.** *For every $n \geq 1$ and $m \geq 2$, Rank$(\textsc{Tribes}^d_{n,m}) \geq n$.*

Proof. We prove this by induction on n. The base case, $n = 1$, is straightforward: $\textsc{Tribes}^d_{1,m}$ is the function $\textsc{Or}_m$, whose rank is 1.

For the inductive step, let $n > 1$, and consider any decision tree Q for $\textsc{Tribes}^d_{n,m}$. Without loss of generality (by renaming variables if necessary), let $x_{1,1}$ be the variable queried at the root node. Let Q_0 and Q_1 be the left and the right subtrees of Q. Then Q_0 computes the function $\textsc{And}_n \circ (\textsc{Or}_{m-1}, \textsc{Or}_m, ..., \textsc{Or}_m)$, and Q_1 computes $\textsc{Tribes}^d_{n-1,m}$, on appropriate variables. For $m \geq 2$, $\textsc{Tribes}^d_{n-1,m}$ is a sub-function of $\textsc{And}_n \circ (\textsc{Or}_{m-1}, \textsc{Or}_m, ..., \textsc{Or}_m)$, and so Proposition 2.4 implies that $\text{Rank}(Q_0) \geq \text{Rank}(\textsc{And}_n \circ (\textsc{Or}_{m-1}, \textsc{Or}_m, ..., \textsc{Or}_m)) \geq \text{Rank}(\textsc{Tribes}^d_{n-1,m})$. By induction, $\text{Rank}(Q_1) \geq \text{Rank}(\textsc{Tribes}^d_{n-1.m}) \geq n-1$. Hence, by definition of rank, $\text{Rank}(Q) \geq 1 + \min\{\text{Rank}(Q_0), \text{Rank}(Q_1\} \geq n$. Since this holds for every decision tree Q for $\textsc{Tribes}^d_{n,m}$, we conclude that $\text{Rank}(\textsc{Tribes}^d_{n,m}) \geq n$, as claimed. ◀

Next, we establish the bounds for $\textsc{And}_n \circ \textsc{Parity}_m = \bigwedge_{i\in[n]} \bigoplus_{j\in[m]} x_{i,j}$. The upper bound below is slightly better than what is implied by Remark 4.4.

▶ **Lemma 4.6.** *For every $n, m \geq 1$, Rank$(\textsc{And}_n \circ \textsc{Parity}_m) \leq n(m-1) + 1$.*

Proof. Recursing on n, we construct decision trees $T_{n,m}$ for $\textsc{And}_n \circ \textsc{Parity}_m$, as in Lemma 4.3. By induction on n, we bound the rank, also additionally using the fact that the rank-optimal decision tree for $\textsc{Parity}_m$ is a complete binary tree.

Base Case: $n = 1$. $\textsc{And}_1 \circ \textsc{Parity}_m = \textsc{Parity}_m$. From Table 1, $\text{Rank}(\textsc{Parity}_m) = m$; let $T_{1,m}$ be the optimal decision tree computing $\textsc{Parity}_m$.

Inductive Step: $n \geq 1$. For $j < n$, let $T_{j,m}$ denote the recursively constructed trees for $\textsc{And}_j \circ \textsc{Parity}_m$. Take the tree T which is $T_{1,m}$ on variables $x_{n,j}$, $j \in [m]$. Attach the tree $T_{n-1,m}$ on variables $x_{i,j}$ for $i \in [n-1]$, $j \in [m]$, to all the 1-leaves of T, to obtain $T_{n,m}$. It is straightforward to see that this tree computes $\textsc{And}_n \circ \textsc{Parity}_m$.

By induction, $\text{Rank}(T_{n-1,m}) \leq (n-1)(m-1) + 1 \geq 1$. Since we do not attach anything to the 0-leaves of $T_{1,m}$ (or equivalently, we attach a rank-0 tree to these leaves), and since $T_{1,m}$ is a complete binary tree, the second statement in Proposition 4.2 yields $\text{Rank}(T_{n,m}) = \text{Rank}(T_{1,m}) + \text{Rank}(T_{n-1,m}) - 1$. Hence $\text{Rank}(T_{n,m}) \leq n(m-1) + 1$, as claimed. ◀

▶ **Lemma 4.7.** *For every $n, m_1, m_2, \ldots, m_n \geq 1$, and functions $g_1, g_2, \ldots, g_n$ each in $\{\textsc{Parity}_m, \neg\textsc{Parity}_m\}$, Rank$(\textsc{And}_n \circ (g_1, g_2, ..., g_n)) \geq (\sum_{i=1}^n (m_i - 1)) + 1$.*

In particular, Rank$(\textsc{And}_n \circ \textsc{Parity}_m) \geq n(m-1) + 1$.

Proofs of this lemma and Lemma 4.6, based on the Prover-Delayer game characterisation Theorem 3.1, appear in Section 8. This lemma is also an immediate consequence of the more general Theorem 6.6 that we prove later.

5 Relation between Rank and Certificate Complexity

The certificate complexity and decision tree complexity are known to be related as follows.

▶ **Proposition 5.1** ([5, 9, 15], see also Theorem 14.3 in [10]). *For every boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$,*

$$C(f) \leq Depth(f) \leq C_0(f)C_1(f)$$

Both these inequalities are tight; the first for the OR and AND functions, and the second for the $\text{TRIBES}_{n,m}$ and $\text{TRIBES}^d_{n,m}$ functions. (For $\text{TRIBES}^d_{n,m}$, $\text{C}_0(\text{TRIBES}^d_{n,m}) = m$, $\text{C}_1(\text{TRIBES}^d_{n,m}) = n$ and $\text{Depth}(\text{TRIBES}^d_{n,m}) = nm$, see e.g. Exercise 14.1 in [10].)

Since Rank $\leq$ Depth, the same upper bound also holds for Rank as well. But it is far from tight for the $\text{TRIBES}_{n,m}$ function. In fact, the upper bound can be improved in general. Adapting the construction given in the proof of Proposition 5.1 slightly, we show the following.

▶ **Lemma 5.2.** *For every Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$,*

$$Rank(f) \leq (C_0(f) - 1)(C_1(f) - 1) + 1$$

Moreover, the inequality is tight as witnessed by AND *and* OR *functions.*

Proof (Sketch). The proof of Proposition 5.1 proceeds by constructing a decision tree in stages. In each stage, all variables from some 0-certificate are queried. Each stage contributes at most C_0 to depth, and reduces C_1 by at least one, giving the bound. We note that since at least one leaf in each stage is a leaf of the final tree, each stage contributes at most $\text{C}_0 - 1$ to rank. Further, in the last stage, the contribution to rank can be reduced to just 1. ◀

From Theorem 4.1, we see that the lower bound on Depth in Proposition 5.1 does not hold for Rank; for $m > n$, $\text{Rank}(\text{TRIBES}^d_{n,m}) = n < m = C(\text{TRIBES}^d_{n,m})$. However, $\min\{\text{C}_0(\text{TRIBES}^d_{n,m}), \text{C}_1(\text{TRIBES}^d_{n,m})\} = n = \text{Rank}(\text{TRIBES}^d_{n,m})$. Further, for all the functions listed in Table 1, $\text{Rank}(f)$ is at least as large as $\min\{\text{C}_0(f), \text{C}_1(f)\}$. However, even this is not a lower bound in general.

▶ **Lemma 5.3.** $\min\{C_0(f), C_1(f)\}$ *is not a lower bound on $Rank(f)$; for the symmetric function $f = \text{MAJ}_n \vee \text{PARITY}_n$, when $n > 4$, $Rank(f) < \min\{C_0(f), C_1(f)\}$.*

Proof. Let f be the function $\text{MAJ}_n \vee \text{PARITY}_n$, for $n > 4$. Then $f(0^n) = 0$ and $\text{C}_0(f, 0^n) = n$, and $f(10^{n-1}) = 1$ and $\text{C}_1(f, 10^{n-1}) = n$. Also, f is symmetric, with $\text{Gap}(f) = n/2$, so by Proposition 2.6, $\text{Rank}(f) = n/2$. ◀

The average certificate complexity is also not directly related to rank.

▶ **Lemma 5.4.** *Average certificate complexity is neither a upper bound nor a lower bound on the rank of a function; for functions $f = \text{AND}_n$ and $g = \text{TRIBES}^d_{n,2}$, $Rank(f) < C_{avg}(f)$ and $C_{avg}(g) < Rank(g)$.*

What can be shown in terms of certificate complexity and rank is the following:

▶ **Lemma 5.5.** *For every Boolean function f, $C_{\min}(f) \leq Rank(f)$. This is tight for* OR_n.

Lemma 5.2 and Lemma 5.5 give these bounds sandwiching $\text{Rank}(f)$:

▶ **Theorem 5.6.** $C_{\min}(f) \leq Rank(f) \leq (C_0(f) - 1)(C_1(f) - 1) + 1 \leq (C(f) - 1)^2 + 1.$

As mentioned in Proposition 2.6, for symmetric functions the rank is completely characterised in terms of Gap of f. How does Gap relate to certificate complexity for such functions? It turns out that certificate complexity is characterized not by Gap but by $\mathrm{Gap}_{\min}$. Using this relation, the upper bound on $\mathrm{Rank}(f)$ from Lemma 5.2 can be improved for symmetric functions to $\mathrm{C}(f)$.

▶ **Lemma 5.7.** *For every symmetric Boolean function f on n variables, $C(f) = n - Gap_{\min}(f)$ and $n - C(f) + 1 \leq Rank(f) \leq C(f)$. Both the inequalities on rank are tight for* $\textsc{Maj}_{2k+1}$.

Proof. We first show $\mathrm{C}(f) = n - \mathrm{Gap}_{\min}(f)$. Consider any interval $[a, b]$ such that $f_{a-1} \neq f_a = f_{a+1} = \ldots = f_b \neq f_{b+1}$. Let x be any input with Hamming weight in the interval $[a, b]$. We show that $C(f, x) = n - (b - a)$.

1. Pick any $S \subseteq [n]$ containing exactly a bit positions where x is 1, and exactly $n - b$ bit positions where x is 0. Any y agreeing with x on S has Hamming weight in $[a, b]$, and hence $f(y) = f(x)$. Thus S is a certificate for x. Hence $C(f, x) \leq n - (b - a)$.
2. Let $S \subseteq [n]$ be any certificate for x. Suppose S contains fewer than a bit positions where x is 1. Then there is an input y that agrees with x on S and has Hamming weight exactly $a - 1$. (Flip some of the 1s from x that are not indexed in S.) So $f(y) \neq f(x)$, contradicting the fact that S is a certificate for x. Similarly, if S contains fewer that $n - b$ bit positions where x is 0, then there is an input z that agrees with x on S and has Hamming weight exactly $b + 1$. So $f(z) \neq f(x)$, contradicting the fact that S is a certificate for x.

Thus any certificate for x must have at least $a + (n - b)$ positions; hence $C(f, x) \geq n - (b - a)$. Since the argument above works for any interval $[a, b]$ where f is constant, we conclude that $\mathrm{C}(f) = n - \mathrm{Gap}_{\min}(f)$.

Next, observe that $\mathrm{Gap}(f) + \mathrm{Gap}_{\min}(f) \leq n - 1$. Hence,

$$n - \mathrm{C}(f) + 1 = \mathrm{Gap}_{\min}(f) + 1 \leq n - Gap(f) = \mathrm{Rank}(f) \leq n - \mathrm{Gap}_{\min}(f) = \mathrm{C}(f).$$

As seen from Table 1, these bounds on Rank are tight for $\textsc{Maj}_{2k+1}$. ◀

Even for the (non-symmetric) functions in Theorem 4.1, $\mathrm{Rank}(f) \leq \mathrm{C}(f)$. However, this is not true in general.

▶ **Lemma 5.8.** *Certificate Complexity does not always bound Rank from above; for the function $f = \textsc{Maj}_{2k+1} \circ \textsc{Maj}_{2k+1}$, $C(f) < Rank(f)$.*

The proof is deferred to Section 6, where we develop techniques to bound the rank of composed functions. We also give, in Section 8, a proof based on the Prover-Delayer game characterisation Theorem 3.1.

6 Rank of Composed functions

In this section we study the rank for composed functions. For composed functions, $f \circ g$, decision tree complexity Depth is known to behave very nicely.

▶ **Proposition 6.1** ([13]). *For Boolean functions f, g, $Depth(f \circ g) = Depth(f)Depth(g)$.*

We want to explore how far something similar can be deduced about $\mathrm{Rank}(f \circ g)$. The first thing to note is that a direct analogue in terms of Rank alone is ruled out.

▶ **Lemma 6.2.** *For general Boolean functions f and g, $Rank(f \circ g)$ cannot be bounded by any function of $Rank(f)$ and $Rank(g)$ alone.*

Proof. Let $f = \textsc{And}_n$ and $g = \textsc{Or}_n$. Then $\text{Rank}(f) = \text{Rank}(g) = 1$. But $\text{Rank}(f \circ g) = \text{Rank}(\textsc{Tribes}^d_n) = n$, as seen in Theorem 4.1. ◀

For $f \circ g$, let T_f, T_g be decision trees for f, g respectively. One way to construct a decision tree for $f \circ g$ is to start with T_f, inflate each internal node u of T_f into a copy of T_g on the appropriate inputs, and attach the left and the right subtree of u as appropriate at the leaves of this copy of T_g. By Proposition 6.1, the decision tree thus obtained for $f \circ g$ is optimal for Depth if one start with depth-optimal trees T_f and T_g for f and g respectively. In terms of rank, we can also show that the rank of the decision tree so constructed is bounded above by $\text{Depth}(T_f)\text{Rank}(T_g) = \text{Depth}_w(f, [r, r, \ldots, r])$, where $r = \text{Rank}(T_g)$. (This is the construction used in the proofs of Lemmas 4.3 and 4.6, where further properties of the $\textsc{Parity}$ function are used to show that the resulting tree's rank is even smaller than $\text{Depth}(f)\text{Rank}(g)$.) In fact, we show below (Theorem 6.3) that this holds more generally, when different functions are used in the composition. While this is a relatively straightforward generalisation here, it is necessary to consider such compositions for the lower bound we establish further on in this section.

▶ **Theorem 6.3.** *For non-constant boolean functions $g_1, \ldots, g_n$ with $Rank(g_i) = r_i$, and for n-variate non-constant booolean function f,*

$$Rank(f \circ (g_1, g_2, ..., g_n)) \le Depth_w(f, [r_1, r_2, ..., r_n]).$$

The really interesting question, however, is whether we can show a good lower bound for the rank of a composed function. This will help us understand how good is the upper bound in Theorem 6.3. To begin with, note that for non-constant Boolean functions f, g, both f and g are sub-functions of $f \circ g$. Hence Proposition 2.4 implies the following.

▶ **Proposition 6.4.** *For non-constant boolean functions f, g,*

$$Rank(f \circ g) \ge \max\{Rank(f), Rank(g)\}.$$

A better lower bound in terms of weighted depth complexity of f is given below. This generalises the lower bounds from Lemmas 4.5 and 4.7. The proofs of those lemmas crucially used nice symmetry properties of the inner function, whereas the bound below applies for any non-constant inner function. It is significantly weaker than the bound from Lemma 4.5 but matches that from Lemma 4.7.

▶ **Theorem 6.5.** *For non-constant boolean functions $g_1, \ldots, g_n$ with $Rank(g_i) = r_i$, and for n-variate non-constant boolean function f,*

$$\begin{aligned} Rank(f \circ (g_1, g_2, ..., g_n)) &\ge Depth_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 \\ &\ge Depth_w(f, [r_1, r_2, ..., r_n]) - (n - 1). \end{aligned}$$

Proof. The second inequality above is straightforward: let T be a decision tree for f that is optimal with respect to weights $r_1 - 1, \ldots, r_n - 1$. Since T can be assumed to be reduced, repeated application of Fact 2.1 shows that the depth of T with respect to weights $r_1, \ldots, r_n$ increases by at most n. Thus $\text{Depth}_w(f, [r_1, \ldots, r_n]) \le \text{Depth}_w(T, [r_1, \ldots, r_n]) \le \text{Depth}_w(T, [r_1 - 1, \ldots, r_n - 1]) + n = \text{Depth}_w(f, [r_1 - 1, \ldots, r_n - 1]) + n$, giving the claimed inequality.

The first inequality is not so straightforward. We prove it by induction on n. Let h denote the function $f \circ (g_1, g_2, ..., g_n)$. For $i \in [n]$, let m_i be the arity of g_i. We call $x_{i,1}, x_{i,2}, \ldots, x_{i,m_i}$ the ith block of variables of h; g_i is evaluated on this block.

In the base case, $n = 1$. Since f is non-constant, $f \in \{x, \neg x\}$; accordingly, h is either g_1 or $\neg g_1$. So $D_w(f, [r_1 - 1]) = r_1 - 1$ and $\text{Rank}(h) = \text{Rank}(g_1) = r_1$, and the inequality holds.

For the inductive step, when $n > 1$, we proceed by induction on $M = \sum_{i=1}^{n} m_i$.

In the base case, $M = n$, and each m_i is equal to 1. Since all g_i's are non-constant, $r_i = 1$ for all i. So $D_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 = D_w(f, [0, 0, ..., 0]) + 1 = 1$. Since all r_i's are 1, each g_i's is either $x_{i,1}$ or $\neg x_{i,1}$, Thus h is the same as f upto renaming of the literals. Hence $\text{Rank}(h) = \text{Rank}(f) \geq 1$.

For the inductive step, $M > n > 1$. Take a rank-optimal decision tree T_h for h. We want to show that $\text{Depth}_w(f, [r_1 - 1, \dots, r_n - 1]) \leq \text{Rank}(T_h) - 1$. Without loss of generality, let $x_{1,1}$ be the variable queried at the root. Let T_0 and T_1 be the left and the right subtree of T_h. For $b \in \{0, 1\}$, let g_1^b be the subfunction of g_1 when $x_{1,1}$ is set to b. Note that T_b computes $h_b \triangleq f \circ (g_1^b, g_2, ..., g_n)$, a function on $M - 1$ variables. We would like to use induction to deduce information about $\text{Rank}(T_b)$. However, g_1^b may be a constant function, and then induction does not apply. So we do a case analysis on whether or not g_1^0 and g_1^1 are constant functions; this case analysis is lengthy and tedious but most cases are straightforward.

- Case 1: Both g_1^0 and g_1^1 are constant functions. Since g_1 is non-constant, $g_1^0 \neq g_1^1$, and $r_1 = \text{Rank}(g_1) = 1$. Assume that $g_1^0 = 0$ and $g_1^1 = 1$; the argument for the other case is identical. For $b \in \{0, 1\}$, let f_b be the function $f(b, x_2, \dots, x_n)$; then $h_b = f_b \circ (g_2, \dots, g_n)$. View f_b as functions on $n - 1$ variables.
  - Case 1a: Both f_0 and f_1 are constant functions. Then f is either x_1 or $\neg x_1$, so $\text{Depth}_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) = \text{Depth}_w(f, [0, r_2 - 1, ..., r_n - 1]) = 0$. Also, in this case, h is either $x_{1,1}$ or $\neg x_{1,1}$, so $\text{Rank}(h) = 1$. Hence the inequality holds.
  - Case 1b: Exactly one of f_0 and f_1 is a constant function; without loss of generality, let f_0 be a constant function. First, observe that for any weights $w_2, \dots, w_n$, $D_w(f, [0, w_2, ..., w_n]) \leq D_w(f_1, [w_2, ..., w_n])$: we can obtain a decision tree for f witnessing this by first querying x_1, making the $x_1 = 0$ child a leaf labeled f_0, and attaching the optimal tree for f_1 on the $x_1 = 1$ branch. Second, note that since f_1 and all g_i are non-constant, so is h_1. Now

$$\begin{aligned}
\text{Rank}(h) &= \text{Rank}(h_1) && \text{since Rank}(h_0) = 0 \\
&\geq D_w(f_1, [r_2 - 1, ..., r_n - 1]) + 1 && \text{by induction hypothesis on } n \\
&\geq D_w(f, [0, r_2 - 1, ..., r_n - 1]) + 1 && \text{by first observation above} \\
&= D_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 && \text{since } r_1 = 1
\end{aligned}$$

  - Case 1c: Both f_0 and f_1 are non-constant functions.

$$\begin{aligned}
\text{Rank}(h) &\geq \max(\text{Rank}(h_0), \text{Rank}(h_1)) \\
&\geq \max_{b \in \{0,1\}} \{D_w(f_b, [r_2 - 1, ..., r_n - 1])\} + 1 && \text{by induction hypothesis on } n \\
&\geq D_w(f, [0, r_2 - 1, ..., r_n - 1]) + 1 && \text{by def. of weighted depth} \\
& && \text{of a tree querying } x_1 \text{ first} \\
&= D_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 && \text{since } r_1 = 1
\end{aligned}$$

- Case 2: One of g_1^0 and g_1^1 is a constant function; assume without loss of generality that g_1^0 be constant. In this case, we can conclude that $\text{Rank}(g_1) = \text{Rank}(g_1^1)$: $\text{Rank}(g_1^1) \leq \text{Rank}(g_1)$ by Proposition 2.4, and $\text{Rank}(g_1) \leq \text{Rank}(g_1^1)$ as witnessed by a decision tree for g_1 that queries $x_{1,1}$ first, sets the $x_{1,1} = 0$ branch to a leaf labeled g_1^0, and attaches an optimal tree for g_1^1 on the other branch. Now

$$\begin{aligned} \text{Rank}(h) &\geq \text{Rank}(h_1) \\ &\geq D_w(f, [\text{Rank}(g_1^1) - 1, r_2 - 1, ..., r_n - 1]) + 1 \quad \text{by induction on } M \\ &= D_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 \quad \text{since } \text{Rank}(g_1^1) = \text{Rank}(g_1) \end{aligned}$$

- Case 3: Both g_1^0 and g_1^1 are non-constant functions. Let $r_1^b = \text{Rank}(g_1^b) \geq 1$. A decision tree for g_1 that queries $x_{1,1}$ first and then uses optimal trees for g_1^0 and g_1^1 has rank $R \geq r_1$ and witnesses that $1 + \max\{r_1^0, r_1^1\} \geq R \geq r_1$. (Note that R may be more than r_1, since a rank-optimal tree for g_1 may not query $x_{1,1}$ first.)
  - Case 3a: $\max_b\{r_1^b\} = r_1 - 1$. Then $R = 1 + \max\{r_1^0, r_1^1\}$, which can only happen if $r_1^0 = r_1^1$, and hence $r_1^0 = r_1^1 = r_1 - 1$. We can further conclude that $r_1 \geq 2$. Indeed, if $r_1 = 1$, then $r_1 - 1 = r_1^0 = r_1^1 = 0$, contradicting the fact that we are in Case 3. For $b \in \{0, 1\}$,

$$\begin{aligned} \text{Rank}(h_b) &= \text{Rank}(f \circ (g_1^b, g_2, \ldots, g_n)) \\ &\geq \text{Depth}_w(f, [r_1^b - 1, r_2 - 1, \ldots, r_n - 1]) + 1 \quad \text{by induction on } M \\ &= \text{Depth}_w(f, [r_1 - 2, r_2 - 1, \ldots, r_n - 1]) + 1 \quad \text{since } r_1 - 1 = r_1^b. \end{aligned}$$

$$\begin{aligned} \text{Hence } \text{Rank}(h) &\geq 1 + \min_b \text{Rank}(h_b) \\ &\geq \text{Depth}_w(f, [r_1 - 2, r_2 - 1, \ldots, r_n - 1]) + 2 \quad \text{derivation above} \\ &\geq \text{Depth}_w(f, [r_1 - 1, r_2 - 1, \ldots, r_n - 1]) + 1 \quad \text{by Fact 2.1} \end{aligned}$$

  - Case 3b: $\max_b\{r_1^b\} > r_1 - 1$. So $\max_b\{r_1^b\} \geq r_1$.

$$\begin{aligned} \text{Rank}(h) &\geq \max_b \text{Rank}(h_b) \\ &\geq \max_b \text{Depth}_w(f, [r_1^b - 1, r_2 - 1, \ldots, r_n - 1]) + 1 \quad \text{by induction on } M \\ &\geq \text{Depth}_w(f, [r_1 - 1, r_2 - 1, \ldots, r_n - 1]) + 1 \quad \text{since } \max_b\{r_1^b\} \geq r_1 \end{aligned}$$

This completes the inductive step for $M > n > 1$ and completes the entire proof. ◀

From Theorems 4.1, 6.3, and 6.5, we obtain the following:

▶ **Theorem 6.6.** *For non-constant boolean functions f, g,*

$$Depth(f)(Rank(g) - 1) + 1 \leq Rank(f \circ g) \leq Depth(f)Rank(g).$$

Both inequalities are tight; the first for $\text{AND}_n \circ \text{PARITY}_m$ *and the second for* TRIBES_n *and* TRIBES_n^d.

Since any non-constant symmetric function is evasive (Proposition 2.2), from Theorems 6.3 and 6.5, we obtain the following:

▶ **Corollary 6.7.** *For non-constant boolean functions $g_1, \ldots, g_n$ with $Rank(g_i) = r_i$, and for n-variate symmetric non-constant booolean function f,*

$$\sum_i r_i - (n - 1) \leq Rank(f \circ (g_1, g_2, ..., g_n)) \leq \sum_i r_i.$$

Using Theorem 6.6, we can now complete the proof of Lemma 5.8.

Proof. (of Lemma 5.8) Consider the composed function $f = \text{MAJ}_{2k+1} \circ \text{MAJ}_{2k+1}$. Note that from the lower bound in Theorem 6.6, and the entries in Table 1, $\text{Rank}(\text{MAJ}_{2k+1} \circ \text{MAJ}_{2k+1}) \geq (2k+1)k + 1$. On the other hand, it is straightforward to verify that $\text{C}(f) = (k+1)^2$. Thus for $k > 1$, $\text{Rank}(f) > \text{C}(f)$. ◀

7 Tightness of Rank and Size relation

In Proposition 2.5, we saw a relation between rank and size. The relationship is essentially tight. The function $f = \textsc{Parity}_n$ witnesses the tightness of both the inequalities. Since $\text{Rank}(\textsc{Parity}) = n$, Proposition 2.5 tells us that $\log \text{DTSize}(\textsc{Parity})$ lies in the range $[n, n \log e]$, and we know that $\log \text{DTSize}(\textsc{Parity}) = n$.

For the $\textsc{Tribes}_n$ function, which has $N = n^2$ variables, we know from Theorem 4.1 that $\text{Rank}(\textsc{Tribes}_n) = n$. Thus Proposition 2.5 tells us that $\log \text{DTSize}(\textsc{Tribes}_n)$ lies in the range $[n, n \log(en)]$. (See also Exercise 14.9 [10] for a direct argument showing $n \leq \log \text{DTSize}(\textsc{Tribes}_n)$). But that still leaves a $(\log(en))$-factor gap between the two quantities. We show that the true value is closer to the upper end. To do this, we establish a stronger size lower bound for decision trees computing $\textsc{Tribes}^d_n$.

▶ **Lemma 7.1.** *For every $n, m \geq 1$, every decision tree for $\textsc{Tribes}^d_{n,m}$ has at least m^n 1-leaves and n 0-leaves.*

Proof. Recall that $\textsc{Tribes}^d_{n,m} = \bigwedge_{i \in [n]} \bigvee_{j \in [m]} x_{i,j}$. We call $x_{i,1}, x_{i,2}, \ldots, x_{i,m}$ the ith block of variables. We consider two special kinds of input assignments: 1-inputs of minimum Hamming weight, call this set S_1, and 0-inputs of maximum Hamming weight, call this set S_0. Each $a \in S_1$ has exactly one 1 in each block; hence $|S_1| = m^n$. Each $b \in S_0$ has exactly m zeroes, all in a single block; hence $|S_0| = n$. We show that in any decision tree T for $\textsc{Tribes}^d_{n,m}$, all the inputs in $S = S_1 \cup S_0$ go to pairwise distinct leaves. Since all inputs in S_1 must go to 1-leaves of T, and all inputs of S_0 must go to 0-leaves, this will prove the claimed statement.

Let a, b be distinct inputs in S_1. Then there is some block $i \in [n]$, where they differ. In particular there is a unique $j \in [m]$ where $a_{i,j} = 1$, and at this position, $b_{i,j} = 0$. The decision tree T must query variable $x_{i,j}$ on the path followed by a, since otherwise it will reach the same 1-leaf on input a' that differs from a at only this position, contradicting the fact that $\textsc{Tribes}^d_{n,m}(a') = 0$. Since $b_{i,j} = 0$, the path followed in T along b will diverge from a at this query, if it has not already diverged before that. So a, b reach different 1-leaves.

Let a, b be distinct inputs in S_0. Let i be the unique block where a has all zeroes; b has all 1s in this block. On the path followed by a, T must query all variables from this block, since otherwise it will reach the same 0-leaf on input a'' that differs from a only at an unqueried position in block i, contradicting $\textsc{Tribes}^d_{n,m}(a'') = 1$. Since a and b differ everywhere on this block, b does not follow the same path as a, so they go to different leaves of T. ◀

We thus conclude that the second inequality in Proposition 2.5 is also essentially tight for the $\textsc{Tribes}^d_n$ function.

The size lower bound from Lemma 7.1 can also be obtained by specifying a good Delayer strategy in the asymmetric Prover-Delayer game and invoking Proposition 3.2.

8 Proofs using Prover-Delayer Games

In this section we illustrate proving rank upper and lower bounds by giving Prover-Delayer Game based proofs for Lemma 4.6, Lemma 4.7 and Lemma 5.8. Theorem 3.1 gives us a way to prove rank upper and lower bounds for boolean functions. In a Prover-Delayer game for R_f, exhibiting a Prover strategy which restricts the Delayer to at most r points gives an upper bound of r on $\text{Rank}(f)$. Similarly, exhibiting a Delayer strategy which scores at least r points irrespective of the Prover strategy shows a lower bound of r on $\text{Rank}(f)$.

Prover strategy for $\textsc{And}_n \circ \textsc{Parity}_m$, proving Lemma 4.6

We give a Prover strategy which restricts Delayer to $n(m-1)+1$ points. The Prover queries variables in row-major order. If on query $x_{i,j}$ the Delayer defers a decision to the Prover, the Prover chooses arbitrarily unless $j = m$. If $j = m$, then the Prover chooses a value which makes the parity of the variables in row i evaluate to 0.

Let j be the first row such that the Delayer defers the decision on $x_{j,m}$ to the Prover. (If there is no such row, set $j = n$.) With the strategy above, the Prover will set $x_{j,m}$ in such a way that the parity of the variables in j-th row evaluates to 0, making f evaluate to 0 and ending the game. The Delayer scores at most $m-1$ points per row for rows before this row j, and at most m points in row j. Hence the Delayer's score is at most $(j-1)(m-1)+m$ points. Since $j \leq n$, the Delayer is restricted to $n(m-1)+1$ points at the end of the game.

Delayer strategy for $\textsc{And}_n \circ \textsc{Parity}_m$, proving Lemma 4.7

We give a Delayer strategy which always scores at least $n(m-1)+1$ points.

On query $x_{i,j}$, if this is the last un-queried variable, or if there is some un-queried variable in the same i-th row, the Delayer defers the decision to the Prover. Otherwise the Delayer responds with a value that makes the parity of the variables in row i evaluate to 1.

This strategy forces the Prover to query all variables to decide the function. The Delayer picks up $m-1$ points per row, and an additional point on the last query, giving a total score of $n(m-1)+1$ points.

Delayer strategy for $f = \textsc{Maj}_{2k+1} \circ \textsc{Maj}_{2k+1}$, proving Lemma 5.8

The following Delayer strategy always scores $(k+1)^2+k^2$ points, greater than $C(f) = (k+1)^2$.

At an intermediate stage of the game, say that a row is b-determined if the variables that are already set in this row already fix the value of $\textsc{Maj}_{2k+1}$ on this row to be b, and is determined if it is b-determined for some b. Let M_b be the number of b-determined rows. If the game has not yet ended, then $M_0 \leq k$ and $M_1 \leq k$.

On query $x_{i,j}$, let n_0, n_1 be the number of variables in row i already set to 0 and to 1 respectively. The Delayer defers the decision if

- row i is already determined, or
- $n_0 = n_1 < k$, or
- $n_0 = n_1 = k$ and $M_0 = M_1$.

Otherwise, if $n_0 \neq n_1$, then the Delayer chooses the value b where $n_b < n_{1-b}$. If $n_0 = n_1 = k$, then the Delayer chooses the value b where $M_b < M_{1-b}$.

This strategy ensures that at all stages until the game ends, $|M_0 - M_1| \leq 1$, and furthermore, in all rows that are not yet determined, $|n_0 - n_1| \leq 1$. Thus a row becomes determined only after all variables in it are queried, and the Delayer gets a point for every other query, making a total of k points per determined row. Further, for $k+1$ rows, the Delayer also gets an additional point on the last queried variable. The game cannot conclude before all $2k+1$ rows are determined, so the Delayer scores at least $(k+1)^2 + k^2$ points.

9 Conclusion

The main thesis of this paper is that the minimal rank of a decision tree computing a Boolean function is an interesting measure for the complexity of the function, since it is not related to other well-studied measures in a dimensionless way. Whether bounds on this measure can be further exploited in algorithmic settings like learning or sampling remains to be seen.

References

1 James Aspnes, Eric Blais, Murat Demirbas, Ryan O'Donnell, Atri Rudra, and Steve Uurtamo. k $^{+}$ decision trees - (extended abstract). In *6th International Workshop on Algorithms for Sensor Systems, Wireless Ad Hoc Networks, and Autonomous Mobile Entities, ALGOSENSORS*, volume 6451 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 2010. full version on author's webpage, http://www.cs.cmu.edu/ odonnell/papers/k-plus-dts.pdf. doi:10.1007/978-3-642-16988-5_7.

2 Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24(4):585–603, 2004. doi:10.1007/s00493-004-0036-5.

3 Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. A characterization of tree-like resolution size. *Information Processing Letters*, 113(18):666–671, 2013. doi:10.1016/j.ipl.2013.06.002.

4 Avrim Blum. Rank-r decision trees are a subclass of r-decision lists. *Information Processing Letters*, 42(4):183–185, 1992. doi:10.1016/0020-0190(92)90237-P.

5 Manuel Blum and Russell Impagliazzo. Generic oracles and oracle classes. In *28th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 118–126. IEEE, 1987.

6 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.

7 Javier Esparza, Michael Luttenberger, and Maximilian Schlund. A brief history of Strahler numbers. In *Language and Automata Theory and Applications - 8th International Conference LATA*, volume 8370 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2014. doi:10.1007/978-3-319-04921-2_1.

8 Juan Luis Esteban and Jacobo Torán. A combinatorial characterization of treelike resolution space. *Information Processing Letters*, 87(6):295–300, 2003.

9 Juris Hartmanis and Lane A Hemachandra. One-way functions and the nonisomorphism of NP-complete sets. *Theoretical Computer Science*, 81(1):155–163, 1991.

10 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.

11 Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. *Electron. Colloquium Comput. Complex.*, 41, 1999. URL: http://eccc.hpi-web.de/eccc-reports/1999/TR99-041/index.html.

12 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

13 Ashley Montanaro. A composition theorem for decision tree complexity. *Chicago Journal of Theoretical Computer Science*, 2014(6), July 2014.

14 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, 2000.

15 Gábor Tardos. Query complexity, or why is it difficult to separate $\mathrm{NP}^A \cap \mathrm{coNP}^A$ from P^A by random oracles A? *Combinatorica*, 9(4):385–392, 1989.

16 György Turán and Farrokh Vatan. Linear decision lists and partitioning algorithms for the construction of neural networks. In *Foundations of Computational Mathematics*, pages 414–423, Berlin, Heidelberg, 1997. Springer.

17 Kei Uchizawa and Eiji Takimoto. Lower bounds for linear decision trees with bounded weights. In *41st International Conference on Current Trends in Theory and Practice of Computer Science SOFSEM*, volume 8939 of *Lecture Notes in Computer Science*, pages 412–422. Springer, 2015. doi:10.1007/978-3-662-46078-8_34.

18 Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. doi:10.1145/1968.1972.

Reachability and Matching in Single Crossing Minor Free Graphs

Samir Datta
Chennai Mathematical Institute, Chennai, India

Chetan Gupta
Aalto University, Finland

Rahul Jain
Fernuniversität in Hagen, Germany

Anish Mukherjee
Institute of Informatics, University of Warsaw, Poland

Vimal Raj Sharma
Indian Institute of Technology, Kanpur, India

Raghunath Tewari
Indian Institute of Technology, Kanpur, India

Abstract

We show that for each single crossing graph H, a polynomially bounded weight function for all H-minor free graphs G can be constructed in logspace such that it gives nonzero weights to all the cycles in G. This class of graphs subsumes almost all classes of graphs for which such a weight function is known to be constructed in logspace. As a consequence, we obtain that for the class of H-minor free graphs where H is a single crossing graph, reachability can be solved in UL, and bipartite maximum matching can be solved in SPL, which are small subclasses of the parallel complexity class NC. In the restrictive case of bipartite graphs, our maximum matching result improves upon the recent result of Eppstein and Vazirani [16], where they show an NC bound for constructing perfect matching in general single crossing minor free graphs.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases Reachability, Matching, Logspace, Single-crossing minor free graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.16

Funding *Samir Datta*: Partially funded by a grant from Infosys foundation and SERB-MATRICS grant MTR/2017/000480.
Chetan Gupta: Supported by Academy of Finland, Grant 321901.
Anish Mukherjee: Supported by the ERC CoG grant TUgbOAT no 772346.
Vimal Raj Sharma: Ministry of Electronics and IT, India, VVS PhD program.
Raghunath Tewari: Young Faculty Research Fellowship, Ministry of Electronics and IT, India.

1 Introduction

Directed graph reachability and perfect matching are two fundamental problems in computer science. The history of the two problems has been inextricably linked together from the inception of computer science (and before!) [18]. The problems and their variants, such as shortest path [13] and maximum matching [14] have classically been studied in the sequential model of computation. Since the 1980s, considerable efforts have been spent trying to find parallel algorithms for matching problems spurred on by the connection to reachability which is, of course, parallelizable. The effort succeeded only in part with the discovery of randomized parallel algorithms [22, 27]. While we know that the reachability problem is complete for the complexity class NL, precise characterization has proved to be elusive for matching problems.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 16; pp. 16:1–16:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The 1990s saw attempts in this direction when surprisingly "small" upper bounds were proved [2] for the perfect matching problem, although in the non-uniform setting. At roughly the same time, parallel algorithms for various versions of the matching problem for restricted graph classes like planar [26] and bounded genus [25] graphs were discovered. The last two decades have seen efforts towards pinning down the exact parallel complexity of reachability and matching related problems in restricted graph classes [6, 24, 9, 10, 5, 21, 19, 20]. Most of these papers are based on the method of constructing *nonzero circulations.*

The circulation of a simple cycle is the sum of its edge-weights in a fixed orientation (see Section 2 for the definition) and we wish to assign polynomially bounded weights to the edges of a graph, such that every simple cycle has a nonzero circulation. Assigning such weights *isolates* a reachability witness or a matching witness in the graph [33]. Constructing polynomially bounded isolating weight function in parallel for general graphs has been elusive so far. The last five years have seen rapid progress in the realm of matching problems, starting with [17] which showed that the method of nonzero circulations could be extended from topologically restricted (bipartite) graphs to general (bipartite) graphs. A subsequent result extended this to all graphs [32]. More recently, the endeavour to parallelize planar perfect matching has borne fruit [30, 4] and has been followed up by further exciting work [3].

We know that polynomially bounded weight functions that give nonzero circulation to every cycle can be constructed in logspace for planar graphs, bounded genus graphs and bounded treewidth graphs [6, 10, 11] . Planar graphs are both $K_{3,3}$-free and K_5-free graphs. Such a weight function is also known to be constructable in logspace for $K_{3,3}$-free graphs and K_5-free graphs, individually [5]. A natural question arises if we can construct such a weight function for H-minor-free graphs for any arbitrary graph H. A major hurdle in this direction is the absence of a space-efficient (Logspace) or parallel algorithm (NC) for finding a structural decomposition of H-minor free graphs. However, such a decomposition is known when H is a single crossing graph. This induces us to solve the problem for single crossing minor-free (SCM-free) graphs. An SCM-free graph can be decomposed into planar and bounded treewidth graphs. Moreover, $K_{3,3}$ and K_5 are single crossing graphs. Hence our result can also be seen as a generalization of the previous results on these classes. There have also been important follow-up works on parallel algorithms for SCM-free graphs [16]. SCM-free graphs have been studied in several algorithmic works (for example [31, 7, 12]).

1.1 Our Result

In this paper, we show that results for previously studied graph classes (planar, constant tree-width and H-minor free for $H \in \{K_{3,3}, K_5\}$) can be extended and unified to yield similar results for SCM-free graphs.

▶ **Theorem 1.** *There is a logspace algorithm for computing polynomially-bounded, skew-symmetric nonzero circulation weight function in SCM-free graphs.*

An efficient solution to the circulation problem for a class of graphs yields better complexity bounds for determining reachability in the directed version of that class and constructing minimum weight maximum-matching in the bipartite version of that class. Theorem 1 with the results of [8, 28], yields the following:

▶ **Corollary 2.** *For SCM-free graphs, reachability is in* $\mathsf{UL} \cap \mathsf{coUL}$ *and minimum weight bipartite maximum matching is in* SPL.

Also using the result of [35], we obtain that the *Shortest path* problem in SCM-free graphs can be solved in $\mathsf{UL} \cap \mathsf{coUL}$.

Overview of Our Techniques and Comparison With Previous Results. We know that for planar graphs and constant treewidth graphs nonzero circulation weights can be constructed in logspace [6, 11]. We combine these weight functions using the techniques from Arora et al. [5], Datta et al. [8] and, Datta et al. [11] together with some modifications to obtain the desired weight function. In [5], the authors decompose the given input graph G ($K_{3,3}$-free or K_5-free) and obtain a component tree that contains planar and constant size components. They modify the components of the component tree so that they satisfy few properties which they use for constructing nonzero circulation weights (these properties are mentioned at the beginning of Section 3). The new graph represented by these modified components preserves the perfect matchings of G. Then, they construct a *working-tree* of height $O(\log n)$ corresponding to this component tree and use it to assign nonzero circulation weights to the edges of this new graph. The value of the weights assigned to the edges of the new graph is exponential in the height of the working tree.

While $K_{3,3}$-free and K_5-free graphs can be decomposed into planar and constant size components, an SCM-free graph can be decomposed into planar and constant treewidth components. Thus the component tree of the SCM-free graph would have several non-planar constant treewidth components. While we can construct a working tree of height $O(\log n)$, this tree would contain constant-treewidth components and hence make it difficult to find nonzero circulation weights. A naïve idea would be to replace each constant treewidth component with its tree decomposition in the working tree. However, the resultant tree would have the height $O(\log^2 n)$. Thus the weight function obtained in this way is of $O(\log^2 n)$-bit. We circumvent this problem as follows: we obtain a component tree T of the given SCM-free graph G and modify its components to satisfy the same property as [5] (however, we use different gadgets for modification). Now we replace each bounded treewidth component with its tree decomposition in T. Using this new component tree, say T', we define another graph G'. We use the technique from [8] to show that if we can construct the nonzero circulation for G', then we can *pull back* nonzero circulation for G. Few points to note here: (i) pull back technique works because of the new gadget that we use to modify the components in T, (ii) since ultimately we can obtain nonzero circulation for G, it allows us to compute maximum matching in G in SPL, which is not the case in [5].

1.2 Organization of the Paper

After introducing the definitions and preliminaries in Section 2, in Section 3 we discuss the weight function that achieves non-zero circulation in single-crossing minor free graphs and its application to maximum matching in Section 4. Finally, we conclude with Section 5.

2 Preliminaries and Notations

Tree decomposition. Tree decomposition is a well-studied concept in graph theory. Tree decomposition of a graph, in some sense, reveals the information of how much tree-like the graph is. We use the following definition of tree decomposition.

▶ **Definition 3.** *Let $G(V,E)$ be a graph and $\tilde{T}$ be a tree, where nodes of the $\tilde{T}$ are $\{B_1,\ldots,B_k \mid B_i \subseteq V\}$ (called bags). T is called a tree decomposition of G if the following three properties are satisfied:*

- $B_1 \cup \ldots \cup B_k = V$,
- *for every edge $(u,v) \in E$, there exists a bag B_i which contains both the vertices u and v,*
- *for a vertex $v \in V$, the bags which contain the vertex v form a connected component in $\tilde{T}$.*

The width of a tree decomposition is defined as one less than the size of the largest bag. The treewidth of a graph G is the minimum width among all possible tree decompositions of G. Given a constant treewidth graph G, we can find its tree decomposition $\tilde{T}$ in logspace such that $\tilde{T}$ has a constant width [15].

▶ **Lemma 4** ([15]). *For every constant c, there is a logspace algorithm that takes a graph as input and outputs its tree decomposition of treewidth at most c, if such a decomposition exists.*

▶ **Definition 5.** *Let G_1 and G_2 be two graphs containing cliques of equal size. Then the* clique-sum *of G_1 and G_2 is formed from their disjoint union by identifying pairs of vertices in these two cliques to form a single shared clique, and then possibly deleting some of the clique edges.*

For a constant k, a k-clique-sum is a clique-sum in which both cliques have at most k vertices. One may also form clique-sums of more than two graphs by repeated application of the two-graph clique-sum operation. For a constant W, we use the notation $\langle \mathcal{G}_{P,W} \rangle_k$ to denote the class of graphs that can be obtained by taking repetitive k-clique-sum of planar graphs and graphs of treewidth at most W. In this paper, we construct a polynomially bounded skew-symmetric weight function that gives nonzero circulation to all the cycles in a graph $G \in \langle \mathcal{G}_{P,W} \rangle_3$. Note that if a weight function gives nonzero circulations to all the cycles in the biconnected components of G, it will give nonzero circulation to all the cycles in G because no simple cycle can be a part of two different biconnected components of G. We can find all the biconnected components of G in logspace by finding all the articulation points. Therefore, without loss of generality, assume that G is biconnected.

The *crossing number* of a graph G is the lowest number of edge crossings of a plane drawing of G. A *single-crossing* graph is a graph whose crossing number is at most 1. SCM-free graphs are graphs that do not contain H as a minor, where H is a fixed single crossing graph. Robertson and Seymour have given the following characterization of SCM-free graphs.

▶ **Theorem 6** ([29]). *For any single-crossing graph H, there is an integer $c_H \geq 4$ (depending only on H) such that every graph with no minor isomorphic to H can be obtained as 3-clique-sum of planar graphs and graphs of treewidth at most c_H.*

Component Tree. In order to construct the desired weight function for a graph $G \in \langle \mathcal{G}_{P,W} \rangle_3$, we decompose G into smaller graphs and obtain a component tree of G defined as follows: we first find 3-connected and 4-connected components of G such that each of these components is either planar or of constant treewidth. We know that these components can be obtained in logspace [34]. Since G can be formed by taking repetitive 3-clique-sum of these components, the set of vertices involved in a clique-sum is called a separating set. Using these components and separating sets, we define a component tree of G. A component tree T of G is a tree such that each node of T contains a 3-connected or 4-connected component of G, i.e., each node contains either a planar or constant treewidth subgraph of G. There is an edge between two nodes of T if the corresponding components are involved in a clique-sum operation. If two nodes are involved in a clique-sum operation, then copies of all the vertices of the clique are present in both components. It is easy to see that T will always be a tree. Within a component, there are two types of edges present, *real* and *virtual edges.* Real edges are those edges that are present in G. Let $\{a, b, c\}$(or $\{a, b\}$) be a separating triplet(or pair) shared by two nodes of T, then there is a clique $\{a, b, c\}$ (or $\{a, b\}$) of virtual edges present in both the components. Suppose there is an edge present in G between any pair of vertices of a separating set. In that case, there is a real edge present between that pair of vertices parallel to the virtual edge, in exactly one of the components which share that separating set.

Weight function and circulation. Let $G(V, E)$ be an undirected graph with vertex set V and edge set E. By $\overleftrightarrow{E}$, we denote the set of bidirected edges corresponding to E. Similarly, by $G(V, \overleftrightarrow{E})$, we denote the graph corresponding to $G(V, E)$ where each of its edges is replaced by a corresponding bidirected edge. A weight function $w : \overleftrightarrow{E} \to \mathbb{Z}$ is called skew-symmetric if for all $e \in \overleftrightarrow{E}$, $w(e) = -w(e^r)$ (where e^r represent the edge with its direction reversed). We know that if w gives nonzero circulation to every cycle that consists of edges of $\overleftrightarrow{E}$ then it isolates a directed path between each pair of vertices in $G(V, \overleftrightarrow{E})$. Also, if G is a bipartite graph, then the weight function w can be used to construct a weight function $w^{\text{und}} : E \to \mathbb{Z}$ that isolates a perfect matching in G [33].

A convention is to represent by $\langle w_1, \dots, w_k\rangle$ the weight function that on edge e takes the weight $\sum_{i=1}^{k} w_i(e)B^{k-i}$ where $w_1, \dots, w_k$ are weight functions such that $\max_{i=1}^{k}(nw_i(e)) \leq B$.

Complexity Classes. The complexity classes L and NL are the classes of languages accepted by deterministic and non-deterministic logspace Turing machines, respectively. UL is a class of languages that can be accepted by an NL machine that has at most one accepting path on each input, and hence $\mathsf{UL} \subseteq \mathsf{NL}$. SPL is the class of languages whose characteristic function can be written as a logspace computable integer determinant.

3 Weight function

In order to construct the desired weight function for a given graph $G_0 \in \langle \mathcal{G}_{P,W}\rangle_3$, we modify the component tree T_0 of G_0 such that it has the following properties.

- No two separating sets share a common vertex.
- A separating set is shared by at most two components.
- any virtual triangle, i.e., the triangle consists of virtual edge, in a planar component is always a face.

Let T be this modified component tree, and G be the graph represented by T. We show that if we have a weight function that gives nonzero circulation to every cycle in G, then we can obtain a weight function that will give nonzero circulation to all the cycles in G_0. Arora et al. [5] showed how a component tree satisfying these properties can be obtained for $K_{3,3}$-free and K_5-free graphs. We give a similar construction below and show that we can modify the components of T_0 such that T satisfies the above properties (see Section 3.1). Note that if the graphs inside two nodes of T_0 share a separating set τ and they both are constant tree-width graphs, then we can take the clique-sum of these two graphs on the vertices of τ, and the resulting graph will also be a constant tree-width graph. Therefore, we can assume that if two components share a separating set, then either both of them are planar, or one of them is planar and the other is of constant tree-width.

3.1 Modifying the Component Tree

In this section, we show that how we obtain the component tree T from T_0 so that it satisfies the above three properties.

(i) No two separating sets share a common vertex. For a node D in T_0, let G_D be the graph inside node D. Assume that G_D contains a vertex v which is shared by separating sets $\tau_1, \tau_2, \dots, \tau_k$, where $k > 1$, present in G_D. We replace the vertex v with a gadget γ defined as follows: γ is a star graph such that v is the center node and $v_1, , v_2, \dots, v_k$ are the leaf

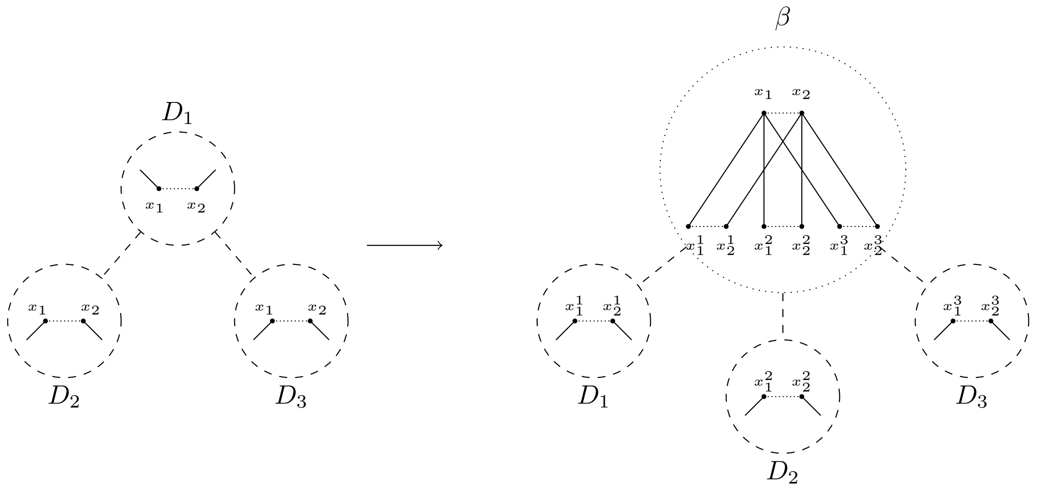


Figure 1 (Left)A separating set $\{x_1, x_2\}$ is shared by components D_1, D_2 and D_3. (Right) Replace them by adding the gadget β and connect D_1, D_2 and D_3 to β.

nodes of γ. The edges which were incident on v and had their other endpoints in τ_i, will now incident on v_i for all $i \in [k]$. All the other edges which were incident on v will continue to be incident on v. We do this for each vertex which is shared by more than one separating set in G_D. Let $G_{D'}$ be the graph obtained after replacing each such vertex with gadget γ. It is easy to see that if G_D was a planar component, then $G_{D'}$ will also be a planar component. We show that the same holds for constant tree-width components as well.

▷ Claim 7. If G_D is a constant treewidth graph, then $G_{D'}$ will also be of constant treewidth.

Proof. Let T_D be a tree decomposition of G_D such that each bag of T_D is of constant size, i.e., contains some constant number of vertices. Let v be a vertex shared by k separating sets $\{x_i, y_i, v\}$, for all $i \in [k]$ in G_D. Let $B_1, B_2, \ldots B_k$ be the bags in T_D that contain separating sets $\{x_1, y_1, v\}, \{x_2, y_2, v\}, \ldots, \{x_k, y_k, v\}$ respectively (note that one bag may contain many separating sets). Now we obtain a tree decomposition $T_{D'}$ of the graph $G_{D'}$ using T_D as follows: add the vertices v_i in the bag B_i, for all $i \in [k]$. Repeat this for each vertex v in G_D, which is shared by more than one separating set to obtain $T_{D'}$. Note that in each bag of T_D we add at most one new vertex with respect to each separating set contained in the bag in order to obtain $T_{D'}$. Since each bag in T_D can contain vertices of only constant many separating sets, size of each bag remain constant in $T_{D'}$. Also, $T_{D'}$ is a tree decomposition of $G_{D'}$. ◁

(ii) A separating set is shared by at most two components. Assume that a separating set of size t, $\tau = \{x_i\}_{i \le t}$ is shared by k components $D_1, D_2, \ldots D_k$, for $k > 2$, in T_0. Let β be a gadget defined as follows: the gadget consists of t star graphs $\{\gamma_i\}_{i \le t}$ such that x_i is the center node of γ_i and each γ_i has k leaf nodes $\{x_i^1, x_i^2, \ldots x_i^k\}$. There are virtual cliques present among the vertices $\{x_i^j\}_{i \le t}$ for all $j \in [k]$ and among $\{x_i\}_{i \le t}$ (see Figure 1). If there is an edge present between any pair of vertices in the set $\{x_i\}_{i \le t}$ in the original graph, then we add a real edge between respective vertices in β. β shares the separating set $\{x_i^j\}_{i \le t}$ with the component D_j for all $j \in [k]$.

Note that in this construction, we create new components (β) while all the other components in the component tree remain unchanged. Notice that the tree-width of β is constant (at most 5 to be precise). We can define a tree decomposition of β of tree-width 5 as follows: $B_0, B'_1, B'_2, \ldots, B'_k$ be the bags in the tree decomposition such that $B_0 = \{x_1, x_2, x_3\}$, $B'_i = \{x_1, x_2, x_3, x^i_1, x^i_2, x^i_3\}$ and there is an edge from B_0 to B'_i for all $i \in [k]$.

(iii) Any virtual triangle, i.e., the triangle consists of virtual edges, in a planar component is always a face. 3-cliques in a 3-clique sum of a planar and a bounded tree-width component is always a face in the planar component. This is because suppose there is a planar component G_i in which the 3-clique on u, v, w occurs but does not form a face. Then the triangle u, v, w is a separating set in G_i, which separates the vertices in its interior V_1 from the vertices in its exterior V_2. Notice that neither of V_1, V_2 is empty by assumption since u, v, w is not a face. However, then we can decompose G_i further.

3.2 Preserving nonzero circulation

We can show that if we replace a vertex with the gadget γ, then the nonzero-circulation in the graph remains preserved: let $G_1(V_1, E_1)$ be a graph such that a vertex v in G_1 is replaced with the gadget γ (star graph). Let this new graph be $G_2(V_2, E_2)$. We show that if we have a skew-symmetric weight function w_2 that gives nonzero circulation to every cycle in $G_2(V_2, \vec{E}_2)$, then we can obtain a skew-symmetric weight function w_1 that gives nonzero circulation to every cycle in $G_1(V_1, \vec{E}_2)$ as follow. Let $u_1, u_2, \ldots, u_k$ be the neighbors of v in G_1. For the sake of simplicity, assume that v is replaced with γ such that γ has only two leaves v_1 and v_2 and v is the center of γ. Now assume that $u_1, u_2, \ldots, u_j$ become neighbors of v_1 and, $u_{j+1}, u_{j+2}, \ldots, u_k$ become neighbors of v_2 in G_2, for some $j < k$. We define a function P that maps each edge of G_1 to at most two edges of G_2 as follows. For edge (u_i, v) in G_1,

$$P(u_i, v) = \begin{cases} \{(u_i, v_1), (v_1, v)\}, & \text{if } u_i \text{ is a neighbor of } v_1 \text{ in } G_2, \\ \{(u_i, v_2), (v_2, v)\}, & \text{if } u_i \text{ is a neighbor of } v_2 \text{ in } G_2. \end{cases}$$

For all the other edges e of G_1, $P(e) = \{e \in G_2\}$. Now given weight function w_2 for G_2, we define weight function w_1 for G_1 as follows:

$$w_1(e) = \sum_{e' \in P(e)} w_2(e')$$

For any $F \subseteq \vec{E_1}$, we define $P(F) = (P(e) \mid e \in F)$. Let C be a simple cycle in G_1. Notice that the set of edges in $P(C)$ form a walk in G_2. Also, note that for some edges e of G_2 both e and e^r may appear in $P(C)$. Let $\hat{E}_2(C)$ be the set of edges in $P(C)$ such that for each $e \in \hat{E}_2(C)$, both e and e^r appear in $P(C)$. Since our weight function is skew-symmetric, we know that $\sum_{e \in \hat{E}_2(C)} w_2(e) = 0$. Also, notice that set of edges in the set $P(C) - \hat{E}_2(C)$ form a simple cycle in G_2 (proof of this is same as the proof of Claim 9, that we prove later in this paper). Let C' be the simple cycle in G_2 formed by the edges in the set $P(C) - \hat{E}_2(C)$. We know that, $\sum_{e \in C} w_1(e) = \sum_{e \in P(C)} w_2(e) = \sum_{e \in C'} w_2(e) + \sum_{e \in \hat{E}_2(C)} w_2(e)$ and since $\sum_{e \in \hat{E}_2(C)} w_2(e) = 0$ we have, $\sum_{e \in C} w_1(e) = \sum_{e \in C'} w_2(e)$.

Therefore, we can say that if w_2 gives nonzero circulation to C', then w_1 gives nonzero circulation to C. To satisfy property 1 on the component tree, we replace vertices of the graph with γ. Furthermore, to satisfy property 2, we replace vertices with the gadget β, which contains nothing but multiple copies of γ. Thus from above, we can conclude that these constructions preserve the nonzero circulation. Now we will work with the graph G and the component tree T.

3.3 Tree decomposition

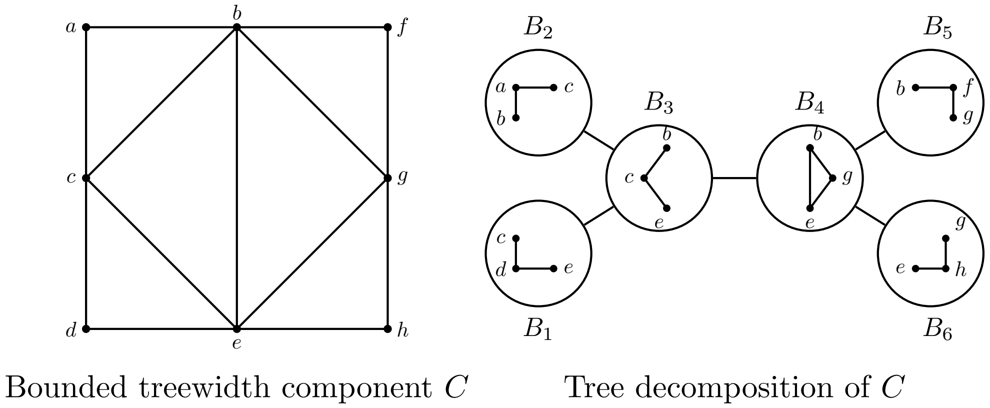
Note that the component tree T of G is also a tree decomposition of G in the sense that we can consider the nodes of the component tree as bags of vertices. We know that T contains two types of nodes: (i) nodes that contain planar graphs, (ii) nodes that contain constant tree-width graphs. We call them *p-type* and *c-type* nodes, respectively.

Now we will construct another tree decomposition T' of G using the component tree T. T' have two types of bags: (i) A bag with respect to each *p-type* node of T, which contains the same set of vertices as the *p-type* node, and (ii) bags obtained from tree decomposition of the component inside each *c-type* node. For a node N of T, let G_N denote the graph inside node N. Let $V(G_N)$ denote the vertices in the graph G_N and T_N be a tree decomposition of G_N obtained using Lemma 4.

- Bags of T' are defined as follows: $\{V(G_N) \mid N \in T$, where N is a *p-type* node in $T\} \bigcup \{B \mid B \in T_N$, where N is a *c-type* node in $T\}$. We know that in a tree decomposition of a graph H, for each clique in H, there exists a bag in the tree decomposition of H that contains all the vertices of the clique. Let τ be a separating set present in the graph contained in a *c-type* node N of T. While constructing a tree decomposition T_N, we consider the virtual clique present among the vertices of τ as a part of G_N. This ensures that there exists a bag in T_N that contains all the vertices of τ.
- Edges in T' are defined as follows: (i) for a *c-type* node N, let B and B' be two bags in T_N. If an edge in T_N connects B and B', then add an edge between them in T' as well,(ii) let N' and N'' be two adjacent nodes in T such that they share a separating set τ. As mentioned above, we know that either both N' and N'' are *p-type* or one of them is *p-type* and the other one is a *c-type* node. If both of them are *p-type* : we add an edge between the bags in T' that contain the vertices $V(G_{N'})$ and $V(G_{N''})$. If N' is a *p-type* and N'' is a *c-type* node: remember that we replaced node N'' by its tree decomposition. let B be any bag in $T_{N''}$ which contains all the vertices of τ. We add an edge between the bag containing vertices $V(G_N)$ and B.

It is easy to see that T' is also a tree decomposition of G. From Lemma 4, we can say that the overall construction of T' remains in logspace. For simplicity, we rename the bags of T' to $B_1, B_2, \ldots, B_k$. Let B_i be a bag in T' corresponding to a node N in T. If N contains a separating set τ, then the set of vertices of τ is also called a separating set in B_i. We will need this notion later on while constructing the weight function.

Note that Arora et al. [5] obtained a component tree $\hat{T}$ of an input graph $\hat{G}$ such that each node of $\hat{T}$ contains either a planar graph or a constant size graph. Then they show that for a cycle C in $\hat{G}$, the nodes which contain edges of C form a connected component of $\hat{T}$. They use this property to construct the desired weight function. Here we can say that T' is also a component tree of G such that each node of T' contains a planar graph or a constant size graph, in a sense that we assign each edge e of G to one of the bags of T', which contains both the endpoints of e. However, we cannot claim that for a cycle C, the bags containing edges of C form a connected component in T' (for example, see Figure 2). Thus in this paper, we use the tree decomposition T' to construct another graph G' and associate edges of G' to the bags to T' such that for each cycle C' in G', the bags which have edges of C' associated with them, form a connected component in T'. We show that if we can



Bounded treewidth component C

Tree decomposition of C

Figure 2 Assume C (left) is a constant treewidth component of G. We replace C with its tree decomposition (right) in the component tree T' and associated each edge of C with one of the bags in the tree decomposition, as shown in the figure. Notice that for cycle *abfghedca* the bags (B_1, B_2, B_5 and B_6), which have edges of the cycle associated with them, do not form a connected component.

construct a skew-symmetric weight function for G' such that it gives nonzero circulation to every cycle in G', then we can obtain a skew-symmetric weight function for G, which gives nonzero circulations to all the cycles in G.

3.4 Construction of G'

We construct $G'(V', E')$ from the tree decomposition T' of $G(V, E)$ as follows. We borrow notation from Datta et al. [11]. Without loss of generality, assume that T' is a rooted tree and the parent-child relationship is well defined.

- Vertex set $V' = \{v_{B_i} \mid B_i \in T', v \in B_i\}$, i.e., for each vertex v of the G, we have copies of v in G' for each bag B_i of T' in which v appears. Copies of vertices of a separating set are also called a separating set in G'.
- Edge set $E' = \{(u_{B_i}, v_{B_i}) \mid (u, v) \in E,\ u \notin parent(B_i)\ or\ v \notin parent(B_i)\} \bigcup \{(v_{B_i}, v_{B_j}) \mid B_i$ and B_j are adjacent in $T'\}$. In other words, we add an edge between the two vertices u and v of same bag B_i if there is an edge between those vertices in G and no ancestor of B_i in T' contains both the vertices u and v. We add an edge between two copies of a vertex if the bags they belong to, are adjacent in T'.

▶ **Lemma 8.** *Given a polynomially bounded, skew-symmetric weight function w' that gives a nonzero circulation to every cycle in G', we can find a polynomially bounded, skew-symmetric weight function w for G that gives a nonzero circulation to every cycle in G.*

Proof. To construct the weight function w, we associate a sequence $P(u, v)$ of edges of G' with each edge (u, v) of G. Assume that T' is a rooted tree, and root is the highest node in the tree. The heights of all the other nodes are one less than that of their parent. As we mentioned that T' is a tree decomposition of G. For an edge (u, v), we know that there are unique highest bags B_1 and B_2 that contain vertices u and v, respectively.

- If $B_1 = B_2$ then $P(u, v) = (u_{B_1}, v_{B_2})$.
- If B_1 is an ancestor of B_2 then
  $P(u, v) = (u_{B_1}, u_{parent(\cdots(parent(B_2)))}), \ldots, (u_{parent(B_2)}, u_{B_2}), (u_{B_2}, v_{B_2})$.

- If B_1 is a descendant of B_2 then
  $P(u,v) = (u_{B_1}, v_{B_1}), (v_{B_1}, v_{parent(B_1)}), \ldots, (v_{parent(\cdots(parent(B_1)))}, v_{B_2})$.

  The weight function w for the graph G is defined as follows:

  $$w(u,v) = \sum_{e \in P(u,v)} w'(e)$$

For a simple cycle $C = e_1, e_2, \ldots, e_j$ in G, we define $P(C) = P(e_1), P(e_2), \ldots, P(e_j)$. Note that $P(C)$ is a closed walk in G'. Let $E'_d(C)$ be the subset of edges of G' such for all edges $e \in E'_d(C)$ both e and e^r appear in $P(C)$, where e^r denotes the edge obtained by reversing the direction of e. We prove that if we remove the edges of $E'_d(C)$ from $P(C)$ then the remaining edges $P(C) - E'_d(C)$ form a simple cycle in G'.

▷ Claim 9. Edges in the set $P(C) - E'_d(C)$ form a simple cycle in G'.

Proof. Note that the lemma follows trivially if $P(C)$ is a simple cycle. Therefore, assume that $P(C)$ is not a simple cycle. We start traversing the walk $P(C)$ starting from the edges of the sequence $P(e_1)$. Let $P(e_k)$ be the first place where a vertex in the walk $P(e_1)P(e_2).....P(e_k)$ repeats, i.e., edges in the sequence $P(e_1)P(e_2).....P(e_{k-1})$ form a simple path, but after adding the edges of $P(e_k)$ some vertices are visited twice in the walk $P(e_1)P(e_2).....P(e_{k-1})P(e_k)$ for some $k \leq j$. This implies that some vertices are visited twice in the sequence $P(e_{k-1})P(e_k)$. Let $e_{k-1} = (u,v)$ and $e_k = (v,x)$. This implies that some copies of the vertex v appear twice in the sequence $P(u,v)P(v,x)$. Let B_1 and B_2 be the highest bags such that B_1 contains the copies of vertices u and v, and B_2 contains the copies of v and x. Let bag B be the lowest common ancestor of B_1 and B_2. We know that B must contain a copy of the vertex v, i.e., v_B. Let B' be the highest bag containing a copy of vertex v, i.e., $v_{B'}$. First, consider the case when neither B_1 is an ancestor of B_2 and vice-versa, other cases can be handled similarly. In that case sequence $P(u,v) = u_{B_1} v_{B_1} v_{parent(B_1)} \ldots v_B \ldots v_{B'}$ and $P(v,w) = v_{B'} \ldots v_B \ldots v_{parent(B_2)} v_{B_2} w_{B_2}$. Note that in $P(u,v)$ a path goes from v_B to $v_{B'}$ and the same path appear in reverse order from $v_{B'}$ to v_B in the sequence $P(v,x)$. Therefore if we remove these two paths from $P(u,v)$ and $P(v,w)$ the remaining subsequence of the sequence $P(u,v)P(v,w)$ will be a simple path, i.e., no vertex will appear twice since B is the lowest common ancestor of B_1 and B_2. Now repeat this procedure for $P_{k+1}, P_{k_2} \ldots$ and so on till P_k. In the end, we will obtain a simple cycle. ◁

Since we assumed that the weight function w' is skew-symmetric, we know that $w'(e) = -w(e^r)$, for all $e \in G'$. This implies that $w'(E'_d(C)) = 0$. Therefore $w(C) = w'(P(C)) = w'(P(C) - E'_d(C))$. From Claim 9 we know that edges in the set $P(C) - E'_d(C)$ form a simple cycle and we assumed that w' gives nonzero circulation to every simple cycle therefore, $w'(P(C) - E'_d(C)) \neq 0$. This implies that $w(C) \neq 0$. This finishes the proof of Lemma 8. ◀

Now the only thing remaining is the logspace construction of the polynomially bounded skew-symmetric weight function w'.

Constructing Weight Function for G'

To construct the weight function w' for G', we associate each edge of G' with some bag of T'. Let (u_{B_i}, v_{B_j}) be an edge in G': (i) if $i = j$, i.e., if B_i and B_j are the same bags. In this case associated (u_{B_i}, v_{B_j}) with that bag, (ii) if $i \neq j$: by our construction of G' we know that either B_i is the parent of B_j or B_j is the parent of B_i (i.e. u_{B_i} and v_{B_j} are the copies of a same vertex of G). In both the cases, associate (u_{B_i}, v_{B_j}) with the parent bag. We will use the following claim later in the paper.

▷ Claim 10. For any cycle, C in G', the bags of T' which have some edge of C associated with them, form a connected component in T'.

Proof. Note that if we treat each vertex in the bags of T' distinctly, then there is a one-to-one correspondence between vertices of G' and vertices in the bags of T'. Therefore, in T' a vertex of G' is identified by its corresponding vertex. Note that all the bags which contain vertices of the cycle C form a connected component in T'. We will now prove that if a bag B contains some vertices of C, then either B has some edges of C associate with it or no bag in the subtree rooted at B has any edge of C associated with it. From this, we can conclude that the bags which have some edges of C associated with them form a connected component in T'.

Assume that B is a bag which contains a vertex of C but no edge of C is associated with it. This implies that C never enters in any of the children of B. Because, let us assume it enters to some child B' of B through some vertex $v_{B'}$ of B'. In that case, there will be an edge $(v_B, v_{B'})$ of C associated with the bag B, which is a contradiction. Therefore subtree rooted at B will not have any edge of the cycle C associated with it. This finishes the proof. ◁

The weight function w' is similar to the one constructed for $K_{3,3}$-free and K_5 free graphs [5]. We assign weights to the edges of the graph G' depending upon the height of the bag they are associated with. The weights assigned to them are exponential in the height of the bags. Therefore, we need the height of a bag to be $O(\log n)$ to obtain a polynomially bounded weight function. Hence similar to [5], we define an auxiliary tree $A(T')$ of the tree T'. In some sense, $A(T')$ is a balanced representation of T', therefore the height of $A(T')$ is $O(\log n)$. Nodes in $A(T')$ are the same as T', i.e. the bags of T', but the edges between the bags are inserted differently. The weight of an edge of G' associated with a bag B of T' depends upon the height of the bag B in $A(T')$.

Auxiliary Tree. In order to construct the auxiliary tree from T', first, we find a node called *center* node $c(T')$. Make this node the root of the $A(T')$. Let $T_1, T_2, \ldots, T_l$ be the subtrees obtained by deleting $c(T')$ from T'. Recursively apply the same procedure on these subtrees and make $c(T_1), c(T_2), \ldots, c(T_l)$ children of $c(T)$. If $c(T)$ shares a separating set τ with T_i then $c(T_i)$ is said to be attached at τ with $c(T)$. Center nodes are chosen in such a way that the resultant tree $A(T')$ has height $O(\log n)$. Readers are referred to Section 3.3 of Arora et al. [5] to see the logspace construction of $A(T')$. We use the same construction here. The height of the root of $A(T')$ is defined as the number of nodes in the longest path from the root $c(T)$ to a leaf node. The heights of other nodes are one less than that of their parent. From now on, we work with $A(T')$. We use the following two properties of the auxiliary tree.

- **(i)** Height of a node in $A(T')$ is $O(\log n)$.
- **(ii)** If $\tilde{T}$ is a subtree of T', then there exists a bag B in $\tilde{T}$ such that all the other bags of $\tilde{T}$ are descendants of B in $A(T')$.

Now we define the weight function w' for the graph G'. Note that the graph induced by the set of edges associated with a bag B_i is either planar or constant size; we call these the components of G'. w' is a linear combination of two weight functions w_1 and w_2. w_1 gives nonzero circulation to those cycles which are completely contained within a component, and w_2 gives nonzero circulation to those cycles which span over at least two components. We define w_1 and w_2 separately for planar and constant size components. Let G'_{B_i} be the graph induced by the set of edges associated with the bag B_i. Let K be a constant such that $K > max(2^{m+2}, 7)$, where m is the maximum number of edges associated with any constant size component.

If G'_{B_i} is a planar component. w_1 for such components is same as the weight function defined in [6] for planar graphs. We know that given a planar graph G, its planar embedding can be computed in logspace [1].

▶ **Theorem 11** ([6]). *Given a planar embedding of a graph H, there exists a logspace computable function w such that for every cycle C of H, circulation of the cycle $w(C) \neq 0$.*

The above weight function gives nonzero circulation to every cycle that is completely contained in a planar component.

The weight function w_2 for planar components is defined as follows. w_2 assigns weights to only those faces of the component, which are adjacent to some separating set. For a subtree of T_s of $A(T')$, let $l(T_s)$ and $r(T_s)$ denote the number of leaf nodes in T_s and root node of T_s, respectively. For a bag B_i, $h(B_i)$ denotes the height of the bag in $A(T')$. If B_i is the only bag in the subtree rooted at B_i, then each face in G'_{B_i} is assigned weight zero. Otherwise, let τ be a separating set where some subtree T_i is attached to B_i. The faces adjacent to τ in G'_{B_i} are assigned weight $2 \times K^{h(r(T_i))} \times l(T_i)$. If a face is adjacent to more than one separating set, then the weight assigned to the face is the sum of the weights due to each separating set. The weight of a face is defined as the sum of the weights of the edges of the face in clockwise order. If we have a skew-symmetric weight function, then the weight of the clockwise cycle will be the sum of the weights of the faces inside the cycle [6]. Therefore assigning positive weights to every face inside a cycle will ensure that the circulation of the cycle is nonzero. Given weights on the faces of a graph, we can obtain weights for the edges so that the sum of the weights of the edges of a face remains the same as the weight of the face assigned earlier [23].

If G'_{B_i} is a constant size component. For this type of component, we need only one weight function. Thus we set w_2 to be zero for all the edges in G'_{B_i} and w_1 is defined as follows. Let $e_1, e_2, \ldots, e_k$ be the edges in the component Q_i, for some $k \leq m$. Edge e_j is assigned weight $2^i \times K^{h(r(T_i))-1} \times l(T_i)$ (for some arbitrarily fixed orientation), Where T_i is the subtree of $A(T')$ rooted at B_i. Note that for any subset of edges of G'_{B_i}, the sum of the weight of the edges in that subset is nonzero with respect to w_1.

The final weight function is $w' = \langle w_1 + w_2 \rangle$. Since the maximum height of a bag in $A(T')$ is $O(\log n)$, the weight of an edge is at most $O(n^c)$, for some constant $c > 0$.

▶ **Lemma 12.** *For a cycle C in G' sum of the weights of the edges of C associated with the bags in a subtree T_i of $A(T')$ is $< K^{h(r(T_i))} \times l(T_i)$.*

Proof. Let $w(C_{T_i})$ denotes the sum of the weight of the edges of a cycle C associated with the bags in T_i. We prove the Lemma by induction on the height of the root of the subtrees of $A(T')$. Note that the Lemma holds trivially for the base case when the height of the root of a subtree is 1.

Induction hypothesis: Assume that it holds for all the subtrees such that the height of their root is $< h(r(T_i))$.

Now we will prove it for T_i. Let $T_i^1, T_i^2, \ldots, T_i^k$ be the subtrees attached to $r(T_i)$.

- First, consider the case when $G'_{r(T_i)}$ is a constant size graph: In this case, we know that the sum of the weights of the edges of C associated with $r(T_i)$ is $\leq \sum_{j=1}^{m} 2^j \times K^{h(r(T_i))-1} \times l(T_i)$ and by the induction hypothesis, we know that $w(C_{T_i^j}) < K^{h(r(T_i^j))} \times l(T_i^j)$, for all $j \in [k]$.

Therefore,

$$\begin{aligned}
w(C_{T_i}) &\leq \sum_{j=1}^{m} 2^j \times K^{h(r(T_i))-1} \times l(T_i) + \sum_{j=1}^{k} K^{h(r(T_i^j))} \times l(T_i^j) \\
w(C_{T_i}) &\leq (2^{m+1} - 1) \times K^{h(r(T_i))-1} \times l(T_i) + K^{h(r(T_i))-1} \times l(T_i) \\
w(C_{T_i}) &\leq (2^{m+1}) \times (K^{h(r(T_i))-1} \times l(T_i)) && [K > 2^{m+2}] \\
w(C_{T_i}) &< K^{h(r(T_i))} \times l(T_i)
\end{aligned}$$

- When $G'_{r(T_i)}$ is a planar graph: let $\tau_1, \tau_2, \ldots, \tau_k$ be the separating sets present in $G'_{r(T_i)}$ such that the subtree T_i^j is attached to $r(T_i)$ at τ_j, for all $j \in [k]$. A separating set can be present in at most 3 faces. Thus it can contribute $2 \times 3 \times K^{h(r(T_i^j))} \times l(T_i^j)$ to the circulation of the cycle C. Therefore,

$$\begin{aligned}
w(C_{T_i}) &\leq \sum_{j=1}^{k} 6 \times K^{h(r(T_i^j))} \times l(T_i^j) + \sum_{j=1}^{k} K^{h(r(T_i^j))} \times l(T_i^j) \\
w(C_{T_i}) &\leq 7 \times K^{h(r(T_i))-1} \times l(T_i) && [K > 7] \\
w(C_{T_i}) &< K^{h(r(T_i))} \times l(T_i)
\end{aligned}$$

◀

▶ **Lemma 13.** *For a cycle, C in G let B_i be the unique highest bag in $A(T')$ that have some edges of C associated with it. Then the sum of the weights of the edges of C associated with B_i will be more than that of the rest of the edges of C associated with the other bags.*

Proof. Let T_i be the subtree of $A(T')$ rooted at B_i. We know that sum of the weights of the edges of C associated B_i is $\geq 2 \times K^{h(r(T_i))-1} \times l(T_i)$. Let $T_i^1, T_i^2, \ldots, T_i^k$ be the subtree of T_i rooted at children of B_i. By Lemma 12, we know that the sum of the weight of the edges of C associated with the bags in these subtrees is $< \sum_{j=1}^{k} K^{h(r(T_i^j))} \times l(T_i^j) = K^{h(r(T_i))-1} \times l(T_i)$. Therefore, the lemma follows. ◀

▶ **Lemma 14.** *Circulation of a simple cycle C in the graph G' is nonzero with respect to w'.*

Proof. If C is contained within a component, i.e., its edges are associated with a single bag B_i, then we know that w_1 assigns nonzero circulation to C. Suppose the edges of C are associated with more than one bag in T'. By Claim 10, we know that these bags form a connected component. By the (ii) property of $A(T')$, we know that there is a unique highest bag B_i in $A(T')$ which have edges of C associated with it. Therefore from Lemma 13 we know that the circulation of C will be nonzero. ◀

Proof of Theorem 1. Proof of Theorem 1 follows from Lemma 8 and 14. ◀

4 Maximum Matching

In this section, we consider the complexity of the maximum matching problem in single crossing minor free graphs. Recently Datta et al. [8] have shown that the bipartite maximum matching can be solved in SPL in the planar, bounded genus, $K_{3,3}$-free and K_5-free graphs.

Their techniques can be extended to any graph class where nonzero circulation weights can be assigned in logspace. For constructing a maximum matching in $K_{3,3}$-free and K_5-free bipartite graphs, they use the logspace algorithm of [5] as a black box. Since from Theorem 1 nonzero circulation weights can be computed for the more general class of any single crossing minor free graphs, we get the bipartite maximum matching result of Corollary 2.

In a related work recently, Eppstein and Vazirani [16] have shown an NC algorithm for the case when the graph is not necessarily bipartite. However, the result holds only for constructing perfect matchings. In non-bipartite graphs, there is no known parallel (e.g., NC) or space-efficient algorithm for deterministically constructing a maximum matching even in the case of planar graphs [30, 8]. Datta et al. [8] givelo an approach to design a *pseudo-deterministic* NC algorithm for this problem. Pseudo-deterministic algorithms are probabilistic algorithms for search problems that produce a unique output for each given input with high probability. That is, they return the same output for all but a few of the possible random choices. We call an algorithm pseudo-deterministic NC if it runs in RNC and is pseudo-deterministic.

Using the Gallai-Edmonds decomposition theorem, [8] shows that the search version of the maximum matching problem reduces to determining the size of the maximum matching in the presence of algorithms to (a) find a perfect matching and to (b) solve the bipartite version of the maximum matching, all in the same class of graphs. This reduction implies a pseudo-deterministic NC algorithm as we only need to use randomization for determining the size of the matching, which always returns the same result. For single crossing minor free graphs, using the NC algorithm of [16] for finding a perfect matching and our SPL algorithm for finding a maximum matching in bipartite graphs, we have the following result:

▶ **Theorem 15.** *Maximum matching in single-crossing minor free graphs (not necessarily bipartite) is in pseudo-deterministic* NC*.*

5 Conclusion

We have given a construction of a nonzero circulation weight function for the class of graphs that can be expressed as 3-clique-sums of planar and constant treewidth graphs. However, it seems that our technique can be extended to the class of graphs that can be expressed as 3-clique-sums of constant genus and constant treewidth graphs. Further extending our results to larger graph classes would require fundamentally new techniques. This is so because the most significant bottleneck in parallelizing matching algorithms for larger graph classes such as apex minor free graphs or H-minor free graphs for a finite H is the absence of a parallel algorithm for the structural decomposition of such families. Thus we would need to revisit the Robertson-Seymour graph minor theory to parallelize it. This paper thus serves the dual purpose of delineating the boundaries of the known regions of parallel (bipartite) matching and reachability and as an invitation to the vast unknown of parallelizing the Robertson-Seymour structure theorems.

References

1 Eric Allender and Meena Mahajan. The complexity of planarity testing. *Information and Computation*, 189:117–134, 2004.

2 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999.

3 Nima Anari and Vijay V. Vazirani. Matching is as easy as the decision problem, in the NC model. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 54:1–54:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

4 Nima Anari and Vijay V. Vazirani. Planar graph perfect matching is in nc. *J. ACM*, 67(4), May 2020. doi:10.1145/3397504.

5 Rahul Arora, Ashu Gupta, Rohit Gurjar, and Raghunath Tewari. Derandomizing isolation lemma for $k_{3,3}$-free and k_5-free bipartite graphs. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, France*, pages 10:1–10:15, 2016.

6 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Transactions on Computation Theory*, 1(1):1–17, 2009. doi:10.1145/1490270.1490274.

7 Erin W. Chambers and David Eppstein. Flows in one-crossing-minor-free graphs. *J. Graph Algorithms Appl.*, 17(3):201–220, 2013.

8 Samir Datta, Raghav Kulkarni, Ashish Kumar, and Anish Mukherjee. Planar maximum matching: Towards a parallel algorithm. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPIcs*, pages 21:1–21:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.21.

9 Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory Comput. Syst.*, 47(3):737–757, 2010.

10 Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N.V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. *Journal of Computer and System Sciences*, 78(3):765–779, 2012. In Commemoration of Amir Pnueli. doi:10.1016/j.jcss.2011.11.002.

11 Samir Datta, Pankaj Kumar, Anish Mukherjee, Anuj Tawari, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of reachability: How many changes can we handle? In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 122:1–122:19, 2020.

12 Erik D. Demaine, Mohammad Taghi Hajiaghayi, Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. *J. Comput. Syst. Sci.*, 69(2):166–195, 2004.

13 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

14 Jack Edmonds. Paths, trees and flowers. *Canadian Journal Of Mathematics*, pages 449–467, 1965.

15 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of bodlaender and courcelle. In *FOCS '10: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, 2010. URL: http://www.eccc.uni-trier.de/report/2010/062/.

16 David Eppstein and Vijay V. Vazirani. NC algorithms for computing a perfect matching and a maximum flow in one-crossing-minor-free graphs. *SIAM J. Comput.*, 50(3):1014–1033, 2021.

17 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in Quasi-NC. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 754–763, 2016.

18 L. R. Ford, Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.

19 Chetan Gupta, Vimal Raj Sharma, and Raghunath Tewari. Reachability in O(log n) Genus Graphs is in Unambiguous Logspace. In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 34:1–34:13, 2019.

20 Chetan Gupta, Vimal Raj Sharma, and Raghunath Tewari. Efficient Isolation of Perfect Matching in O(log n) Genus Bipartite Graphs. In Javier Esparza and Daniel Krá?, editors, *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:13, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

21 Vivek Anand T. Kallampally and Raghunath Tewari. Trading determinism for time in space bounded computations. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 10:1–10:13, 2016.

22 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 22–32. ACM, 1985.

23 Arpita Korwar. Matching in planar graphs. Master's thesis, IITK, 2009.

24 Jan Kynčl and Tomáš Vyskočil. Logspace reduction of directed reachability for bounded genus graphs to the planar case. *ACM Transactions on Computation Theory*, 1(3):1–11, 2010. doi:10.1145/1714450.1714451.

25 Meena Mahajan and Kasturi R. Varadarajan. A new nc-algorithm for finding a perfect matching in bipartite planar and small genus graphs (extended abstract). In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 351–357, New York, NY, USA, 2000. ACM. doi:10.1145/335305.335346.

26 Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal on Computing*, 24:1002–1017, 1995.

27 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 345–354, 1987.

28 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.

29 Neil Robertson and Paul D. Seymour. Excluding a graph with one crossing. In *Graph Structure Theory, Proceedings of a AMS-IMS-SIAM Joint Summer Research Conference on Graph Minors, June 22 to July 5, 1991, Seattle, USA*, pages 669–675, 1991.

30 Piotr Sankowski. NC algorithms for weighted planar perfect matching and related problems. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 97:1–97:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

31 Simon Straub, Thomas Thierauf, and Fabian Wagner. Counting the number of perfect matchings in K_5-free graphs. *Theory Comput. Syst.*, 59(3):416–439, 2016.

32 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in Quasi-NC. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA*, pages 696–707. IEEE Computer Society, 2017.

33 Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Inf. Comput.*, 215:1–7, 2012.

34 Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free Graphs and K_5-free Graphs is in Unambiguous Log-Space. In *17th International Conference on Foundations of Computation Theory (FCT)*, Lecture Notes in Computer Science 5699, pages 323–334. Springer-Verlag, 2009.

35 Thomas Thierauf and Fabian Wagner. The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. *Theory Comput. Syst.*, 47(3):655–673, 2010.

Approximating the Number of Prime Factors Given an Oracle to Euler's Totient Function

Yang Du
Departments of EECS, CSE Division, University of Michigan, Ann Arbor, MI, USA

Ilya Volkovich
Computer Science Department, Boston College, Chestnut Hill, MA, USA

Abstract

In this work we devise the first efficient *deterministic* algorithm for approximating $\omega(N)$ – the number of prime factors of an integer $N \in \mathbb{N}$, given in addition oracle access to Euler's Totient function $\Phi(\cdot)$. We also show that the algorithm can be extended to handle a more general class of additive functions that "depend solely on the exponents in the prime factorization of an integer"[1]. In particular, our result gives the first algorithm that approximates $\omega(N)$ without necessarily factoring N. Indeed, all the previously known algorithms for computing or even approximating $\omega(N)$ entail factorization of N, and therefore are either randomized [12, 9] or require the Generalized Riemann Hypothesis (GRH) [10].

Our approach combines an application of Coppersmith's method for finding non-trivial factors of integers whose prime factors satisfy certain "relative size" conditions of [11], together with a new upper bound on $\Phi(N)$ in terms of $\omega(N)$ which could be of independent interest.

2012 ACM Subject Classification Mathematics of computing → Number-theoretic computations; Theory of computation → Problems, reductions and completeness; Theory of computation → Computational complexity and cryptography

Keywords and phrases Euler's Totient Function, Integer Factorization, Number of Prime Factors, Derandomization

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.17

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

The Fundamental Theorem of Arithmetic states that each $N \in \mathbb{N}$ admits a unique factorization into a product of powers of prime numbers $N = p_1^{\alpha_1} \cdot \ldots \cdot p_k^{\alpha_k}$. The *integer factorization problem* asks to compute those prime factors given N as an input. In addition to being a central problem in algorithmic number theory, integer factorization has a direct application to cryptography as all known approaches to break the RSA cryptosystem involve integer factorization. Indeed, there is no known efficient algorithm for the problem and it is believed to be computationally hard. In light of this hardness, a large body of work [10, 4, 14, 8, 11, 6, 7] has been dedicated to the study of the computational complexity of integer factorization when in addition the algorithm is given oracle access to some function $f : \mathbb{N} \to \mathbb{N}$ that provides "useful information" about N. On the practical side, oracles can model extra information on N obtained by means of side-channel attacks.

One such line of work considers integer factorization algorithms that are given oracle access to Euler's Totient function $\Phi(\cdot)$ (see Definition 12 for a formal definition). In a seminal work [10], Miller has shown that one can efficiently factor integers with an oracle to Φ. More

[1] This class and terminology were introduced and studied by Shallit & Shamir in [13].

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 17; pp. 17:1–17:10
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

precisely, the result is an efficient *deterministic* factorization algorithm, however its correctness relies on the Generalized Riemann Hypothesis (GRH). Subsequently, Long [9] and Rabin [12] replaced the GRH assumption with randomness, thus obtaining (unconditional) efficient *randomized* factorization algorithms. It has since been an open question to derandomize these algorithms unconditionally. In this paper we make another step towards the resolution of this problem.

1.1 Previous Results

Landau [8] and Woll [14] have shown that one can efficiently compute the square-free part of an integer[2] given an oracle access to $\Phi(\cdot)$. Building on this result, Żralek [15] and later on Hittmeir & Pomykała [6] achieved another milestone by exhibiting a reduction of integer factorization to the computation of Φ in deterministic subexponential-time. Morain et al. [11] applied Coppersmith's method (see e.g. [5]) to find non-trivial factors of integers whose prime factors satisfy certain "relative size" conditions.

1.2 Our Results

We take a slightly different approach: can we learn some "useful information" about N, given oracle access to $\Phi(\cdot)$? Formally, given an integer $N \in \mathbb{N}$ as an input, we would like to compute $f(N)$ for some "interesting" function $f : \mathbb{N} \to \mathbb{N}$ that can be efficiently computed given the complete factorization of N. A particular example of such a function is $\omega(N)$, which is defined as the number of prime factors of N. Our main result is an approximation algorithm for $\omega(N)$.

▶ **Theorem 1.** *There exist an efficient deterministic algorithm that given $N \in \mathbb{N}$ as an input, outputs an integer L satisfying: $\omega(N) \leq L \leq 3^{\omega(N)}$, given in addition oracle access to $\Phi(\cdot)$.*

▶ Remark 2. Although, $\omega(N) \leq \log N$, computing the actual value of $\omega(N)$ is *believed* to be as hard as factoring N (see e.g. [3, 1]).

▶ Remark 3. Our result is non-trivial when $\omega(N) = O(1)$ or more generally, when $\omega(N) = o(\log \log N)$.

To put our result in context, consider a decision version of the problem: given $N \in \mathbb{N}$ and oracle access to $\Phi(\cdot)$ (as before), decide if $\omega(N) = k$ for a **fixed** $k \in \mathbb{N}$. For $k = 1$ the problem corresponds to primality testing, which can trivially be solved given $\Phi(N)$ since $\Phi(N) = N - 1$ if and only if N is prime[3]. The case $k = 2$ is handled by a falklore result (See Lemma 13 for more details). For $k \geq 3$ the problem is still open.
It is also important to point out that one of the results of [6] provides an efficient deterministic factorization algorithm for N with $\omega(N) = O(1)$, if **in addition** the algorithm given the prime factorization of $\Phi(N)$. To the best of our knowledge, there is no known efficient deterministic algorithm to compute the prime factorization of $\Phi(N)$, even given oracle access to $\Phi(\cdot)$.

In [13], Shallit & Shamir introduced and studied the class of functions that "depend solely on the exponents in the prime factorization of an integer" (exponent-dependent functions, for short). They also showed that $d(N)$ - the number of positive divisors of N, is complete for this class under deterministic Turing reductions. More formally,

[2] See definition 10 for a formal definition.
[3] The breakthrough result of [2] gives an efficient deterministic primality test without $\Phi(N)$.

▶ **Lemma 4** ([13]). *Let $f : \mathbb{N} \to \mathbb{N}$ be an exponent-dependent function that can be efficiently (and deterministically) computed given the factorization of N. Then there exists an efficient deterministic algorithm that given $N \in \mathbb{N}$ as an input, outputs $f(N)$, given in addition oracle access to $d(\cdot)$.*

We observe that $\Phi(N)$ together with $\omega(N)$ are hard for the class of exponent-dependent functions.

▶ **Observation 5.** *For $f : \mathbb{N} \to \mathbb{N}$ as above, there exists a deterministic algorithm that given $N \in \mathbb{N}$ as an input, outputs $f(N)$, given in addition oracle access to $\Phi(\cdot)$ and $\omega(\cdot)$.*

As an important corollary, we obtain that if one can compute $\omega(N)$ **exactly**, given oracle access to $\Phi(\cdot)$ then $\Phi(\cdot)$ is (by itself) hard for the class of exponent-dependent functions. Such a result would constitute another important milestone on route to derandomization of the results of [10, 12, 9]. We show that a similar statement holds w.r.t approximations for *additive* functions: i.e. $f(N \cdot M) = f(N) + f(M)$ for coprime M and N (see Definition 14 for more details).

▶ **Theorem 6.** *Let $f : \mathbb{N} \to \mathbb{N}$ be an additive exponent-dependent function that can be efficiently (and deterministically) computed given the factorization of N. Then there exist an efficient deterministic algorithm that given $N \in \mathbb{N}$ as an input, outputs an integer L satisfying: $f(N) \leq L \leq 3^{f(N)}$, given in addition oracle access to $\Phi(\cdot)$.*

As an instantiation, we obtain an approximation algorithm for $\Omega(N)$ - the number of prime factors of N with multiplicity (see Example 15 for more details).

▶ **Corollary 7.** *There exist an efficient deterministic algorithm that given $N \in \mathbb{N}$ as an input, outputs an integer L satisfying: $\Omega(N) \leq L \leq 3^{\Omega(N)}$, given in addition oracle access to $\Phi(\cdot)$.*

Finally, we note that our main result is obtained using the following new upper bound on $\Phi(N)$ in terms of $\omega(N)$ which could be of independent interest.

▶ **Theorem 8.** *Let $N \in \mathbb{N}$ and let $k = \omega(N)$. Then $\Phi(N) \leq \left(\sqrt[k]{N} - 1\right)^k$.*

1.3 Techniques

In this section we give the outline of the proof of our main result.

First, we give a new upper bound on $\Phi(N)$ in terms of $k = \omega(N)$. Next, we show that there exists an efficient procedure that finds a non-trivial factor of N, if N has a "small" factor. Alternatively, if the procedure fails, then N cannot have a small factor. We complement this result by showing that if N still satisfies the upper bound for a "much" larger value of k then N must have a small factor.

Based on the above, our main algorithm operates as follows: given $N \in \mathbb{N}$ it first attempts to find a non-trivial factor D of N. If it succeeds, then the algorithm proceeds recursively on N/D and D. Otherwise (i.e. if the procedure fails), as discussed above, N cannot have a small factor. This, in turn, implies that the largest value of k satisfying the upper bound cannot be "much larger" than $\omega(N)$ as otherwise N must have had a small factor.

1.4 Organization

The paper is organized as follows: we start by some basic definitions and notations in Section 2. In that section we also prove Observation 5 and Theorem 6 as these follow immediately from the definitions. In Section 3 we give our main algorithm and a new upper bound on $\Phi(N)$ in terms of $\omega(N)$, thus proving Theorems 1 and 8. We show some numerical examples in Section 4. Finally, we conclude with discussion and open questions in Section 5.

2 Preliminaries

Let $\mathcal{P}$ denote the set of all primes. The Fundamental Theorem of Arithmetic states that each integer $N \in \mathbb{N}$ has a unique prime factorization. That is, N can be uniquely written as $N = \prod_{i=1}^{k} p_i^{\alpha_i}$ such that for all i: $p_i \in \mathcal{P}$ and $\alpha_i \in \mathbb{N}$.

▶ **Definition 9** (Roughness). *Let $r \in \mathbb{N}$. A number $N \in \mathbb{N}$ is called* r-rough *if all its prime factors are greater then or equal to r. That is, $\forall i : p_i \geq r$.*

▶ **Definition 10** (Radicals). *We define the* square-free part *or* radical *of N as* $\operatorname{rad}(N) \triangleq \prod_{i=1}^{k} p_i$. *$N$ is called* square-free *or* radical *iff $\forall i : \alpha_i = 1$. In other words, N is radical iff $N = \operatorname{rad}(N)$. We define the set $\mathbb{SQF} \subseteq \mathbb{N}$ as the set of all square-free integers.*

▶ **Definition 11** (Square-free decomposition). *A* square-free decomposition *of $N \in \mathbb{N}$ is a factorization of N as $N = N_1^1 N_2^2 N_3^3 \ldots N_\ell^\ell$ such that* $\gcd(N_i, N_j) = 1$ *and $N_i \in \mathbb{SQF}$.*

By The Fundamental Theorem of Arithmetic, each integer admits a *unique* square-free decomposition. Furthermore, observe that $\ell \leq \log N$. In addition, $\operatorname{rad}(N) = N_1 N_2 N_3 \cdots N_\ell$.

▶ **Definition 12** (Euler's Totient Function). *$\Phi(N) : \mathbb{N} \to \mathbb{N}$ is defined as $\Phi(N) \triangleq \prod_{i=1}^{k} p_i^{\alpha_i - 1} \cdot (p_i - 1)$.*

▶ **Lemma 13.** *Below are some useful properties and facts.*

1. *$\Phi(N) \leq N - 1$. Equality holds if and only if N is prime.*
2. $\dfrac{\Phi(N)}{N} = \dfrac{\Phi(\operatorname{rad}(N))}{\operatorname{rad}(N)}$.
3. *Folklore: Let N be a product of two distinct primes $N = pq$. There exists an algorithm that given N and $\Phi(N)$, outputs p and q, in time* $\operatorname{polylog}(N)$.
   A sketch of the proof appears in Section A.
4. *[14, 8]: There exists an algorithm that given N and oracle access to $\Phi(\cdot)$, outputs the square-free decomposition of N, in time* $\operatorname{polylog}(N)$.

▶ **Definition 14** ([13]). *We call a function $f : \mathbb{N} \to \mathbb{C}$* exponent-dependent *if it depends solely on the exponents in the prime factorization N.*
In addition, we say that f is additive, *if $f(N \cdot M) = f(N) + f(M)$ for all coprime $N, M \in \mathbb{N}$.*

▶ **Example 15.** Prominent examples include the following functions.

1. $\Omega(N) \triangleq \sum_{i=1}^{k} \alpha_i$ - the number of prime factors of N with multiplicity.
2. $\omega(N) \triangleq k$ - the number of prime factors of N **without** multiplicity.
3. $\mu : \mathbb{N} \to \{-1, 0, 1\}$ - Möbius Function:

$$\mu(N) = \begin{cases} 0 & \text{if } N \text{ is not square-free} \\ (-1)^k & \text{if } N \text{ is a product of } k \text{ (distinct) primes} \end{cases}$$

4. $d(N) \triangleq \prod_{i=1}^{k} (\alpha_i + 1)$ - the number of positive divisors of N.

Here, $\Omega(N)$ and $\omega(N)$ are additive.

We remark that there is no known polynomial-time algorithm for computing any of the above functions. Indeed, they are *believed* to be as hard as (complete) integer factorization [3, 1]. Several relations between these (and other) functions have been established in [10, 13, 4, 14, 8, 11]. In particular, in [13] it was shown that $d(N)$ is complete for the class of exponent-dependent functions under Turing reductions, by showing that an oracle to $d(\cdot)$ can be used to compute $e(N) \triangleq \{\alpha_1, \ldots, \alpha_k\}$ - the **multiset** of exponents in the prime factorization of N. We observe that $e(N)$ can be easily constructed from a square-free decomposition of an integer N, given oracle access to $\omega(\cdot)$.

▶ **Observation 16.** *Let $N = N_1^1 N_2^2 N_3^3 \cdots N_\ell^\ell$ be the square-free decomposition of N. Then $e(N) = \{(i, \omega(N_i)) \mid \omega(N_i) > 0\}$. That is, $e(N)$ contains all the values $w(N_i)$ greater than 0, where each $w(N_i)$ appears i times.*

Observation 5 follows by combining the above observation with Part 4 of Lemma 13. The following is another observation immediate from the definition.

▶ **Observation 17.** *Let $N = N_1^1 N_2^2 N_3^3 \cdots N_\ell^\ell$ be the square-free decomposition of N and let $f : \mathbb{N} \to \mathbb{N}$ be an exponent-dependent additive function. Then $f(N) = \sum\limits_{i=1}^{\ell} \omega(N_i) \cdot f(2^i)$.*

Theorem 6 follows by combining the above with Part 4 of Lemma 13 and Theorem 24.

▶ **Lemma 18.** *We have three inequalities that will be used in the later sections.*

1. *AM-GM Inequality. Let $x_1, \ldots, x_m \geq 0$: Then the arithmetic mean is greater then or equal to the geometric mean of these numbers:*

$$\frac{x_1 + \cdots + x_m}{m} \geq \sqrt[m]{x_1 \cdots x_m}.$$

2. *Multivariate Bernoulli Inequality. Let $0 \leq \varepsilon_1, \ldots, \varepsilon_m \leq 1$, then it follows that*

$$\prod_{i=1}^{m} (1 - \varepsilon_i) \geq 1 - \sum_{i=1}^{m} \varepsilon_i$$

3. *For any positive integer k, and $0 \leq x \leq \dfrac{1}{2k}$ and $\ell \geq 2k$, we have*

$$(1 - x)^\ell \leq 1 - kx.$$

Proof. We will show the proof of Part 3 of Lemma 18. Since $1 - x$ is less then 1 and $\ell \geq 2k$, then $(1-x)^\ell \leq (1-x)^{2k}$, we now only need to show that $(1-x)^{2k} \leq 1 - kx$. We know that $(1-x)^{2k}$ is a convex function on $[0, 1]$ because the second derivative $2k(2k-1)(1-x)^{2k-2} > 0$. In addition, it is clear that both side of the inequality is evaluated to be 1 at $x = 0$, and the first derivative of $(1-x)^{2k}$ is less then the first derivative of $1 - kx$ at $x = 0$. Therefore, we only need to show $(1-x)^{2k} \leq 1 - kx$ is established at $x = \dfrac{1}{2k}$, which is

$$\left(1 - \frac{1}{2k}\right)^{2k} \leq e^{-1} \leq \frac{1}{2} = 1 - k \cdot \frac{1}{2k}.$$

We can see that when k goes from 1 to infinity, the left hand side approached to $1/e$ from the bottom, so it is always less then $1/2$. ◀

3 Algorithm and Technical Results

In this section we prove our main result (Theorem 1) and give a new upper bound on $\Phi(N)$ in terms of $\omega(N)$, thus proving Theorem 8. We first give the upper bound for the case of square-free integers.

▶ **Lemma 19.** *Let $N \in \mathbb{SQF}$ and let $k = \omega(N)$. Then* $\frac{\Phi(N)}{N} \leq \left(1 - \frac{1}{\sqrt[k]{N}}\right)^k$

Proof. Let $N = p_1 \cdots p_k$. Then by applying AM-GM inequality (Lemma 18) we get:

$$\left(\frac{\Phi(N)}{N}\right)^{1/k} = \sqrt[k]{\prod_{i=1}^{k}\left(1-\frac{1}{p_i}\right)} \leq \frac{\sum_{i=1}^{k}\left(1-\frac{1}{p_i}\right)}{k} = 1 - \frac{1}{k}\sum_{i=1}^{k}\frac{1}{p_i} \leq 1 - \sqrt[k]{\prod_{i=1}^{k}\frac{1}{p_i}} = 1 - \frac{1}{\sqrt[k]{N}}.$$

◀

Next, we extend the upper bound to arbitrary N. Theorem 8 follows from the next corollary.

▶ **Corollary 20.** *Let $N \in \mathbb{N}$ and let $k = \omega(N)$. Then* $\frac{\Phi(N)}{N} \leq \left(1 - \frac{1}{\sqrt[k]{\mathrm{rad}(N)}}\right)^k \leq \left(1 - \frac{1}{\sqrt[k]{N}}\right)^k$

Proof. Apply Lemma 19 on $\mathrm{rad}(N)$ together with Lemma 13, Part 2. ◀

Next, we show that there exists an efficient procedure that finds a non-trivial factor of N, if N has a small factor. Alternatively, if the procedure fails, then N cannot have a small factor. Our result relies on the following result of [11] that uses Coppersmith's method to find non-trivial factors of integers whose prime factors satisfy certain relative size conditions.

▶ **Lemma 21** ([11]). *Suppose $\omega(N) \geq 3$ and there exists $1 \leq r < \omega(N)$ such that*

$$\alpha_r \geq 2 \sum_{i=r+1}^{\omega(N)} \alpha_i,$$

where $\alpha_i \triangleq \log_N(p_i)$. Then there exists an algorithm that given N and $\Phi(N)$ recovers the factor $D = \prod_{i=1}^{r} p_i$, in time $\mathrm{polylog}(N)$.

▶ **Theorem 22.** *Let $N \in \mathbb{SQF}$ and suppose N has a factor smaller then $N^{\frac{1}{3^{\omega(N)-1}}}$. Then there exists an algorithm that given N and $\Phi(N)$ as input, finds a non-trivial factor of N, in time* $\mathrm{polylog}(N)$.

Proof. Let $N = p_1 \cdots p_k$ with $p_1 > p_2 > \ldots > p_k$, and $\omega(N)$ is hence equal to k. Observe that the premises of the claim are equivalent to $p_k \leq N^{\frac{1}{3^{k-1}}}$, namely $\alpha_k \leq \frac{1}{3^{k-1}}$. It is sufficient to show that N satisfies that premises of Lemma 21. We suppose for contradiction that for all $1 \leq r \leq k-1$,

$$\alpha_r < 2 \sum_{i=r+1}^{k} \alpha_i. \tag{1}$$

We claim that it implies that for all $1 \leq i \leq k-2$, we have $\alpha_{k-i} < 2 \cdot 3^{i-1} \cdot \alpha_k$. We prove this by induction. The base case $\alpha_{k-1} < 2\alpha_k$ follows directly from Equation 1 when we set $r = k-1$. Now, assume that for all $1 \leq i \leq m$, $\alpha_{k-i} < 2 \cdot 3^{i-1} \cdot \alpha_k$ is established, then for $i = m+1$ we have:

$$\alpha_{k-m-1} < 2 \sum_{i=k-m}^{k} \alpha_i < 2 \left[\sum_{i=1}^{m} \left(2 \cdot 3^{i-1} \cdot \alpha_k\right) + \alpha_k \right] = 2 \left[2 \cdot \frac{3^m - 1}{3-1} + 1 \right] \alpha_k = 2 \cdot 3^m \cdot \alpha_k.$$

Therefore,

$$1 = \alpha_1 + \cdots + \alpha_k < 2\left(3^{k-2} + 3^{k-1} + \ldots + 1\right)\alpha_k + \alpha_k = \left(2 \cdot \frac{3^{k-1}-1}{3-1} + 1\right)\alpha_k = 3^{k-1} \cdot \alpha_k \leq 1$$

leading to a contradiction. ◀

We complement the above result by showing that (under certain technical conditions) if N satisfies the bound of Lemma 19 with a value of k "much" larger then $\omega(N)$ then N must have a small factor.

▶ **Lemma 23.** *Let $N \in \mathbb{SQF}$ and suppose that N is $(2\omega(N))$-rough. Furthermore, suppose that $\Phi(N)^{1/\ell} \leq N^{1/\ell} - 1$ for some $\ell \geq 2\omega(N)$. Then N has a factor smaller then $N^{1/\ell}$.*

Proof. As before, let $N = p_1 \cdots p_k$ with $p_1 > \cdots > p_k$ so $\omega(N) = k$. By the premises, $p_k \geq 2k$. Next, since $\Phi(N)^{1/\ell} \leq N^{1/\ell} - 1$ we have

$$\frac{\Phi(N)}{N} \leq \left(1 - \frac{1}{N^{1/\ell}}\right)^{\ell}.$$

By the Multivariate Bernoulli Inequality in Lemma 18 part 2, we have

$$\frac{\Phi(N)}{N} \geq 1 - \sum_{i=i}^{k} \frac{1}{p_i} \geq 1 - \frac{k}{p_k}.$$

Therefore, by combining the above we obtain:

$$\left(1 - \frac{k}{p_k}\right)^{1/\ell} \leq 1 - \frac{1}{N^{1/\ell}}.$$

By Lemma 18 part 3, since $\frac{1}{p_k} \leq \frac{1}{2k}$ and $\ell \geq 2k$ we have that:

$$1 - \frac{1}{p_k} \leq \left(1 - \frac{k}{p_k}\right)^{1/\ell}.$$

Therefore,

$$1 - \frac{1}{p_k} \leq 1 - \frac{1}{N^{1/\ell}} \qquad \Longrightarrow \qquad p_k \leq N^{1/\ell}.$$ ◀

Given $N \in \mathbb{N}$, our main algorithm operates as follows: first, it invokes the procedure in Theorem 22 attempting to find a non-trivial factor D of N. If it succeeds, then the algorithm proceeds recursively on N/D and D. Otherwise (i.e. if the procedure fails), as discussed above, N cannot have a small factor. This, in turn, implies that the largest value of k satisfying the bound in Lemma 19 has to be at most $3^{\omega(N)}$ as otherwise, by Lemma 23 N must have had a small factor.

▶ **Theorem 24.** *There exist an algorithm that given $N \in \mathbb{SQF}$ as an input, outputs an integer L satisfying: $\omega(N) \leq L \leq 3^{\omega(N)}$, given in addition oracle access to $\Phi(\cdot)$, in time* polylog(N)*. The description of the algorithm is given below in Algorithm 1.*

Algorithm 1 Approximation of number of factors (ANF).

```
Function: ANF($N$)
Input: $N \in \mathbb{SQF}$ and oracle access to $\Phi(\cdot)$.
Output: An integer $L$ satisfying: $\omega(N) \leq L \leq 3^{\omega(N)}$.
// $N$ is prime
if $\Phi(N) = N - 1$ then return 1;
if $N$ has 2 prime factors then return 2; / Invoking the algorithm from Lemma 13
  Part 3.
for $M = 1$ to $2\log N$ do
    if $M \mid N$ then return ANF($M$) + ANF($N/M$);
Invoke the algorithm from Theorem 22 on $N$ and $\Phi(N)$. Let $D$ be the output;
if $D \mid N$ then return ANF($D$) + ANF($N/D$);
// Otherwise:
$L = \max_{\ell} \Phi(N) \leq \left(N^{1/\ell} - 1\right)^{\ell}$;
return $L$;
```

Proof. The proof is by induction on $\omega(N)$. The base cases, i.e. $\omega(N) = 1, 2$, follow from Lemma 13, Parts 1 and 3. Now suppose $\omega(N) \geq 3$. We consider a few cases.

1. N has a factor $M \leq 2\log N$.
   Observe that $M, N/M \in \mathbb{SQF}$. In addition, as $\omega(N) = \omega(M) + \omega(N/M)$ we have that $\omega(M), \omega(N/M) < \omega(N)$. Therefore, by the induction hypothesis:
   $$\omega(M) \leq \mathrm{ANF}(M) \leq 3^{\omega(M)} \text{ and } \omega(N/M) \leq \mathrm{ANF}(N/M) \leq 3^{\omega(N/M)}.$$
   Consequently:
   $$\mathrm{ANF}(N) = \mathrm{ANF}(M) + \mathrm{ANF}(N/M) \leq 3^{\omega(M)} + 3^{\omega(N/M)} \leq 3^{\omega(M)} \cdot 3^{\omega(N/M)} = 3^{\omega(N)}.$$
   and
   $$\mathrm{ANF}(N) = \mathrm{ANF}(M) + \mathrm{ANF}(N/M) \geq \omega(M) + \omega(N/M) = \omega(N).$$
2. The algorithm from Theorem 22 produces a factor D such that $D \mid N$.
   The claim follows by repeating the argument from the previous case.
3. $L < 2\omega(N)$. Then clearly, $L \leq 3^{\omega(N)}$.
4. W.l.o.g none of the above conditions hold.
   As $\omega(N) \leq \log N$ and N has no factors less then $2\log N$, it follows that N is $(2\omega(N))$-rough. By Lemma 23, N has a factor smaller then $N^{1/L}$. On the other hand, by Theorem 22, N has no factors smaller then $N^{\frac{1}{3^{\omega(N)-1}}}$. Consequently, $L \leq 3^{\omega(N)-1} \leq 3^{\omega(N)}$.
   Finally, by Lemma 19: $\omega(N) \leq L$. ◀

Theorem 1 follows by combining the above Theorem with Part 4 of Lemma 13.

4 Examples

We use Algorithm 1 (ANF) to find an upper bound for the number of prime factors of a square-free integer N, given in addition $\Phi(N)$.

4.1 Example 1

Let us find the number of prime factors of

$$N = 50000000479987 = 3005349551 \times 127 \times 131,$$

where $\alpha_1 = 0.691869, \alpha_2 = 0.154557, \alpha_3 = 0.153574$. The algorithm will first find that it does not satisfy the base case, then iterate from $M = 1$ to $2 \log N (= 91)$. After that, it will invoke the algorithm from Theorem 22, since $\alpha_1 \geq 2(\alpha_2 + \alpha_3)$. That algorithm will return $p_1 = 3005349551$. Then it will return $\mathrm{ANF}(16637) + \mathrm{ANF}(3005349551) = 3$, since both will satisfy the base cases.

4.2 Example 2

We will find the number of prime factors of

$$N = 504203634089 = 389 \times 331 \times 283 \times 137 \times 101,$$

where $\alpha_1 = 0.221314$, $\alpha_2 = 0.215322$, $\alpha_3 = 0.209508$, $\alpha_4 = 0.182585$, $\alpha_5 = 0.171271$. The algorithm will first find that it does not satisfy the base case, then iterate from $M = 1$ to $2 \log N (= 54)$. Since none of M-s divides N, it will then invoke the algorithm from Theorem 22. As no $1 \leq r < 5$ satisfies

$$\alpha_r \geq 2 \sum_{i=r+1}^{5} \alpha_i,$$

the algorithm will compute the approximation: i.e. return the largest ℓ satisfying $\Phi(N)^{1/\ell} \leq N^{1/\ell} - 1$. As $\Phi(N) = 491059008000$, the algorithm outputs $L = 6 < 3^5$.

5 Discussion & Open Questions

In this paper we give the first efficient *deterministic* algorithm that approximates $\omega(N)$, given oracle access to $\Phi(\cdot)$. In addition, in our approach we attempt to compute/approximate $\omega(N)$ "directly" rather than factoring N first (although we may eventually end up succeeding to factor N). Below are some open questions.

- The immediate open question is, of course, to compute $\omega(N)$ exactly or at least improve the approximation ratio. As was noted, we believe that the exact computation of $\omega(N)$, given oracle access to $\Phi(\cdot)$, would constitute an important milestone. See discussion after Observation 5 for more details.
- Can we extend this result to other "interesting" functions that can be efficiently computed given the complete factorization of N?
- In particular, can we devise an efficient algorithm for computing the Möbius function $\mu(N)$? Given the results of [8, 14], this would be equivalent to determining the parity of $\omega(N)$.

References

1 L. M. Adleman and K. S. McCurley. Open problems in number theoretic complexity, II. In *Algorithmic Number Theory, First International Symposium, ANTS-I, Ithaca, NY, USA, May 6-9, 1994, Proceedings*, pages 291–322, 1994. doi:10.1007/3-540-58691-1.

2 M. Agrawal, N. Kayal, and N. Saxena. Primes is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

3 E. Bach. Intractable problems in number theory. In *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, pages 77–93, 1988. doi:10.1007/0-387-34799-2.

4 E. Bach, G. L. Miller, and J. Shallit. Sums of divisors, perfect numbers and factoring. *SIAM J. Comput.*, 15(4):1143–1154, 1986. doi:10.1137/0215083.

5 D. Coppersmith. Finding a small root of a univariate modular equation. In *Advances in Cryptology - EUROCRYPT*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 1996. doi:10.1007/3-540-68339-9.

6 M. Hittmeir and J. Pomykala. Deterministic integer factorization with oracles for euler's totient function. *Fundam. Inform.*, 172(1):39–51, 2020. doi:10.3233/FI-2020-1891.

7 J. Kim, I. Volkovich, and N. X. Zhang. The power of leibniz-like functions as oracles. In *The 15th International Computer Science Symposium in Russia, CSR*, volume 12159 of *Lecture Notes in Computer Science*, pages 263–275. Springer, 2020. doi:10.1007/978-3-030-50026-9.

8 S. Landau. Some remarks on computing the square parts of integers. *Inf. Comput.*, 78(3):246–253, 1988. doi:10.1016/0890-5401(88)90028-4.

9 D. L. Long. Random equivalence of factorization and computation of orders. Technical Report 284, Princeton University, 1981.

10 G. L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13(3):300–317, 1976. doi:10.1016/S0022-0000(76)80043-8.

11 F. Morain, G. Renault, and B. Smith. Deterministic factoring with oracles. *CoRR*, abs/1802.08444, 2018. arXiv:1802.08444.

12 M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980.

13 J. Shallit and A. Shamir. Number-theoretic functions which are equivalent to number of divisors. *Inf. Process. Lett.*, 20(3):151–153, 1985. doi:10.1016/0020-0190(85)90084-5.

14 H. Woll. Reductions among number theoretic problems. *Inf. Comput.*, 72(3):167–179, 1987. doi:10.1016/0890-5401(87)90030-7.

15 B. Zralek. A deterministic version of pollard's p-1 algorithm. *Math. Comput.*, 79(269):513–533, 2010. doi:10.1090/S0025-5718-09-02262-5.

A Missing Proofs

Sketch of the proof of Part 4 of Lemma 13. Consider the following quadratic equation

$$f(x) \triangleq x^2 - (N - \Phi(N) + 1)x + N = 0.$$

Observe that $f(x) = (x-p)(x-q)$. Hence, the solutions of the equation are exactly p and q. ◀

Fully Dynamic Algorithms for Knapsack Problems with Polylogarithmic Update Time

Franziska Eberle
Faculty of Mathematics and Computer Science, University of Bremen, Germany

Nicole Megow
Faculty of Mathematics and Computer Science, University of Bremen, Germany

Lukas Nölke
Faculty of Mathematics and Computer Science, University of Bremen, Germany

Bertrand Simon
IN2P3 Computing Center, CNRS, Villeurbanne, France

Andreas Wiese
Department of Industrial Engineering, University of Chile, Santiago, Chile

Abstract

Knapsack problems are among the most fundamental problems in optimization. In the Multiple Knapsack problem, we are given multiple knapsacks with different capacities and items with values and sizes. The task is to find a subset of items of maximum total value that can be packed into the knapsacks without exceeding the capacities. We investigate this problem and special cases thereof in the context of *dynamic algorithms* and design data structures that efficiently maintain near-optimal knapsack solutions for dynamically changing input. More precisely, we handle the arrival and departure of individual items or knapsacks during the execution of the algorithm with worst-case update time polylogarithmic in the number of items. As the optimal and any approximate solution may change drastically, we maintain implicit solutions and support polylogarithmic time query operations that can return the computed solution value and the packing of any given item.

While dynamic algorithms are well-studied in the context of graph problems, there is hardly any work on packing problems (and generally much less on non-graph problems). Motivated by the theoretical interest in knapsack problems and their practical relevance, our work bridges this gap.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases Fully dynamic algorithms, knapsack problem, approximation schemes

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.18

Related Version *Full Version*: `https://arxiv.org/abs/2007.08415`

Funding This work is partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project Number 146371743 – TRR 89 Invasive Computing.

Acknowledgements We thank Martin Böhm, Peter Kling, and Jens Schlöter for the fruitful discussions.

1 Introduction

Knapsack problems are among the most fundamental optimization problems. In their most basic form, we are given a knapsack capacity $S \in \mathbb{N}$ and a set of n items, where each item $j \in [n] := \{1, 2, \ldots, n\}$ has a size $s_j \in \mathbb{N}$ and a value $v_j \in \mathbb{N}$. The Knapsack problem asks for a subset of items, $P \subseteq [n]$, with maximal total value $v(P) := \sum_{j \in P} v_j$ and with a total size $s(P) := \sum_{j \in P} s_j$ that does not exceed the knapsack capacity S. In the more general Multiple Knapsack problem, we are given m knapsacks with capacities S_i for $i \in [m]$.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 18; pp. 18:1–18:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Here, the task is to select m disjoint subsets $P_1, P_2, \dots, P_m \subseteq [n]$ such that subset P_i satisfies the capacity constraint $s(P_i) \leq S_i$ and the total value of all subsets $\sum_{i\in[m]} v(P_i)$ is maximized.

Multiple Knapsack is strongly NP-hard, even for identical knapsack capacities, as it is a special case of bin packing. Knapsack, on the other hand, is only weakly NP-hard and admits pseudo-polynomial time algorithms, the first one being already published in the 1950s [5].

As a consequence of these hardness results, each of the knapsack variants has been studied extensively through the lens of approximation algorithms. Of particular interest are *approximation schemes*, families of polynomial-time algorithms that compute for each $\varepsilon > 0$ a $(1-\varepsilon)$-approximate solution, i.e., a feasible solution with value within a factor of $(1-\varepsilon)$ of the optimal solution value. Based on the dependency on ε of the respective running time, we distinguish *Polynomial Time Approximation Schemes (PTAS)* with arbitrary dependency on ε, *Efficient PTAS (EPTAS)* where arbitrary functions $f(\varepsilon)$ may only appear as a multiplicative factor, and *Fully Polynomial Time Approximation Schemes (FPTAS)* with polynomial dependency on $\frac{1}{\varepsilon}$.

The first approximation scheme for Knapsack was an FPTAS by Ibarra and Kim [43] and initiated a long sequence of follow-up work, which is still active [17, 52]. Multiple Knapsack is substantially harder and does not admit an FPTAS, unless P = NP, even with two identical knapsacks [19]. However, approximation schemes with running times of the form $n^{f(\varepsilon)}$ (PTASs) are known [19, 54] as well as improvements to only $f(\varepsilon)n^{\mathcal{O}(1)}$ (EPTASs) [48, 50]. All these algorithms are *static* in the sense that the full instance is given to an algorithm and is then solved.

Given the ubiquitous dynamics of real-world instances, it is natural to ask for *dynamic algorithms* that adapt to small changes in the packing instance while spending only little computation time. More precisely, during the execution of the algorithm, items and knapsacks arrive and depart and the algorithm needs to maintain an approximate knapsack solution with an *update time* polylogarithmic in the number of items in each step. A dynamic algorithm is then a data structure that implements these updates efficiently and supports relevant query operations.

A practical application is the dynamic estimation of the profit for scheduling jobs in computing clusters in which virtual machines can be moved among physical machines [6]. This allows the service provider to adapt the provided capacity, i.e., the currently running servers, to the current demand, see, e.g., [13, 23, 59]. An efficient framework for Multiple Knapsack can be viewed as a first-stage decision tool: In real-time, it determines whether the customer in question should be allowed into the system based on the cost of possibly powering and using additional servers. As the service provider has to decide immediately which request she wants to accept, she needs to obtain the information *fast*, i.e., sublinear in the number of requests already in the system.

Generally, dynamic algorithms constitute a vibrant research field in the context of graph problems. We refer to surveys [15, 26, 38] for an overview on dynamic graph algorithms. Interestingly, only for a small number of graph problems there are dynamic algorithms known with *polylogarithmic* update time, among them connectivity problems [40, 42], the minimum spanning tree [42], and vertex cover [9, 11]. Recently, this was complemented by conditional lower bounds that are typically *linear* in the number of nodes or edges; see, e.g., [2]. Over the last few years, the generalization of dynamic vertex cover to dynamic set cover gained interest leading to near-optimal approximation algorithms with polylogarithmic update times [1, 8, 10, 34]. Also, recently, algorithms have been developed for maintaining maximal independent sets, e.g., [4, 18, 64], and approximate maximum independent sets in special graph classes [12, 20, 39].

For packing problems, there are hardly any dynamic algorithms with small update time known. A notable exception is a result for bin packing that maintains a $\frac{5}{4}$-approximative solution with $\mathcal{O}(\log n)$ update time [45]. This lack of efficient dynamic algorithms is in stark contrast to the aforementioned intensive research on computationally efficient algorithms for packing problems. Our work bridges this gap initiating the design of data structures and algorithms that efficiently maintain near-optimal solutions.

Our Contribution. In this paper, we present dynamic algorithms for maintaining approximate solutions for three problems of increasing complexity: Knapsack, Multiple Knapsack with identical knapsack sizes, and general Multiple Knapsack. Our algorithms are *fully dynamic* which means that in an update operation they can handle the arrival or departure of an item and of a knapsack. Further, we consider the *implicit solution* or *query* model, in which an algorithm is not required to store the solution explicitly in memory such that the solution can be read in linear time at any given point of the execution. Instead, the algorithm may maintain the solution implicitly with the guarantee that a query about the packing can be answered in polylogarithmic time.

We give *worst-case* guarantees for update and query times that are polylogarithmic in n, the number of items currently in the input, and bounded by a function of $\varepsilon > 0$, the desired approximation accuracy. For some special cases, we can even ensure a polynomial dependency on $\frac{1}{\varepsilon}$. In others, we justify the exponential dependency with corresponding lower bounds. Denote by $v_{\max}$ the currently largest item value and by $\overline{v}$ an upper bound on $v_{\max}$ that is known in advance.

1. For Multiple Knapsack, we design a dynamic algorithm maintaining a $(1-\varepsilon)$-approximate solution with update time $2^{f(1/\varepsilon)}\left(\frac{1}{\varepsilon}\log n \log \overline{v}\right)^{\mathcal{O}(1/\varepsilon)}(\log S_{\max})^{\mathcal{O}(1)}$, where f is quasi-linear, and query time $\left(\frac{1}{\varepsilon}\log n\right)^{\mathcal{O}(1)}$.
2. The exponential dependency on $\frac{1}{\varepsilon}$ in the update time for Multiple Knapsack is indeed necessary, even for two identical knapsacks. We show that there is no $(1-\varepsilon)$-approximate dynamic algorithm with update time $\left(\frac{1}{\varepsilon}\log n\right)^{\mathcal{O}(1)}$, unless $\mathrm{P} = \mathrm{NP}$.
3. For Knapsack, we give a dynamic $(1-\varepsilon)$-approximation algorithm with update time $\left(\frac{1}{\varepsilon}\log(nv_{\max})\right)^{\mathcal{O}(1)} + \mathcal{O}\left(\frac{1}{\varepsilon}\log n \log \overline{v}\right)$ and constant query times.
4. For Multiple Knapsack with *identical knapsacks* with capacity S each, we improve the update time to $\left(\frac{1}{\varepsilon}\log n \; \log v_{\max} \; \log S\right)^{\mathcal{O}(1)}$ if $m \geq \frac{16}{\varepsilon^7}\log^2 n$ with query time $\left(\frac{1}{\varepsilon} \; \log n\right)^{\mathcal{O}(1)}$.

In each update step, we compute only implicit solutions and provide query operations for the solution value, the knapsack of a queried item, and the complete solution. These queries are consistent between two update steps and run efficiently, i.e., run in time polynomial in $\log n$ and $\log \overline{v}$ and linear in the output size. We remark that it is not possible to maintain a solution with a non-trivial approximation guarantee explicitly with only polylogarithmic update time (even amortized) since it might be necessary to change $\Omega(n)$ items per iteration, e.g., if a very large and very profitable item is inserted and removed in each iteration.

We remark that our result yields a static algorithm with a near-linear running time in n.

Our Techniques. Maybe surprisingly, we recompute a $(1-\varepsilon)$-approximate solution from scratch in polylogarithmic time after each update. More precisely, we compute a $(1-\varepsilon)$-estimate of the value of Opt and additionally store all information that is needed in order to answer any query in polylogarithmic time. Interestingly, this shows that for such computations, we do not need exact knowledge about the whole input, but only a small amount of information of polylogarithmic size. We show that this information can be extracted efficiently from suitable data structures in which we store the input items and knapsacks. Even more, we show that we can maintain these data structures in polylogarithmic time per update.

On a high level, we reduce the overall problem to two subproblems solved independently. In the first one, we are given only few knapsacks, $m = \left(\frac{1}{\varepsilon}\log n\right)^{\mathcal{O}(1)}$ many, which are the largest knapsacks in the original input. Here, we observe that if we select the $\frac{m}{\varepsilon}$ most valuable items in the optimal solution correctly, we can afford to fill the remaining space in the knapsacks greedily, i.e., highest density (value divided by size) first, and charge the resulting loss to the valuable items. We cannot guess these most valuable items explicitly, but we show that we can select a small set of candidates for these items and guess a few placeholder items for the remaining ones. This yields an instance with only $\left(\frac{1}{\varepsilon}\log n\right)^{\mathcal{O}(1)}$ items on which we run a known EPTAS for Multiple Knapsack [50] yielding a running time of $\left(\frac{1}{\varepsilon}\log n\right)^{\mathcal{O}(1)}$. For the special case of a single knapsack, we show that we can invoke an FPTAS instead, which improves the running time.

In the second subproblem, we are given a potentially large set of knapsacks, and we are allowed to use an additional set of $\left(\frac{1}{\varepsilon}\log n\right)^{\Theta(1)}$ knapsacks that the optimal solution does not use (resource augmentation). We introduce a technique that we call *oblivious linear grouping*. Linear grouping is a standard technique used in order to round a set of one-dimensional items that need to be packed into a given set of containers (e.g., in bin packing), such that they have at most $\frac{1}{\varepsilon}$ different sizes after the rounding (at the expense of leaving an ε-fraction of the items out). However, in our setting we do not know a priori which input items need to be packed, and therefore we cannot apply this technique directly. Instead, we show that we can round the input items to $(\frac{1}{\varepsilon}\log n)^{\mathcal{O}(1)}$ different sizes such that we lose at most a factor of $(1-\varepsilon)$ *independently* of what the optimal solution looks like. In fact, our rounding method is even oblivious to the input knapsacks. Therefore, we believe that it might be useful also for other dynamic packing problems or for speeding up static algorithms. After rounding the items to $\left(\frac{1}{\varepsilon}\log n\right)^{\mathcal{O}(1)}$ different sizes, we set up a configuration-LP that has a configuration for each possible set of relatively large items that together fit inside a knapsack. Thanks to our rounding, there are only polylogarithmically many configurations and we can solve this LP in time $\left(\frac{1}{\varepsilon}\log n\right)^{\mathcal{O}(1/\varepsilon)}$. We use the additional knapsacks in order to compensate errors when rounding the LP, i.e., due to rounding up the fractional variables and adding small items greedily into the remaining space of the knapsacks. Special care is necessary since the sizes of the knapsacks can differ and hence some item might be relatively large in some knapsack, but relatively small in another knapsack.

Further Related Work. Since the first approximation scheme for Knapsack [43] running times have been improved steadily [17,30,31,52,55,58,67] with $\mathcal{O}(n\log\frac{1}{\varepsilon}+(\frac{1}{\varepsilon})^{9/4})$ by Jin [52] being the currently fastest. Recent work on conditional lower bounds [22,57] implies that Knapsack does not admit an FPTAS with running time of $\mathcal{O}((n+\frac{1}{\varepsilon})^{2-\delta})$, for any $\delta > 0$, unless $(min,+)$-convolution has a subquadratic algorithm [17,65].

A PTAS for Multiple Knapsack was first presented by Chekuri and Khanna [19] and EPTASs due to Jansen [48,50] are also known. The fastest of these algorithms [50] has a running time of $2^{\mathcal{O}(\log^4(1/\varepsilon)/\varepsilon)} + n^{\mathcal{O}(1)}$. The mentioned algorithms are all static and assume full knowledge about the instance for which a complete solution has to be found. In particular, their solutions might change completely when a single item is added to the input which makes a full recomputation necessary. The algorithm in [19] invokes a guessing step with $n^{f(1/\varepsilon)}$ many options which are too many for a polylogarithmic update time. The EPTASs in [48,50] use a configuration linear program of size $\Omega(n)$ which is also prohibitively large for such an update time.

The dynamic arrival and removal of items exhibits some similarity to knapsack models with incomplete information. For example, in the *online* knapsack problem [61] items arrive online one by one. When an item arrives, an algorithm must irrevocably accept or reject it before the next item arrives. Various problem variants have been studied, e.g., with resource augmentation [47], the removable online knapsack problem [21, 35–37, 46], and with advice [14]. Other models with uncertainty in the item set or the knapsack capacity include the *stochastic* knapsack problem [7, 25, 60] and *robust* knapsack problems [16, 27, 62, 70]. Related to our setting are also online models with a softened irrevocability requirement, e.g., online optimization with *recourse* [29, 33, 44, 63] or *migration* [51, 68, 69] allows to adapt previously taken decisions in a limited way. We are not aware of work on knapsack problems in these settings and, again, the goal is to bound the amount of change needed to maintain good online solutions regardless of the computational effort.

2 Roadmap and Preliminaries

First, in this section, we formalize the operations that our data structures support, describe auxiliary data structures that we need, and define how we round the item values. Then, in Section 3, we describe algorithms for one knapsack and for a polylogarithmic number of knapsacks. In Section 4, we present an algorithm for (many) identical knapsacks and an algorithm under resource augmentation (in the form of a polylogarithmic number of additional knapsacks) in the setting of (many) knapsacks with possibly different capacities. Finally, we present in Section 5 an algorithm for the general case that uses the previously mentioned algorithms as subroutines. Additionally, in the full version of this paper [28], we show that our update time cannot be improved to $(\log n/\varepsilon)^{\mathcal{O}(1)}$, unless P=NP.

From the perspective of a data structure that implicitly maintains near-optimal solutions for Multiple Knapsack, our algorithms support several update and query operations which are listed below. They allow for the output of (parts of) the current solution, or for specific changes to the input of Multiple Knapsack, causing the computation of a new solution.

- **Insert (Remove) Item:** Inserts (removes) an item into (from) the input.
- **Insert (Remove) Knapsack:** Inserts (removes) a knapsack into (from) the input.

A new solution can be output, entirely or in parts, using the following query operations.

- **Query Item j:** Returns whether item j is packed in the current solution and if this is the case, additionally returns the knapsack containing it.
- **Query Solution Value:** Returns the value of the current solution.
- **Query Entire Solution:** Returns all items in the current solution, together with the information in which knapsack each such item is packed.

Importantly, queries are consistent in-between two update operations. However, their answers are not independent of each other but depend on the queries as well as their order.

For simplicity, we assume that elementary operations (e.g., additions) can be handled in constant time. Additionally, we assume without loss of generality that $\frac{1}{\varepsilon} \in \mathbb{N}$. We also assume that at the very beginning we start with no items and no knapsacks, and initialize all needed auxiliary data structures accordingly. If one wants to start with a specific set of items and/or knapsacks, one can insert them with our insertion routines, using polylogarithmic time per insertion.

Auxiliary Data Structures. We employ auxiliary data structures in which we store (subsets of) input items and input knapsacks, sorted according to some specific values, e.g., size or capacity. We need to be able to quickly access elements, compute the largest prefix of elements such that the sum according to some property, e.g., the total size, is below a given threshold, and compute in such a prefix the sum according to some element property, e.g., the total value. Note that these prefixes are w.r.t. the fixed ordering of the elements, while the element property for the threshold or computing the sum might be different. To this end, we employ as an auxiliary data structure a variation of balanced search trees that store elements according to some given ordering. For computing the mentioned prefix sums, we store in each internal node v the sums of the elements in the subtree rooted at v according to each property, e.g., size, value, or capacity. When we need to compute some largest prefix, we simply output the index of its last element.

▶ **Lemma 1.** *There is a data structure maintaining a sorting of n' elements w.r.t. some key value such that (i) insertion, deletion, or search by key value of an element takes $\mathcal{O}(\log n')$ time, and (ii) prefixes and prefix sums w.r.t. any element property can be computed in time $\mathcal{O}(\log n')$.*

Rounding Values. A crucial ingredient of our algorithms is the partitioning of items into only few *value classes* V_ℓ, where for each ℓ the class V_ℓ consists of each input item j with $(1+\varepsilon)^\ell \leq v_j < (1+\varepsilon)^{\ell+1}$. Upon arrival of some item j, we calculate the index ℓ_j such that $j \in V_{\ell_j}$ and store the tuple (j, v_j, s_j, ℓ_j) representing j in the auxiliary data structures of the respective algorithm. In the following, we pretend for each ℓ that each item in V_ℓ has value $(1+\varepsilon)^\ell$, which loses only a factor of $\frac{1}{1+\varepsilon}$ in the total profit of any solution.

▶ **Lemma 2.** *(i) There are at most $\mathcal{O}\big(\frac{\log v_{\max}}{\varepsilon}\big)$ many value classes. (ii) For optimal solutions* Opt *and* Opt$'$ *for the original and rounded instance, $v(\text{Opt}') \geq (1-\varepsilon)\cdot v(\text{Opt})$.*

3 A Single Knapsack

In this section, we first present a dynamic algorithm for the case of one single knapsack, summarized in the following theorem. Afterwards, we will argue how to extend our techniques to the setting of a polylogarithmic number of knapsacks.

▶ **Theorem 3.** *For $\varepsilon > 0$, there is a fully dynamic algorithm for* Knapsack *that maintains $(1-\varepsilon)$-approximate solutions with update time $\mathcal{O}\big(\frac{\log^4(nv_{\max})}{\varepsilon^9}\big) + \mathcal{O}\big(\frac{1}{\varepsilon}\log n \log \overline{v}\big)$. Furthermore, queries of single items and the solution value can be answered in time $\mathcal{O}(1)$.*

We partition the items in the optimal solution Opt into high- and low-value items, respectively. The high-value items are the $\frac{1}{\varepsilon}$ most valuable items of Opt, and the low-value items are the remaining items of Opt. We compute a small set of candidate items $H_{\frac{1}{\varepsilon}}$ that intuitively contains all relevant high-value items in Opt. Also, we guess a placeholder item for the low-value items, that is large enough to accomodate low-value items of enough profit fractionally. We can assume that in an optimal fractional solution (of low-value items) at most one item is selected non-integrally. Hence, we can drop this item and charge it to the $\frac{1}{\varepsilon}$ high-value items. This results in a knapsack instance with only $\mathcal{O}\left(\frac{1}{\varepsilon^3}\right)$ items which we solve with an FPTAS.

Formally, denote by $\text{Opt}_{\frac{1}{\varepsilon}}$ a set of $\frac{1}{\varepsilon}$ most valuable items of Opt. We break ties by picking smaller items. Denote by $V_{\ell_{\max}}$ and $V_{\ell_{\min}}$ the highest resp. lowest value class of an element in $\text{Opt}_{\frac{1}{\varepsilon}}$ and let $n_{\min} := |\text{Opt}_{\frac{1}{\varepsilon}} \cap V_{\ell_{\min}}| \leq \frac{1}{\varepsilon}$. Furthermore, denote by $\mathcal{V}_L$ the value of the items in $\text{Opt} \setminus \text{Opt}_{\frac{1}{\varepsilon}}$, rounded down to the next power of $(1+\varepsilon)$. To efficiently implement our algorithm, we maintain several data structures, using Lemma 1. We store items of each non-empty value class V_ℓ (at most $\log_{1+\varepsilon} v_{\max}$) in a data structure ordered non-decreasingly by size. Second, for each possible value class V_ℓ (at most $\log_{1+\varepsilon}\overline{v}$), we maintain a data structure that contains each input item j with $j \in V_{\ell'}$ for some $\ell' \le \ell$, ordered non-increasingly by density $\frac{v_j}{s_j}$. In particular, we maintain such a data structure even if V_ℓ itself is empty (since the data structure might still contain items from classes $V_{\ell'}$ with $\ell' < \ell$). This leads to the additive term in the update time of $\mathcal{O}(\log n \log_{1+\varepsilon}\overline{v})$. We use additional auxiliary data structures to store our solution and support queries.

Algorithm. The algorithm computes an implicit solution as follows.

1) **Compute a set $H_{\frac{1}{\varepsilon}}$ of high-value candidates:** Guess the values $\ell_{\max}$, $\ell_{\min}$, and $n_{\min}$. If $(1+\varepsilon)^{\ell_{\min}} \ge \varepsilon^2 \cdot (1+\varepsilon)^{\ell_{\max}}$, define $H_{\frac{1}{\varepsilon}}$ to be the set containing the $\frac{1}{\varepsilon}$ smallest items of each of the value classes $V_{\ell_{\min}+1}, \dots, V_{\ell_{\max}}$, plus the $n_{\min}$ smallest items from $V_{\ell_{\min}}$. Otherwise, set $H_{\frac{1}{\varepsilon}}$ to be the union of the $\frac{1}{\varepsilon}$ smallest items of each of the value classes with values in $[\varepsilon^2 \cdot (1+\varepsilon)^{\ell_{\max}}, (1+\varepsilon)^{\ell_{\max}}]$.
2) **Create a placeholder item B:** Guess $\mathcal{V}_L$ and consider items with value at most $(1+\varepsilon)^{\ell_{\min}}$ sorted by density. Remove the $n_{\min}$ smallest items of $V_{\ell_{\min}}$ until the next iteration. For the remaining items, compute the minimal size of fractional items necessary to reach a value $\mathcal{V}_L$. We do this via prefix sum computations on the data structure that contains all items in $V_{\ell'}$ for each $\ell' \le \ell_{\min}$, ordered non-increasingly by density. Then B is given by $v_B = \mathcal{V}_L$ and with s_B equal to the size of those low-value items.
3) **Use an FPTAS:** On the instance I, consisting of $H_{\frac{1}{\varepsilon}}$ and the placeholder item B, run an FPTAS parameterized by ε (we use the one by Jin [52]) to obtain a packing P.
4) **Implicit solution:** Among all guesses, keep the solution P with the highest value. Pack items from $H_{\frac{1}{\varepsilon}}$ as in P and, if $B \in P$, also pack the low-value items completely contained in B (note that at most one item is packed fractionally in B). While used candidate items from $H_{\frac{1}{\varepsilon}}$ can be stored explicitly, low-value items are given only implicitly by saving the correct guesses and computing membership in B on a query.

Analysis. We show that the above algorithm attains an approximation ratio of $(1-\varepsilon)$. A factor of $(1-\varepsilon)$ is lost due to the approximation ratio of the FPTAS. An additional factor of $(1-\varepsilon)$ is lost in each of the following steps. To obtain a candidate set $H_{\frac{1}{\varepsilon}}$ of constant cardinality, we restrict the item values to $[\varepsilon^2 \cdot (1+\varepsilon)^{\ell_{\max}}, (1+\varepsilon)^{\ell_{\max}}]$. Since $|\textsc{Opt}_{\frac{1}{\varepsilon}}| = \frac{1}{\varepsilon}$, this excludes items from $\textsc{Opt}$ with a total value of at most $\frac{1}{\varepsilon} \cdot \varepsilon^2\,(1+\varepsilon)^{\ell_{\max}} \le \varepsilon \cdot \textsc{Opt}$. Furthermore, due to guessing $\mathcal{V}_L$ up to a power of $(1+\varepsilon)$, we get $v_B = \mathcal{V}_L \ge \frac{1}{1+\varepsilon} \cdot v(\textsc{Opt} \setminus \textsc{Opt}_{\frac{1}{\varepsilon}})$. Finally, in Step 2, at most one item was cut fractionally. It is charged to the $\frac{1}{\varepsilon}$ items of $\textsc{Opt}_{\frac{1}{\varepsilon}}$, using that each of them has a larger value.

The running time can be verified easily by multiplying the numbers of guesses for each value as well as the running time of the FTPAS. The latter is $\mathcal{O}\left(\frac{1}{\varepsilon^4}\right)$, since we designed $H_{\frac{1}{\varepsilon}}$ to contain only a constant number of items, namely $\mathcal{O}\left(\frac{1}{\varepsilon^3}\right)$ many.

Queries. We show how to efficiently handle the different types of queries.

- **Single Item Query:** If the queried item is contained in $H_{\frac{1}{\varepsilon}}$, its packing was saved explicitly. Otherwise, if B is packed, we save the last, i.e., least dense, item contained entirely in B. By comparing with this item, membership in B can be decided in constant time on a query.
- **Solution Value Query:** While the algorithm works with rounded values, we use the data structures of Lemma 1 to retrieve the actual item values. We store the actual solution value in the update step by adding the actual values of the packed items from $H_{\frac{1}{\varepsilon}}$ and determining the actual value of items in B with a prefix computation. On query, we return the stored value.

- **Query Entire Solution:** Output the stored packing of candidates. If B was packed, iterate over items in B in the respective density-sorted data structure and output them.

Polylogarithmically many knapsacks. One can show that the queries can be performed in the claimed running times which completes the proof of Theorem 3, see the full version of this paper [28]. We can extend the above technique to the setting of m knapsacks, at the expense of increasing the update time and query time by a factor $m^{\mathcal{O}(1)}$, and using an EPTAS for Multiple Knapsack [50] instead of an FPTAS (see [28]).

▶ **Theorem 4.** *For $\varepsilon > 0$, there is a dynamic algorithm for Multiple Knapsack that achieves an approximation factor of $(1-\varepsilon)$ with update time $2^{f(1/\varepsilon)}\big(\frac{m}{\varepsilon}\log(nv_{\max})\big)^{\mathcal{O}(1)} + \mathcal{O}\big(\frac{1}{\varepsilon}\,\log\bar{v}\,\log n\big)$, with f quasi-linear. Item queries are answered in time $\mathcal{O}\big(\log\frac{m^2}{\varepsilon^6}\big)$, solution value queries in time $\mathcal{O}(1)$, and queries of one knapsack or the entire solution in time linear in the output.*

4 Identical Knapsacks

In this section, we present our algorithm for an arbitrary (large) number of identical knapsacks. Also, we describe an extension to the case where the knapsacks have different sizes and we can use some additional knapsacks as resource augmentation.

4.1 Oblivious Linear Grouping

We start with our oblivious linear grouping routine that we use in order to round the item sizes, aiming at only few different types of items. We say that two items j, j' are of the same *type* if $\{j,j'\}\subseteq V_\ell$ for some ℓ and if $s_j = s_{j'}$. We round the items implicitly, i.e., we compute thresholds $\{\bar{s}_1,...,\bar{s}_k\}$ and we round up the size s_j of each item j to the next larger value in this set.

▶ **Lemma 5.** *Given a set J' with $|\mathrm{OPT}\cap J'|\le n'$ for all optimal solutions OPT, there is an algorithm with running time $\mathcal{O}\big(\frac{\log^5 n'}{\varepsilon^5}\big)$ that rounds the items in J' to item types $\mathcal{T}$ with $|\mathcal{T}|\le\mathcal{O}\big(\frac{\log^2 n'}{\varepsilon^4}\big)$ and ensures $v(\mathrm{OPT}_{\mathcal{T}})\ge\frac{(1-\varepsilon)(1-2\varepsilon)}{(1+\varepsilon)^2}v(\mathrm{OPT})$. Here, $\mathrm{OPT}_{\mathcal{T}}$ is the optimal solution attainable by packing item types $\mathcal{T}$ instead of the items in J' and using $J\setminus J'$ as is.*

Algorithm. In the following, we use the notation X' for a set X to refer to $X\cap J'$ while X'' refers to $X\setminus J'$. Recall that item values of items in J are rounded to powers of $1+\varepsilon$ to create the value classes V_ℓ where each item $j\in V_\ell$ has value $(1+\varepsilon)^\ell$. We guess $\ell_{\max}$ which is defined to be the guess for the highest value ℓ with $V'_\ell\cap\mathrm{OPT}\ne\emptyset$ and let $\bar{\ell}:=\ell_{\max}-\lceil\log_{1+\varepsilon}(n'/\varepsilon)\rceil$.

1) For each ℓ with $\bar{\ell}\le\ell\le\ell_{\max}$ and each $n_\ell=(1+\varepsilon)^k$ with $0\le k\le\log_{1+\varepsilon}n'$ do: Consider the n_ℓ smallest elements of V'_ℓ (sorted by increasing size) and determine the $\frac{1}{\varepsilon}$ many (almost) equal-sized groups $G_1(n_\ell),\dots,G_{1/\varepsilon}(n_\ell)$ of $\lceil\varepsilon n_\ell\rceil$ or $\lfloor\varepsilon n_\ell\rfloor$ elements. If $\varepsilon n_\ell\notin\mathbb{N}$, ensure that $|G_k(n_\ell)|\le|G_{k'}(n_\ell)|\le|G_k(n_\ell)|+1$ for $k\le k'$. If $\frac{1}{\varepsilon}$ is not a natural power of $(1+\varepsilon)$, create $G_1(\frac{1}{\varepsilon}),\dots,G_{1/\varepsilon}(\frac{1}{\varepsilon})$ where $G_k(\frac{1}{\varepsilon})$ is the kth smallest item in V'_ℓ. Let $G_1(n_\ell),\dots,G_{1/\varepsilon}(n_\ell)$ be the corresponding groups sorted increasingly by the size of the items. Let $j_k(n_\ell)=\max\{j : j\in G_k(n_\ell)\}$ be the last index belonging to group $G_k(n_\ell)$. After having determined $j_k(n_\ell)$ for each possible value n_ℓ (including $\frac{1}{\varepsilon}$) and for each $1\le k\le\frac{1}{\varepsilon}$, the size of each item j is rounded up to the size of the next larger item j' such that there exists k and ℓ satisfying $j'=j_k(n_\ell)$.
2) Discard each item j with $j\in V'_\ell$ for $\ell<\bar{\ell}$.

Analysis. Despite the new approach to apply linear grouping simultaneously to many possible values of n_ℓ, the analysis builds on standard techniques. The loss in the objective function due to rounding item values is bounded by a factor of $\frac{1}{1+\varepsilon}$ by Lemma 2. As $\bar{\ell}$ is chosen such that n' items of value at most $(1+\varepsilon)^{\bar{\ell}}$ contribute less than an ε-fraction of $\textsc{Opt}'$, the loss in the objective function by discarding items in value classes V'_ℓ with $\ell < \bar{\ell}$ is bounded by a factor $(1-\varepsilon)$. By taking only $(1+\varepsilon)^{\lfloor \log_{1+\varepsilon} n_\ell \rfloor}$ items of V'_ℓ instead of n_ℓ, we lose at most a factor $\frac{1}{1+\varepsilon}$. The groups created by oblivious linear grouping are an actual refinement of the groups created by classical linear grouping. Thus, we pack our items similarly: not packing the group with the largest items (at the loss of a factor of $(1-2\varepsilon)$) allows us to "move" all rounded items of group $G_k(n_\ell)$ to the positions of the (not rounded) items in group $G_{k+1}(n_\ell)$. Combining, we obtain $v(\textsc{Opt}_{\mathcal{T}}) \geq \frac{(1-\varepsilon)(1-2\varepsilon)}{(1+\varepsilon)^2} v(\textsc{Opt})$.

Since $\mathcal{T}$ contains at most $\frac{1}{\varepsilon}\big(\big\lceil \frac{\log n'/\varepsilon}{\log(1+\varepsilon)} \big\rceil + 1\big)$ different value classes, and as it suffices to use $\big\lceil \frac{\log n'}{\log(1+\varepsilon)} \big\rceil + 1$ many different values for $n_\ell = |\textsc{Opt} \cap V'_\ell|$, we have $|\mathcal{T}| \leq \mathcal{O}(\frac{\log^2 n'}{\varepsilon^4})$. Using the access times given in Lemma 1 bounds the running time. For details, see the full version of this paper [28].

4.2 A Dynamic Algorithm for Many Identical Knapsacks

We give a dynamic algorithm with approximation ratio $(1-\varepsilon)$ for Multiple Knapsack, assuming that all knapsacks have the same size S. We assume $m \leq n$ as otherwise, the problem is trivial. We focus on instances where m is large, i.e., $m \geq \frac{16}{\varepsilon^7} \log^2 n$. If $m \leq \frac{16}{\varepsilon^7} \log^2 n$, we use the algorithm due to Theorem 4. In the following, we prove Theorem 6.

▶ **Theorem 6.** *If $m \geq \frac{16}{\varepsilon^7} \log^2 n$, there is a dynamic algorithm for Multiple Knapsack with identical knapsacks with approximation factor $(1-\varepsilon)$ and update time $\big(\frac{\log U}{\varepsilon}\big)^{\mathcal{O}(1)}$, where $U = \max\{Sm, nv_{\max}\}$. Queries for single items and the solution value can be answered in time $\mathcal{O}\big(\frac{\log n}{\varepsilon}\big)^{\mathcal{O}(1)}$ and $\mathcal{O}(1)$, respectively. The solution P can be returned in time $|P|\big(\frac{\log n}{\varepsilon}\big)^{\mathcal{O}(1)}$.*

Our strategy is the following: we partition the input items into large and small items, which are defined w.r.t. the size S of each knapsack. To the large items, we apply oblivious linear grouping, obtaining a polylogarithmic number of item types. We guess the total size of the small items in the optimal solution. Then, we formulate the problem as a configuration linear program (LP) which has a variable for each feasible configuration for a knapsack. A configuration describes how many large items of each type are packed in a knapsack. Also, we ensure that there will be enough space for the small items left. This is similar in spirit to the LPs used in [48,50]; however, we use variables only for the configurations of the big items and we have only a polylogarithmic number of item types, which yields a smaller LP which we can solve faster. We round the obtained fractional solution, using that $m > \frac{16}{\varepsilon^7} \log^2 n$ and that basic feasible solutions to the LP are sparse.

Definitions and Data Structures. We partition the items into two sets, J_B, the *big* items, and J_S, the *small* items, with sizes $s_j \geq \varepsilon S$ and $s_j < \varepsilon S$, respectively. For an optimal solution $\textsc{Opt}$, define $\textsc{Opt}_B := \textsc{Opt} \cap J_B$ and $\textsc{Opt}_S := \textsc{Opt} \cap J_S$.

We maintain three types of auxiliary data structures from Lemma 1: we maintain one such data structure in which we store all items in the order of their arrivals and store the size s_j, the value v_j, and the value class ℓ_j of each item j. For each value class V_ℓ, we maintain a data structure which contains all big items of V_ℓ, ordered non-decreasingly by size. Finally, for the small items (of all value classes together), we maintain a data structure in which they are sorted non-increasingly by density. Upon arrival of a new item j, we insert j into each corresponding data structure.

Algorithm.

1) **Linear grouping of big items:** Guess $\ell_{\max}$, which we define to be the largest index ℓ with $V_\ell \cap \textsc{Opt}_B \neq \emptyset$. Via oblivious linear grouping with $J' = J_B$ and $n' = \min\{\frac{m}{\varepsilon}, n_B\}$ we obtain $\mathcal{T}$; for each item type t, denote by n_t the number of items of this type (the multiplicity of t).
2) **Configurations:** Let $\mathcal{C}$ denote the set of all configurations, i.e., of all multisets of item types whose total size is at most S. For each $c \in \mathcal{C}$, denote by v_c and s_c the total value and size of the item types in c.
3) **Small items:** We guess v_S which we define to be the largest power of $1+\varepsilon$ that is at most $v(\textsc{Opt}_S)$. Let P be the maximal prefix of small items (sorted by non-increasing density) with $v(P) < v_S$. Set $s_S := s(P)$.
4) **Configuration ILP:** We compute an extreme point solution of the LP relaxation of the following configuration ILP with variables y_c for $c \in \mathcal{C}$ for the current guesses $\ell_{\max}$ and v_S (implying s_S). Here, y_c counts how often a certain configuration c is used and n_{tc} denotes the number of items of type t in configuration c.

$$\begin{array}{llcll}
\max & \sum_{c\in\mathcal{C}} y_c v_c & & & \\
\text{subject to} & \sum_{c\in\mathcal{C}} y_c s_c & \leq & \lfloor(1-3\varepsilon)m\rfloor S - s_S & \\
& \sum_{c\in\mathcal{C}} y_c & \leq & \lfloor(1-3\varepsilon)m\rfloor & \\
& \sum_{c\in\mathcal{C}} y_c n_{tc} & \leq & n_t & \text{for all } t\in\mathcal{T} \\
& y_c & \in & \mathbb{Z}_{\geq 0} & \text{for all } c\in\mathcal{C}
\end{array} \tag{P}$$

By the first inequality, the configurations fit into $\lfloor(1-3\varepsilon)m\rfloor$ knapsacks while reserving sufficient space for the small items. The second constraint limits the total number of configurations that are packed. The third inequality ensures that only available items are used.
5) **Obtaining an integral solution:** We round up each variable of the obtained fractional solution, yielding an integral solution $\bar{y}$. As $m \geq \frac{16}{\varepsilon^7}\log^2 n$ and extreme point solutions have only $|\mathcal{T}|+2$ non-zero variables, one can show that $\bar{y}$ still satisfies the relaxed constraints $\sum_{c\in\mathcal{C}} \bar{y}_c s_c \leq \lfloor(1-2\varepsilon)m\rfloor S - s_S$ and $\sum_{c\in\mathcal{C}} \bar{y}_c \leq \lfloor(1-2\varepsilon)m\rfloor$. In case that a constraint $\sum_{c\in\mathcal{C}} \bar{y}_c n_{tc} \leq n_t$ is violated for some type t, we intuitively drop items of type t from some knapsacks until the constraint is satisfied. Let P_B denote the resulting packing.
6) **Packing small items:** Consider the maximal prefix P of small items with $v(P) < v_S$ and let $j^\star$ be the densest small item not in P. Pack $j^\star$ into one of the knapsacks kept empty by P_B. Then, fractionally fill up the $\lfloor(1-2\varepsilon)m\rfloor$ knapsacks used by P_B and place any "cut" item into the $\lceil\varepsilon m\rceil$ additional knapsacks that are still empty.

Analysis. The loss in the objective function value due to linear grouping of big items is bounded by $\frac{(1-\varepsilon)(1-2\varepsilon)}{(1+\varepsilon)^2}$ by Lemma 5. Restricting a solution to its $\lfloor(1-3\varepsilon)m\rfloor$ most valuable knapsacks and guessing the value of small items in these knapsacks only up to a factor of $(1+\varepsilon)$ as done by (P) costs at most a factor of $\frac{1-4\varepsilon}{1+\varepsilon}$ in the objective function value.

For solving the LP-relaxation of the configuration ILP (P), we apply the Ellipsoid method [32] on its dual, using an FPTAS for Knapsack as a separation oracle. For this, we need to handle some technical complications due to the first two constraints of (P), which yield additional variables in the dual, and due to the fact that we can solve the separation problem only up to a factor of $(1+\varepsilon)$ (see [28] for details). Via Gaussian elimination, we transform the obtained fractional solution into a basic feasible solution with the same objective function value. As argued above, since any basic feasible solution has at most $|\mathcal{T}|+2$ non-zero variables, our integral solution $\bar{y}$ uses at most $\lfloor(1-2\varepsilon)m\rfloor$ knapsacks and it has at least the profit of the fractional solution. Given the packing of big items, we pack the small items in a First Fit manner as described in the algorithm.

To bound the running time of our algorithm, we use Lemma 5, show that the relaxation of the configuration ILP can be solved in time $\left(\frac{\log U}{\varepsilon}\right)^{\mathcal{O}(1)}$ with the Ellipsoid method, and use the fact that the algorithm needs at most $\mathcal{O}\left(\frac{\log(nv_{\max})\log v_{\max}}{\varepsilon^2}\right)$ many guesses, see [28] for details.

Queries. In contrast to the previous section, for transforming an implicit solution into an explicit packing, the query operation has to compute the knapsack where a queried item j is packed. We do not explicitly store the packing of any item, but instead we define and update pointers for small items and for each item type, that indicate the knapsacks where the corresponding items are packed. To stay consistent with the precise packing of a particular item between two update operations, we additionally cache query answers.

- **Single Item Query:** For small items, only the prefix of densest items is part of our solution. For big items of a certain type, only the smallest items are packed by the implicit solution. In both cases, we use the corresponding pointer to determine the knapsack.
- **Solution Value Query:** As the algorithm works with rounded values, we use prefix computations on the small items and on any value class of big items to calculate and store the current solution value. Given a query, we return the stored solution value.
- **Query Entire Solution:** We use prefix computations on the small items as well as on the value classes of the big items to determine the packed items. Then, we use the Single Item Query to determine their respective knapsacks.

▶ **Lemma 7.** *The solution determined by the query algorithms is feasible and achieves the claimed total value. The query times of our algorithm are as follows: Single item queries can be answered in time* $\mathcal{O}\left(\log n+\max\left\{\log\frac{\log n}{\varepsilon},\frac{1}{\varepsilon}\right\}\right)$, *solution value queries can be answered in time* $\mathcal{O}(1)$, *and queries of the entire solution* P *can be answered in time* $\mathcal{O}\left(|P|\frac{\log^4 n}{\varepsilon^4}\log\frac{\log n}{\varepsilon}\right)$.

We extend our techniques above to an algorithm for knapsacks of arbitrary sizes, assuming that we have $\left(\frac{\log n}{\varepsilon}\right)^{\Theta(1/\varepsilon)}$ additional knapsacks (of capacity at least as large as the largest original knapsack) as resource augmentation available. The intuition is that these additional knapsacks are sufficient to compensate errors when rounding the LP-relaxation of (P). However, additional care is needed since whether an item is big or small now depends on the knapsack.

▶ **Theorem 8.** *For* $\varepsilon>0$, *there is a dynamic algorithm for Multiple Knapsack that, given* $\left(\frac{\log n}{\varepsilon}\right)^{\Theta(1/\varepsilon)}$ *additional knapsacks as resource augmentation, achieves an approximation factor of* $(1+\varepsilon)$ *with update time* $\left(\frac{1}{\varepsilon}\log n\right)^{\mathcal{O}(1/\varepsilon)}(\log m\log S_{\max}\log v_{\max})^{\mathcal{O}(1)}$. *Item queries are answered in time* $\mathcal{O}\left(\log m+\frac{\log n}{\varepsilon^2}\right)$, *and the solution* P *is output in time* $\mathcal{O}\left(|P|\frac{\log^3 n}{\varepsilon^4}\left(\log m+\frac{\log n}{\varepsilon^2}\right)\right)$.

5 Solving Multiple Knapsack

Having laid the groundwork with the previous two sections, we finally show how to maintain solutions for arbitrary instances of the Multiple Knapsack problem, and give the main result of this paper, summarized in the following theorem. Note that we assume $n\geq m$ as otherwise only the n largest knapsacks are used.

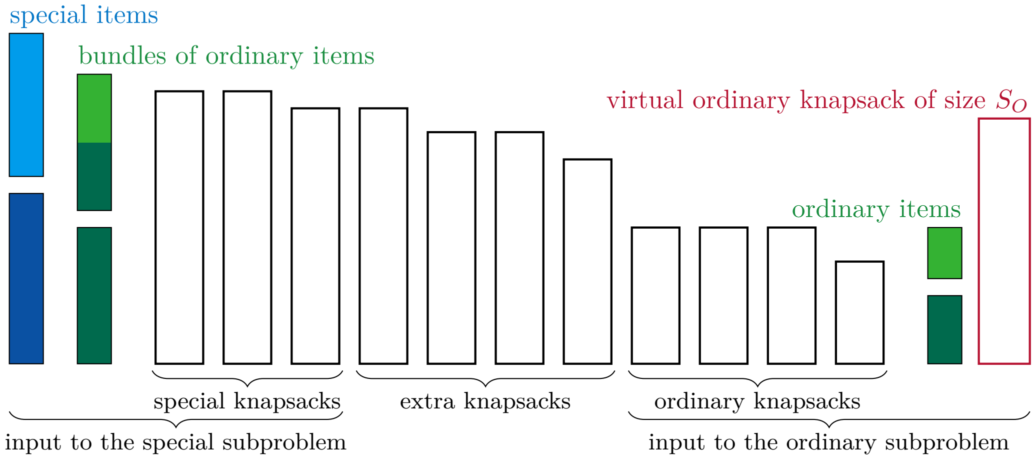

special items
bundles of ordinary items
virtual ordinary knapsack of size S_O
ordinary items
special knapsacks
extra knapsacks
ordinary knapsacks
input to the special subproblem
input to the ordinary subproblem

Figure 1 Input of the special and the ordinary subproblems: Based on the current guess for the extra knapsacks, the knapsacks are partitioned into three groups (special, extra, and ordinary). When an item fits into at least one ordinary knapsack, it is ordinary and special otherwise. The total size of ordinary items placed by Opt in special knapsacks gives the size of the virtual ordinary knapsack. The ordinary items packed into this virtual knapsack are further assigned to bundles of equal size, which are then part of the input to the special subproblem.

▶ **Theorem 9.** *For $\varepsilon > 0$, there is a dynamic, $(1-\varepsilon)$-approximate algorithm for Multiple Knapsack with update time $2^{f(1/\varepsilon)}\left(\frac{1}{\varepsilon}\log n \log v_{\max}\right)^{\mathcal{O}(1/\varepsilon)}(\log S_{\max})^{\mathcal{O}(1)} + \mathcal{O}\left(\frac{1}{\varepsilon}\log \overline{v}\, \log n\right)$, where f is quasi-linear. Item queries are served in time $\mathcal{O}\left(\frac{\log n}{\varepsilon^2}\right)$ and the solution P can be output in time $\mathcal{O}\left(\frac{\log^4 n}{\varepsilon^6}|P|\right)$.*

We obtain this result by partitioning the knapsacks into three sets, special, extra and ordinary knapsacks, and solving the respective subproblems. This has similarities to the approach in [48]; however, there it was sufficient to have only two groups of knapsacks. On a high level, the special knapsacks are the $(\log n)^{\mathcal{O}(1/\varepsilon)}$ largest input knapsacks and, intuitively, we apply the algorithm due to Theorem 4 to them (for a suitably defined set of input items). The extra knapsacks are $(\log n)^{\mathcal{O}(1/\varepsilon)}$ knapsacks that are smaller than the special knapsacks, but larger than the ordinary knapsacks. We ensure that there is a (global) $(1-\varepsilon)$-approximate solution in which they are all empty; see Figure 1. We apply the algorithm due to Theorem 8 to the ordinary and extra knapsacks, where the extra knapsacks form the additional knapsacks used as resource augmentation.

Definitions and Data Structures. Let $L = \left(\frac{\log n}{\varepsilon}\right)^{\Theta(1/\varepsilon)}$. We assume that $m > \left(\frac{1}{\varepsilon}\right)^{4/\varepsilon} \cdot L$, since otherwise we simply apply Theorem 8. Consider $\frac{1}{\varepsilon}$ groups of knapsacks with sizes $\frac{L}{\varepsilon^{3i}}$, for $i = 0, 1, \ldots, \frac{1}{\varepsilon} - 1$, such that the first group, i.e., $i = 0$, consists of the L largest knapsacks, the second, i.e., $i = 1$, of the $\frac{L}{\varepsilon^3}$ next largest, and so on. In Opt, one of these contains items with total value at most $\varepsilon \cdot \text{Opt}$. Let $k \in \{0, 1, \ldots, \frac{1}{\varepsilon} - 1\}$ be the index of such a group and let $L_S := \sum_{i=0}^{k-1} \frac{L}{\varepsilon^{3i}}$. We define the L_S largest input knapsacks to be the *special* knapsacks. The *extra* knapsacks are the $\frac{L}{\varepsilon^{3k}} > \frac{L_S}{\varepsilon^2} + L$ next largest, and the *ordinary* knapsacks the remaining ones.

Call an item *ordinary* if it fits into the largest ordinary knapsack and *special* otherwise. Denote by J_O and J_S the set of ordinary and special items, respectively, and by S_O the total size of ordinary items that Opt places in special knapsacks, rounded down to the next power of $(1+\varepsilon)$. Since we use the algorithms from Theorems 4 and 8 as subroutines, we require the maintenance of the corresponding data structures.

Algorithm.

1) **Oblivious linear grouping:** Compute $\mathcal{O}\big(\frac{\log^2 n}{\varepsilon^4}\big)$ item types as described in Section 4.1. Guess k and determine whether items of a certain type are ordinary or special.
2) **High-value ordinary items:** Place each of the $\frac{L_S}{\varepsilon^2}$ most valuable ordinary items in an empty extra knapsack. On a tie choose the larger item. Denote this set of items by J_E.
3) **Virtual ordinary knapsack:** Guess S_O and add a virtual knapsack with capacity S_O to the ordinary subproblem. In the LP used in the proof of Theorem 8, treat every ordinary item as small item in this knapsack and do not use configurations.
4) **Solve ordinary instance:** Remove temporarily the set J_E from the data structures of the ordinary subproblem. Solve the subproblem with the virtual knapsack as in Theorem 8 and use extra knapsacks for resource augmentation. When rounding up variables, fill the $\mathcal{O}\big(\frac{\log^2 n}{\varepsilon^4}\big)$ rounded items from the virtual knapsack into extra knapsacks.
5) **Create bundles** Consider the items that remain in the virtual ordinary knapsack after rounding. Sort them by type (first value, then size) and cut them to form $\frac{L_S}{\varepsilon}$ bundles B_O of equal size. For each bundle, remember how many items of each type are placed entirely inside it. Place cut items into extra knapsacks. Consider each $B \in B_O$ as an item of size and value equal to the fractional size respectively value of items placed entirely in B.
6) **Solve special instance:** Temporarily insert the bundles in B_O into the data structures used in the special subproblem. Solve this subproblem with the algorithm due to Theorem 4.
7) **Implicit solution:** Among all guesses, keep the solution P_F with the highest value. Store items in J_E and their placement explicitly. Revert the removal of J_E from the ordinary data structures after the next update. For the remaining items, the solutions are given as in the respective subproblem, with the exception of items packed in the virtual ordinary knapsack. The solution of these items is stored implicitly by deciding membership in a bundle on a query.

Queries. We essentially use the same approach as in Theorems 4 and 8 for the ordinary and special subproblem, respectively. However, special care has to be taken with items in the virtual knapsack. In the ordinary subproblem, we assume that items of a certain type which are packed in the virtual knapsack are the first, i.e., smallest, of that type. We can therefore decide in constant time whether or not an item is contained in the virtual knapsack and, if this is the case, fill it into the free space in special knapsacks reserved by bundles. We do this efficiently by using a first fit algorithm on the knapsacks with reserved space. Since items in extra knapsacks are stored explicitly, they can be accessed in constant time. See [28] for details.

Hardness of approximation. It is a natural question whether the update time of our algorithms for Multiple Knapsack can be improved to $\big(\frac{1}{\varepsilon}\log n\big)^{\mathcal{O}(1)}$. We show that this is impossible, unless P=NP.

▶ **Theorem 10.** *Unless* P = NP, *there is no fully dynamic algorithm for* Multiple Knapsack *that maintains a* $(1-\varepsilon)$*-approximate solution in update time polynomial in* $\log n$ *and* $\frac{1}{\varepsilon}$, *for* $m < \frac{1}{3\varepsilon}$.

We give a proof in the full version [28]. We remark that this result can be extended to a larger number of knapsacks by adding an appropriate number of sufficiently small knapsacks, i.e., polynomially many in n.

6 Conclusion

Any dynamic algorithm can be turned into a non-dynamic one by having n items arrive one by one, incurring an additional linear factor in the running time. Hence, lower bounds for the running times of static approximation schemes yield lower bounds for update times of dynamic algorithms. Our running times for the problems with identical capacities are tight in the sense that the algorithms yield a static FPTAS (resp. EPTAS) matching known lower bounds.

Clearly, it would be interesting to generalize our results beyond Multiple Knapsack. A natural generalization is d-dimensional Knapsack, where the items and knapsacks have a size in each of the d dimensions, and a feasible packing of a subset of items must meet the capacity constraint in each dimension. A reduction to one dimension by [24] immediately yields a dynamic $\frac{1-\varepsilon}{d}$-approximation, but designing a dynamic framework with a better guarantee than this remains open. Note that unless W[1] = FPT, 2-dimensional knapsack *does not* admit a dynamic algorithm maintaining a $(1-\varepsilon)$-approximation in worst-case update time $f(\varepsilon)n^{\mathcal{O}(1)}$ [56].

A recent line of research exploits fast techniques for solving convolution problems to speed up knapsack algorithms (exact and approximate); see, e.g., [3, 17, 52, 55, 66]. In fact, it has been shown that Knapsack is computationally equivalent to the $(\min,+)$-convolution problem [22]. It seems worth exploring whether such techniques are useful in the dynamic setting. Here, it is unclear whether the re-computation of a solution in a new iteration can be done in polylogarithmic time. It is also open whether such techniques can be applied for solving Multiple Knapsack, even in the static setting.

We hope to foster further research for other packing, scheduling and, generally, non-graph problems. For bin packing and for makespan minimization on uniformly related machines, we notice that existing PTAS techniques from [53] and [41, 49] combined with rather straightforward data structures can be lifted to a fully dynamic algorithm framework for the respective problems.

References

1 Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In *STOC*, pages 114–125. ACM, 2019.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443. IEEE Computer Society, 2014.

3 Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster knapsack and graph algorithms. In *ICALP*, volume 132 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

4 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 382–405. IEEE, 2019.

5 Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, USA, 1957.

6 Anton Beloglazov and Rajkumar Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *CCGRID*, pages 577–578. IEEE Computer Society, 2010.

7 Anand Bhalgat, Ashish Goel, and Sanjeev Khanna. Improved approximation results for stochastic knapsack problems. In *SODA*, pages 1647–1665. SIAM, 2011.

8 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Design of dynamic algorithms via primal-dual method. In *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 206–218. Springer, 2015.

9 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *SODA*, pages 470–489. SIAM, 2017.

10 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. A new deterministic algorithm for dynamic set cover. In *FOCS*, pages 406–423. IEEE Computer Society, 2019.

11 Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2 + \varepsilon)$-approximate minimum vertex cover in $\mathrm{o}(1/\varepsilon^2)$ amortized update time. In *SODA*, pages 1872–1885. SIAM, 2019.

12 Sujoy Bhore, Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Dynamic geometric independent set. *arXiv preprint*, 2020. `arXiv:2007.08643`.

13 Norman Bobroff, Andrzej Kochut, and Kirk A. Beaty. Dynamic placement of virtual machines for managing SLA violations. In *Integrated Network Management*, pages 119–128. IEEE, 2007.

14 Hans-Joachim Böckenhauer, Dennis Komm, Richard Královic, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theor. Comput. Sci.*, 527:61–72, 2014.

15 Nicolas Boria and Vangelis Th. Paschos. A survey on combinatorial optimization in dynamic environments. *RAIRO - Operations Research*, 45(3):241–294, 2011.

16 Christina Büsing, Arie M. C. A. Koster, and Manuel Kutschka. Recoverable robust knapsacks: the discrete scenario case. *Optim. Lett.*, 5(3):379–392, 2011.

17 Timothy M. Chan. Approximation schemes for 0-1 knapsack. In *SOSA@SODA*, volume 61 of *OASICS*, pages 5:1–5:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

18 Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *FOCS*, pages 370–381. IEEE, 2019.

19 Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, 2005.

20 Spencer Compton, Slobodan Mitrović, and Ronitt Rubinfeld. New partitioning techniques and faster algorithms for approximate interval scheduling. *arXiv preprint*, 2020. `arXiv:2012.15002`.

21 Marek Cygan, Łukasz Jeż, and Jiří Sgall. Online knapsack revisited. *Theory Comput. Syst.*, 58(1):153–190, 2016.

22 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min, +)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019.

23 Khuzaima Daudjee, Shahin Kamali, and Alejandro López-Ortiz. On the online fault-tolerant server consolidation problem. In *SPAA*, pages 12–21. ACM, 2014.

24 Wenceslas Fernandez de la Vega and George S. Lueker. Bin packing can be solved within 1+epsilon in linear time. *Combinatorica*, 1(4):349–355, 1981.

25 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Math. Oper. Res.*, 33(4):945–964, 2008.

26 Camil Demetrescu, David Eppstein, Zvi Galil, and Giuseppe F. Italiano. *Dynamic Graph Algorithms*, page 9. Chapman & Hall/CRC, 2 edition, 2010.

27 Yann Disser, Max Klimm, Nicole Megow, and Sebastian Stiller. Packing a knapsack of unknown capacity. *SIAM J. Discret. Math.*, 31(3):1477–1497, 2017.

28 Franziska Eberle, Nicole Megow, Lukas Nölke, Bertrand Simon, and Andreas Wiese. Fully dynamic algorithms for knapsack problems with polylogarithmic update time. *CoRR*, abs/2007.08415, 2020. `arXiv:2007.08415`.

29 Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-dynamic bin packing with little repacking. In *ICALP*, volume 107 of *LIPIcs*, pages 51:1–51:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

30 George Gens and Eugene Levner. Computational complexity of approximation algorithms for combinatorial problems. In *MFCS*, volume 74 of *Lecture Notes in Computer Science*, pages 292–300. Springer, 1979.

31 George Gens and Eugene Levner. Fast approximation algorithms for knapsack type problems. In *Optimization Techniques*, pages 185–194. Springer, 1980.

32 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

33 Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: Maintaining a constant-competitive steiner tree online. *SIAM J. Comput.*, 45(1):1–28, 2016.

34 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *STOC*, pages 537–550. ACM, 2017.

35 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Randomized algorithms for removable online knapsack problems. In *FAW-AAIM*, volume 7924 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 2013.

36 Xin Han, Yasushi Kawase, Kazuhisa Makino, and He Guo. Online removable knapsack problem under convex function. *Theor. Comput. Sci.*, 540:62–69, 2014.

37 Xin Han and Kazuhisa Makino. Online removable knapsack with limited cuts. *Theor. Comput. Sci.*, 411(44-46):3956–3964, 2010.

38 Monika Henzinger. The state of the art in dynamic graph algorithms. In *SOFSEM*, volume 10706 of *Lecture Notes in Computer Science*, pages 40–44. Springer, 2018.

39 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic Approximate Maximum Independent Set of Intervals, Hypercubes and Hyperrectangles. In *SoCG*, volume 164 of *LIPIcs*, pages 51:1–51:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SoCG.2020.51.

40 Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.

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

42 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

43 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975.

44 Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM J. Discret. Math.*, 4(3):369–384, 1991.

45 Zoran Ivkovic and Errol L. Lloyd. Fully dynamic algorithms for bin packing: Being (mostly) myopic helps. *SIAM J. Comput.*, 28(2):574–611, 1998.

46 Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 293–305. Springer, 2002.

47 Kazuo Iwama and Guochuan Zhang. Online knapsack with resource augmentation. *Inf. Process. Lett.*, 110(22):1016–1020, 2010.

48 Klaus Jansen. Parameterized approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 39(4):1392–1412, 2009.

49 Klaus Jansen. An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables. *SIAM J. Discrete Math.*, 24(2):457–485, 2010.

50 Klaus Jansen. A fast approximation scheme for the multiple knapsack problem. In *SOFSEM*, volume 7147 of *Lecture Notes in Computer Science*, pages 313–324. Springer, 2012.

51 Klaus Jansen and Kim-Manuel Klein. A robust AFPTAS for online bin packing with polynomial migration. *SIAM J. Discret. Math.*, 33(4):2062–2091, 2019.

52 Ce Jin. An improved FPTAS for 0-1 knapsack. In *ICALP*, volume 132 of *LIPIcs*, pages 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

53 Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *FOCS*, pages 312–320. IEEE Computer Society, 1982.

54 Hans Kellerer. A polynomial time approximation scheme for the multiple knapsack problem. In *RANDOM-APPROX*, volume 1671 of *Lecture Notes in Computer Science*, pages 51–62. Springer, 1999.

55 Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an FPTAS for the knapsack problem. *J. Comb. Optim.*, 8(1):5–11, 2004.

56 Ariel Kulik and Hadas Shachnai. There is no EPTAS for two-dimensional knapsack. *Inf. Process. Lett.*, 110(16):707–710, 2010.

57 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *ICALP*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

58 Eugene L. Lawler. Fast approximation algorithms for knapsack problems. *Math. Oper. Res.*, 4(4):339–356, 1979.

59 Yusen Li, Xueyan Tang, and Wentong Cai. On dynamic bin packing for resource allocation in the cloud. In *SPAA*, pages 2–11. ACM, 2014.

60 Will Ma. Improvements and generalizations of stochastic knapsack and markovian bandits approximation algorithms. *Math. Oper. Res.*, 43(3):789–812, 2018.

61 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Math. Program.*, 68:73–104, 1995.

62 Nicole Megow and Julián Mestre. Instance-sensitive robustness guarantees for sequencing with unknown packing and covering constraints. In *ITCS*, pages 495–504. ACM, 2013.

63 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. *SIAM J. Comput.*, 45(3):859–880, 2016.

64 Morteza Monemizadeh. Dynamic maximal independent set. *arXiv preprint*, 2019. `arXiv:1906.09595`.

65 Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. A subquadratic approximation scheme for partition. In *SODA*, pages 70–88. SIAM, 2019.

66 Adam Polak, Lars Rohwedder, and Karol Wegrzycki. Knapsack and subset sum with small items. In *ICALP*, volume 198 of *LIPIcs*, pages 106:1–106:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

67 Donguk Rhee. Faster fully polynomial approximation schemes for knapsack problems. Master's thesis, Massachusetts Institute of Technology, 2015.

68 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, 2009.

69 Martin Skutella and José Verschae. Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. *Math. Oper. Res.*, 41(3):991–1021, 2016.

70 Gang Yu. On the max-min 0-1 knapsack problem with robust optimization applications. *Oper. Res.*, 44(2):407–415, 1996.

Largest Similar Copies of Convex Polygons in Polygonal Domains

Taekang Eom
Department of Computer Science and Engineering,
Pohang University of Science and Technology, South Korea

Seungjun Lee
Department of Computer Science and Engineering,
Pohang University of Science and Technology, South Korea

Hee-Kap Ahn
Department of Computer Science and Engineering, Graduate School of Artificial Intelligence,
Pohang University of Science and Technology, South Korea

Abstract

Given a convex polygon with k vertices and a polygonal domain consisting of polygonal obstacles with n vertices in total in the plane, we study the optimization problem of finding a largest similar copy of the polygon that can be placed in the polygonal domain without intersecting the obstacles. We present an upper bound $O(k^2n^2\lambda_4(k))$ on the number of combinatorial changes occurred to the underlying structure during the rotation of the polygon, together with an $O(k^2n^2\lambda_4(k)\log n)$-time deterministic algorithm for the problem. This improves upon the previously best known results by Chew and Kedem [SoCG89, CGTA93] and Sharir and Toledo [SoCG91, CGTA94] on the problem in more than 27 years. Our result also improves the time complexity of the high-clearance motion planning algorithm by Chew and Kedem.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Polygon placement, Largest similar copy, Polygonal domain

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.19

Related Version *Full Version*: https://arxiv.org/abs/2012.06978

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)).

1 Introduction

Finding a largest object of a certain shape that can be placed in a polygonal environment has been considered as a fundamental problem in computational geometry. This kind of optimization problems arise in various applications, including the metal industry where we want to find a largest similar pattern containing no faults in a piece of material. There is also a correspondence to motion planning problems [12, 13, 17] and shape matching [10].

In the *polygon placement problem*, we are given a container and a fixed shape, and want to find a largest object of the shape that can be inscribed in the container. There are various assumptions on the object to be placed, the motions allowed, and the environment the object is placed within. In many cases, the container is a convex or simple polygon, possibly with holes. Typical shapes are squares, triangles with/without fixed interior angles, and rectangles with/without fixed aspect ratios. For the motions, we may allow translation or both translation and rotation, together with scaling. When scaling is not allowed, the

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 19; pp. 19:1–19:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problem is to find a copy of a given object under translation or rigid motion that can be inscribed in a container [7, 5]. When both translation and scaling are allowed, the objective becomes to find a largest homothetic copy of a given object that can be inscribed in a container [11, 15]. When rotation is allowed, together with translation and scaling, the problem is to find a largest similar copy of a given object that can be inscribed in a container and it becomes more involved; it may require to capture every change occurring to the underlying structure during the rotation of the polygon, and therefore the complexity of the algorithms may depend on the total number of the changes.

In this paper, we consider the polygon placement problem under translation, rotation, and scaling. We aim to find a largest similar copy of a given convex polygon P with k vertices that can be inscribed in a polygonal domain Q consisting of polygonal obstacles with n vertices in total. This problem has been considered fairly well for many years. See Chapter 50 of the Handbook of Discrete and Computational Geometry [12].

The earliest result was perhaps the SoCG'89 paper by Chew and Kedem [8]. They considered the problem and gave an incremental technique for handling all combinatorial changes to the Delaunay triangulation of the polygonal domain Q under the distance function induced by the input polygon P during the rotation of P. They gave an upper bound $O(k^4 n\lambda_4(kn))$ on the number of combinatorial changes, together with a deterministic $O(k^4 n\lambda_4(kn)\log n)$-time algorithm, where $\lambda_s(n)$ is the length of the longest Davenport–Schinzel sequence of order s including n distinct symbols. A few years later, the bound was improved to $O(k^4 n\lambda_3(n))$ by the same authors, and thus the running time of the algorithm became $O(k^4 n\lambda_3(n)\log n)$ [9].

Toledo [21], and Sharir and Toledo [19] studied this problem (they called this problem *the extremal polygon containment problem*) and applied the motion-planning algorithm [13] to solve this problem. They gave an algorithm with running time $O(k^2 n\lambda_4(kn)\log^3(kn)\log\log(kn))$ that uses the parametric search technique of Megiddo [16].

For these two running times, $O(k^4 n\lambda_3(n)\log n)$ and $O(k^2 n\lambda_4(kn)\log^3(kn)\log\log(kn))$, the latter one is asymptotically smaller for large k ($k > n$) while the former one is asymptotically smaller for small k.

There is a randomized algorithm by Agarwal et al. [2] that finds a largest similar copy in $O(kn\lambda_6(kn)\log^3(kn)\log^2 n)$ expected time using the parametric search technique of Megiddo [16]. Agarwal et al. [1] also considered a special case of the problem for finding a largest similar copy of a given convex k-gon contained in a convex n-gon and gave an $O(kn^2\log n)$-time algorithm. Very recently, there were results on two variants. Given a set of n points in the plane, Bae and Yoon [6] gave an $O(n^2\log n)$-time algorithm for finding a largest square that contains no input point in its interior, but contains input points on every side or on three sides. Lee et al. [14] gave an algorithm for finding a largest triangle with fixed interior angles in a simple polygon with n vertices in $O(n^2\log n)$ time.

However, no improvement to the worst-case time bounds by Chew and Kedem, and Sharir and Toledo has been known for the polygon placement problem.

New result. We present an upper bound $O(k^2 n^2\lambda_4(k))$ on the combinatorial changes, which directly improves the worst-case time bound for the algorithm to $O(k^2 n^2\lambda_4(k)\log n)$. This improves upon the previously best known results by Chew and Kedem [9] and Sharir and Toledo [19] in more than 27 years.

Compared to the combinatorial bound $O(k^4 n\lambda_3(n))$ by Chew and Kedem, our bound is $o(k^3 n^2\log^* k)$ while their bound is $O(k^4 n^2\alpha(n))$, because $\lambda_4(k) = o(k\log^* k)$ [20] and $\lambda_3(n) = \Theta(n\alpha(n))$. Therefore, our algorithm outperforms theirs for both k and n. Compared to the time bound $O(k^2 n\lambda_4(kn)\log^3(kn)\log\log(kn))$ by Sharir and Toledo [19], our running time outperforms theirs for both k and n without resorting to parametric search, because $n\lambda_4(k)\log n = o(\lambda_4(kn)\log^3(kn)\log\log(kn))$. Thus our result improves upon the best deterministic result for the problem introduced in Chapter 50 of the Handbook of Discrete and Computational Geometry [12].

Compared to the randomized algorithm using parametric search by Agarwal et al. [2], the worst-case running time of our algorithm, without resorting to parametric search, is asymptotically smaller than their expected running time when $k = O(\log^4 n)$ while it is unclear how to compare the two worst-case running time and the expected running time for larger k.

There is some correspondence between the combinatorial complexity and motion planning problems [9, 12]. In the *high-clearance motion planning,* the goal is to find the path of a convex polygonal robot P contained in a polygonal domain Q from an initial position to a final position while remaining "as far as possible" from the boundaries of Q throughout translations and rotations of P. Chew and Kedem [9] gave an $O(k^4 n\lambda_3(n)\log n)$-time algorithm for high-clearance motion planning, where k and n are the numbers of vertices of P and Q, respectively. Since the running time is dominated by the number of combinatorial changes, our result improves the running time to $O(k^2 n^2\lambda_4(k)\log n)$.

Our result provides some insight for improving the time bounds for some other combinatorial problems with moving objects, for instance the Voronoi diagrams and Delaunay triangulations for moving points under various convex distances [3, 18].

Our improvement may seem marginal compared to previously best ones. However, this is the first and only result that pushes the (worst-case) complexity barrier for the last 27 years. We conjecture that the tight bound is $\Theta(k^2 n^2)$, though we do not have a proof yet. Our result has factor $O(\lambda_4(k))$ to the conjecture, and it could be a stepping stone to closing the gap.

Overview of techniques. We achieve the improved upper bound by carefully analyzing the combinatorial changes in the edge Delaunay triangulation of Q (to be defined later, shortly eDT) while rotating P, and by reducing the candidate size to consider for the changes. Our strategy follows the approach of Chew and Kedem [9], which consists of two parts: Counting the combinatorial changes in eDT for a constant k, and then counting the combinatorial changes with respect to k.

1. In the first part, we analyze the combinatorial changes for a fixed k. We consider a family of functions defined for each vertex and edge of P, and compute their lower envelope. Since there are $O(k)$ vertices and edges of P, we compute $O(k)$ lower envelopes. We show that the complexity of each lower envelope is $O(n)$. Then we compute the breakpoints on the lower envelope of the lower envelopes. To bound the number of combinatorial changes in eDT, we consider a placement of a scaled copy of P such that a vertex of Q and a vertex of P are in contact, which we call a *hinge.* We show that the number of breakpoints on the lower envelope defined for each hinge is $O(n)$. Since there are $O(n)$ hinges for a constant k, the number of combinatorial changes in eDT for θ increasing from 0 to 2π is $O(n^2)$.
2. In the second part, we analyze the combinatorial changes to eDT with respect to k. A combinatorial change to eDT corresponds to a quadruplet of pairs, each pair consisting of an element of Q and an element of P touching each other in some placement of a scaled copy of P simultaneously. To count the quadruplets inducing combinatorial changes to eDT, we consider the triplets of such pairs and define a function for each triplet implying

the size of the scaled copy of P defined by the triplet, satisfying the followings: For the lower envelope L of the functions, a combinatorial change corresponds to an intersection of two such functions appearing in L. That is, every combinatorial change to eDT occurs at a breakpoint on the lower envelope of the functions. So, the complexity of the lower envelope bounds the number of combinatorial changes that occur during the rotation of P. We reduce the complexity bound on the lower envelope by classifying the combination of pairs for the quadruplets.

While this high-level strategy may appear similar to the previous one [9], there are a few major differences and difficulties in improving the bound. In the first part, we improve upon the previous bound $O(n\lambda_3(n))$ by Chew and Kedem as follows. We partition the family of functions to subfamilies such that the functions in the same subfamily have the same domain length, and therefore the complexity of their lower envelope becomes linear to the number of functions [6, 14]. Thus, the total upper bound is improved to $O(n^2)$.

In the second part, instead of the quadruplets considered by Chew and Kedem, we consider the triplets of pairs only and show that the functions for the triplets, in their lower envelope, give us an upper bound on the number of the combinatorial changes to eDT. These functions must reflect the placement of a scaled copy of P in Q as well as the scaling factor. We define functions satisfying this requirement and show that every combinatorial change to eDT occurs at a breakpoint on the lower envelope of the functions. There are $O(k^3n^2)$ such functions and two functions intersect each other at most four times. Thus, the complexity of the lower envelope of the functions is $O(\lambda_6(k^3n^2))$ as the lower envelope corresponds to a Davenport-Schinzel sequence of order 6. To reduce the bound, we classify the functions into types based on the combinations of pairs defining the functions, and show that any two functions belonging to the same type intersect each other less than four times. By applying the partition method in the first part and the classification on the functions above, we show that the complexity of the lower envelope becomes $O(k^2n^2\lambda_4(k))$.

Due to the limit of space, the proofs of some lemmas and corollaries are given in the full version of the paper.

2 Preliminary

A Davenport–Schinzel sequence is a sequence of symbols in which the frequency of any two symbols appearing in alternation is limited. A sequence of symbols is a Davenport–Schinzel sequence of order s if it has no alternating subsequences of length $s+2$. We use $\lambda_s(n)$ to denote the length of the longest Davenport–Schinzel sequence of order s that includes n distinct symbols.

We use some properties related to Davenport–Schinzel sequences in analyzing algorithms. Let $\mathcal{F} = \{f_1, \ldots, f_n\}$ be a collection of n partially-defined, continuous, one-variable real-valued functions. The points at which two functions intersect each other in their graphs and the endpoints of function graphs are called the *breakpoints*. If any two functions of $\mathcal{F}$ intersect each other in their graphs at most s times, the lower envelope of $\mathcal{F}$ has at most $\lambda_{s+2}(n)$ breakpoints [4]. We introduce some technical lemmas that are used in Section 3.

▶ **Lemma 1** (Lemma 14 of [14]). *Assume any two functions f_i and f_j of $\mathcal{F}$ intersect each other in their graphs at most once and each function f_i has domain D_i of length d. If there is a constant c such that $|\bigcup D_i| = cd$, then the lower envelope of $\mathcal{F}$ has $O(n)$ breakpoints.*

▶ **Lemma 2.** *Let $\mathcal{G} = \{g_1, \ldots, g_m\}$ be a collection of m partially-defined, piecewise continuous, one-variable real-valued functions. Let n be the total number of continuous pieces in the function graphs of $\mathcal{G}$. If any two continuous pieces intersect each other in at most s points, the lower envelope of $\mathcal{G}$ has $O(\frac{n}{m}\lambda_{s+2}(m))$ breakpoints.*

We introduce the *edge Voronoi diagram* and its dual, the *edge Delaunay triangulation* (eDT). The set S of sites consists of the edges (open line segments) and their endpoints in the polygonal domain Q. For a convex polygon P in $\mathbb{R}^2$ containing the origin in its interior, the P-distance from a point p to a point q is $d_P(p,q) = \inf\{\mu \mid q \in p + \mu P\}$. The edge Voronoi diagram is a subdivision of the plane into regions such that the points in the same region have the same nearest site under P-distance. See Figure 1(a).

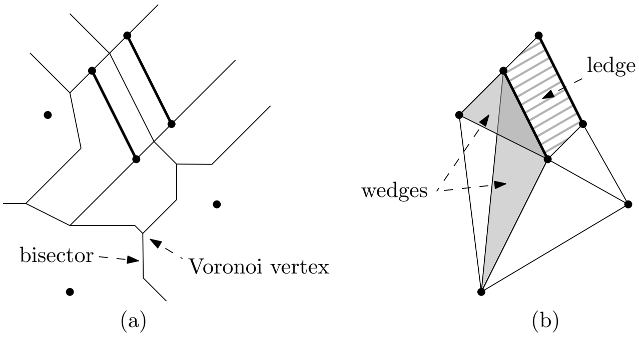


Figure 1 (a) The edge Voronoi diagram of sites, two open line segments (thick) and seven points, under P-distance when P is an axis-aligned square. (b) The edge Delaunay triangulation dual to the edge Voronoi diagram in (a).

The edge Voronoi diagram consists of *Voronoi vertices* and *Voronoi edges (bisectors)*. A point in the plane is a Voronoi vertex if and only if there is an empty circle defined by the P-distance centered at the point and touching three or more sites. The number of Voronoi vertices is linear to the complexity of the sites [15]. A Voronoi edge is a polygonal line that connects two Voronoi vertices. Each point on a Voronoi edge is equidistant from the two sites defining the edge under P-distance. The edge Voronoi diagram can be constructed in $O(kn \log kn)$ time and $O(kn)$ space [15], where k and n are the numbers of vertices in P and Q, respectively.

Just as the standard Delaunay triangulation is the dual of the standard Voronoi diagram, the edge Delaunay triangulation (eDT) is the dual of the edge Voronoi diagram. It has three types of *generalized edges*: *edges*, *wedges*, and *ledges*. An edge connects two point sites, a wedge connects a point site and a segment site, and a ledge connects two segment sites. See Figure 1(b). The edge Delaunay triangulation is a planar graph consisting of point sites, segment sites, generalized edges, and empty triangles. Since a ledge is a trapezoid or a degenerate trapezoid, eDT may not be a triangulation.

The edge Delaunay triangulation can be constructed by first building the edge Voronoi diagram and then tracing the diagram to determine the sites that define each portion of the Voronoi edges and vertices. The type of a generalized edge is determined by the sites defining the corresponding Voronoi edge.

2.1 The Algorithm of Chew and Kedem

We present a sketch of the algorithm by Chew and Kedem. Imagine we rotate P by angle θ in counterclockwise direction, and let P_θ be the rotated copy of P. A homothetic copy $\mathcal{P}_\theta$ of P_θ is said to be *feasible* if $\mathcal{P}_\theta$ is inscribed in Q. For the set S of the sites consisting of the edges and their endpoints in Q, let eDT_θ denote the edge Delaunay triangulation of the sites in S under P_θ-distance. For a face T of eDT_θ, we say $\mathcal{P}_\theta$ is *associated to* T if it touches every site defining T. For $\mathcal{P}_\theta$ associated to T, the set of the elements (vertices or edges) of

$\mathcal{P}_\theta$ touching the sites defining T becomes the *label of* T. See Figure 2(a). For a site s of S, the *label of* s is the set of elements of $\mathcal{P}_\theta$ touching s, for $\mathcal{P}_\theta$ associated to the faces incident to s. See Figure 2(b).

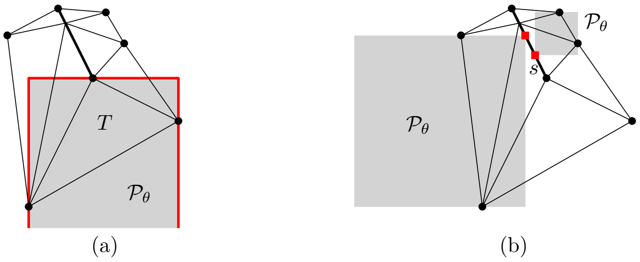


Figure 2 Labels of faces of eDT_θ and edge sites of S for an axis-aligned square P_θ. (a) The label of face T is the set of the edges (red segments) of $\mathcal{P}_\theta$, each containing a site defining T. (b) The label of edge site s is the set of the corners (red points) of $\mathcal{P}_\theta$'s lying on s.

Their algorithm classifies two possible types of changes, an edge change and a label change in eDT_θ while θ increases. In an edge change, a new generalized edge appears or an existing edge disappears. This change occurs when $\mathcal{P}_\theta$ touches four elements of Q, resulting in a flip of the diagonals in the quadrilateral formed by the four edges of eDT_θ. In a label change, the label of a face in eDT_θ changes. This occurs when two or more elements of $\mathcal{P}_\theta$ touch the same site, but the structure of eDT_θ may remain unchanged. Any edge or label change is called a *combinatorial change* to eDT_θ.

Their algorithm maintains a representation for eDT_θ while θ increases. It starts by constructing eDT_θ at $\theta = 0$. An edge change is detected by checking the edges of eDT_θ and a label change is detected by checking the faces of eDT_θ. For each generalized edge in eDT_θ, it determines at which orientation this edge ceases to be valid due to an interaction with its neighbors. For each face in eDT_θ, the algorithm determines at which orientation the label of this face changes. The algorithm maintains the edges and faces of eDT_θ in a priority queue, ordered by the orientations at which they change. At each succeeding stage of the algorithm, it determines which generalized edge is the next one to disappear or which face is the next one to have its label changed as θ increases.

For an edge change, a new edge appears in eDT_θ. Then the algorithm updates eDT_θ and the priority queue information on the new edge and its neighboring edges and faces (an edge change) and for the face and the edges incident to it (a label change). A priority queue can be implemented such that each operation can be done in $O(\log m)$ time, where m is the maximum number of items in the queue. Since there are $O(n)$ edges and faces in the queue at any time, each priority queue operation takes time $O(\log n)$.

For each event of a face T disappearing at θ_e, the algorithm finds the maximal interval $\mathcal{I} = [\theta_s, \theta_e]$ of θ such that T appears in eDT_θ. To find $\mathcal{I}$ in $O(1)$ time, it stores at T the orientation at which it starts to appear in eDT_θ. Then it computes the orientation $\theta^* \in \mathcal{I}$ that maximizes the area of each $\mathcal{P}_\theta$ that touches every site defining T simultaneously. Since $\mathcal{I}$ is maximal, $\mathcal{P}_\theta$ is feasible for every $\theta \in \mathcal{I}$ but not for any $\theta \notin \mathcal{I}$ sufficiently close to $\mathcal{I}$. Thus, the algorithm considers all orientations θ such that $\mathcal{P}_\theta$ is feasible and computes the placement and orientation of the largest similar copy of P. The area function of $\mathcal{P}_\theta$ can be computed in $O(1)$ time and there are $O(1)$ $\mathcal{P}_\theta$ that touch every site defining T simultaneously. Chew and Kedem gave an upper bound $O(k^4 n\lambda_3(n))$ on the number of combinatorial changes, and their algorithm takes $O(k^4 n\lambda_3(n)\log n)$ time [9].

3 The number of changes in eDT_θ

We show that the number of combinatorial changes in eDT_θ during the rotation is $O(k^2n^2\lambda_4(k))$. This directly improves the time bound of the algorithm by Chew and Kedem to $O(k^2n^2\lambda_4(k)\log n)$. We analyze the number of combinatorial changes in eDT_θ for a constant k in Section 3.1, and then analyze the number of combinatorial changes with respect to k in Section 3.2 using the result in Section 3.1.

An ordered pair (A, B) is a *side contact pair* if A is a side of Q and B is a corner of P, and a *corner contact pair* if A is a corner of Q and B is a side of P. A homothetic copy $\mathcal{P}$ of P *satisfies* a contact pair (A, B) if B in $\mathcal{P}$ touches A. See Figure 3 (a). Recall that a homothetic copy $\mathcal{P}$ is said to be *feasible* if $\mathcal{P}$ is inscribed in Q. Note that a homothetic copy $\mathcal{P}$ is not necessarily feasible even if $\mathcal{P}$ satisfies a contact pair.

3.1 The number of changes for fixed k

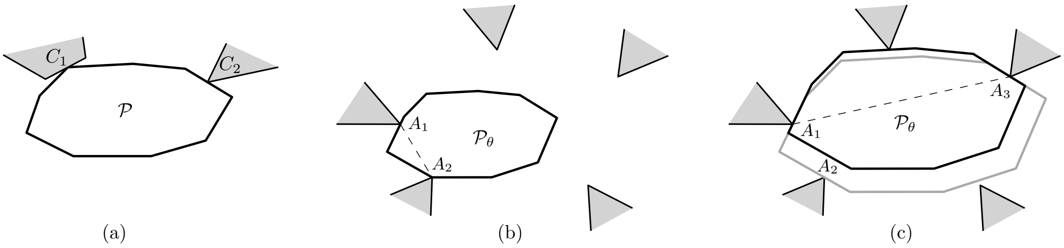


Figure 3 (a) $\mathcal{P}$ satisfies a side contact pair C_1 and a corner contact pair C_2. (b) The segment A_1A_2 is a reported edge with $Q_H = A_2$. (c) The segment A_1A_3 is an unreported edge because no hinge is involved in the segment for a feasible $\mathcal{P}_\theta$.

For a constant k, we improve upon the previously best upper bound $O(n\lambda_3(n))$ by Chew and Kedem [9] to $O(n^2)$. A key idea is to consider the lower envelope of some functions related to the expansion factor, one for each edge and vertex of P, and then to analyze the lower envelope of those lower envelopes. By careful analysis on the complexities of the lower envelopes, we show that the number of combinatorial changes to eDT_θ for θ increasing from 0 to 2π is $O(n^2)$.

To bound the number of changes tight, we classify the generalized edges into two types and count them separately. Chew and Kedem also used this approach. An ordered pair (Q_H, P_H) is a *hinge* if Q_H is a corner of Q and P_H is a corner of P. For a hinge $H = (Q_H, P_H)$ and a contact pair $C = (A, B)$, the generalized edge connecting Q_H and A is a *reported edge* if there is a feasible $\mathcal{P}_\theta$ for some θ satisfying both H and C. An edge of eDT_θ is an *unreported edge* if it is not a reported edge. See Figure 3(b,c). We use the numbers of changes to the reported edges and to the labels in counting the changes to the unreported edges.

Changes to the reported edges and the label changes to point sites. We count the changes to the reported edges and the changes to the labels of point sites in eDT_θ for θ increasing from 0 to 2π. We define the *expansion function* $E_{HC}(\theta)$ for a hinge H and a contact C to be the minimal expansion factor of $\mathcal{P}_\theta$ satisfying H and C. For a hinge $H = (Q_H, P_H)$, let $\mathcal{F}_H$ be the set of all expansion functions satisfying H and another contact pair. An expansion function $E_{HC}(\theta)$ of $\mathcal{F}_H$ for a contact $C = (A, B)$ appears in the lower envelope of $\mathcal{F}_H$ at θ if the generalized edge connecting Q_H and A is a reported edge in eDT_θ.

For a set X of functions, let $\mathsf{B}(X)$ denote the number of breakpoints on the lower envelope of the functions in X. Then the number of changes to the reported edges in eDT_θ which involve H is bounded by $\mathsf{B}(\mathcal{F}_H)$.

Every label change to a point site involves a hinge. See Figure 4(a). An intersection of E_{HC_1} and E_{HC_2} of $\mathcal{F}_H$ for contact pairs C_1 and C_2 appears in the lower envelope of $\mathcal{F}_H$ if a label change to a point site is induced by C_1, C_2 and H. Then the number of label changes to the point sites in eDT_θ which involve H is bounded by $\mathsf{B}(\mathcal{F}_H)$.

▶ **Proposition 3** (Proposition 3 of [9]). *Two expansion functions E_{HC_1} and E_{HC_2} intersect each other in at most one point in their graphs if both C_1 and C_2 are corner contact pairs, or both are side contact pairs. If one is a corner contact pair and the other is a side contact pair, E_{HC_1} and E_{HC_2} intersect each other in at most two points in their graphs.*

Let v_i and e_i denote the vertices and edges of P for $i = 1, \dots, k$. For each $i = 1, \dots, k$, let $\mathcal{C}_{1i} = \{(e, v_i) \mid e \text{ is an edge of } Q\}$ be the set of side contact pairs and let $\mathcal{C}_{2i} = \{(v, e_i) \mid v \text{ is a vertex of } Q\}$ be the set of corner contact pairs. Let $\mathcal{F}_{ji} = \{E_{HC} \mid C \in \mathcal{C}_{ji}\}$ for $j = 1, 2$.

▶ **Lemma 4.** *$\mathsf{B}(\mathcal{F}_{ji}) = O(n)$ for each $j = 1, 2$ and $i = 1, \dots, k$.*

Proof. $\mathsf{B}(\mathcal{F}_{1i}) = O(n)$ for each i since E_{HC_1} and E_{HC_2} intersect each other only at the boundaries of their intervals for $C_1, C_2 \in \mathcal{C}_{1i}$.

Consider now $\mathsf{B}(\mathcal{F}_{2i})$. Two expansion functions E_{HC_1} and E_{HC_2} intersect each other in at most one point for $C_1, C_2 \in \mathcal{C}_{2i}$. Also, E_{HC} has the same length of domain for all $C \in \mathcal{C}_{2i}$. Thus, $\mathsf{B}(\mathcal{F}_{2i}) = O(n)$ by Lemma 1. ◀

From Lemma 2, Proposition 3, and Lemma 4, we achieve an upper bound on $\mathsf{B}(\mathcal{F}_H)$.

▶ **Lemma 5.** *$\mathsf{B}(\mathcal{F}_H) = O(\lambda_3(k)n)$.*

Proof. Let $\mathcal{F}_j = \{f_{j1}, \dots, f_{jk}\}$ for $j = 1, 2$, where f_{ji} is the lower envelope of $\mathcal{F}_{ji}$ for each $i = 1, \dots, k$ and $j = 1, 2$. Let $\mathcal{L}_j$ denote the lower envelope of $\mathcal{F}_j$. Then, the lower envelope of $\mathcal{F}_H$ is the lower envelope of $\mathcal{L}_1$ and $\mathcal{L}_2$. The number of breakpoints on $\mathcal{L}_j$ is $O(\lambda_3(k)n)$ for $j = 1, 2$ by Lemma 2, Proposition 3, and Lemma 4. Then $\mathsf{B}(\mathcal{F}_H) = O(\lambda_3(k)n)$, because two continuous pieces, one from $\mathcal{L}_1$ and one from $\mathcal{L}_2$, intersect each other in at most two points by Proposition 3. ◀

By Lemma 5, the number of changes to the reported edges and the number of label changes to the point sites are $O(k\lambda_3(k)n^2)$.

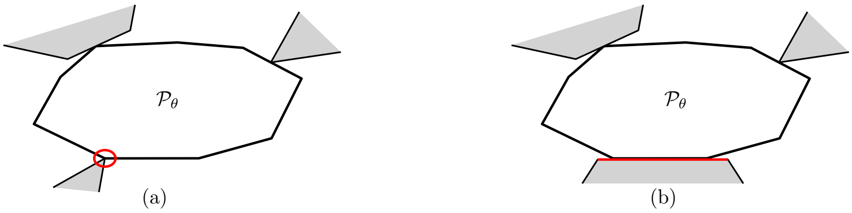


Figure 4 (a) Label change to a point site (hinge). (b) Label change to an edge site.

Label changes to edge sites. We count the changes to the labels of edge sites in eDT_θ for θ increasing from 0 to 2π. Imagine we fix an edge e of Q and an edge g of P. See Figure 4(b). Then, the number of label changes to edge site e with g is $O(n)$ because there are $O(n)$ different $\mathcal{P}_\theta$'s, each associated to a face of eDT_θ while e and g are aligned and touching each other. Thus, the total number of label changes to all edge sites is $O(kn^2)$.

Changes to unreported edges. We count the changes to the unreported edges using the number of changes to the reported edges and to the labels, and Lemma 6.

▶ **Lemma 6** (Lemma 2 of [9]). *Every edge of eDT$_\theta$ is either a reported edge or a diagonal in a convex l-gon, $l \leq 3k$, whose sides are either reported edges or portions of edge sites.*

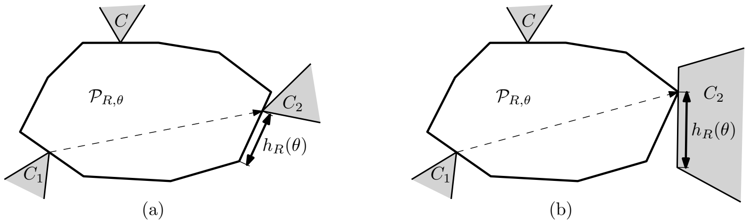
Let G_θ be the graph whose edges are the reported edges in eDT$_\theta$ and portions of edge sites in Lemma 6. We count the changes to the unreported edges which are diagonals in a face of G_θ for an interval of θ with no label change to eDT$_\theta$. Observe that no combinatorial change occurs to G_θ for the interval. Any change to an unreported edge involves four sites lying on a face boundary of G_θ. There are at most four changes for a group of four sites. We describe the details on this bound in Section 4 in the full version. Since each face has at most $3k$ edges by Lemma 6, there are at most $\binom{3k}{4}$ such groups. Thus, $O(k^4)$ changes occur to the unreported edges for the boundary of a face g of G_θ for an interval of θ with no label change to the faces of eDT$_\theta$ intersecting g. Since the number of changes to the reported edges and to the labels is $O(k\lambda_3(k)n^2)$, there are $O(k^5n^2\lambda_3(k))$ combinatorial changes to eDT$_\theta$.

▶ **Theorem 7.** *For a polygonal domain Q of size n and a convex k-gon P, the number of combinatorial changes to eDT$_\theta$ for θ increasing from 0 to 2π is $O(n^2)$ for a constant k.*

3.2 The number of changes with respect to k

We now consider k as a variable and bound the number of changes to eDT$_\theta$. Since each triangular face in eDT$_\theta$ is defined by three elements (edges or vertices) of P, we choose three elements of P and use their convex hull in the counting. Then by Theorem 7, the total number of faces in eDT$_\theta$ for all these convex hulls is $O(k^3n^2)$ for θ increasing from 0 to 2π.

Let $\mathcal{T}$ be the set of all faces of eDT$_\theta$ for the convex hull of three elements B_1, B_2, B_3 of P such that the contact pairs inducing the face have B_1, B_2, and B_3 as their elements. Consider two faces T and T' of eDT$_\theta$ for two distinct orientations θ_1 and θ_2 with $\theta_1 < \theta_2$ that are defined by the same sites. We consider T and T' as distinct faces if there is any change to T or T' in eDT$_\theta$ for θ increasing from θ_1 to θ_2. For a face $T \in \mathcal{T}$, let $C(T)$ be the set of contact pairs which defines T, and let $I(T)$ be the interval of θ at which T appears in eDT$_\theta$.



(a) (b)

Figure 5 $\mathcal{P}_{R,\theta}$ and $h_R(\theta)$ for a restricted contact pair $R = (C, I)$ and θ. Let $h_R(\theta)$ be the distance from the clockwise endpoint (with respect to the dashed ray) of the side element of C_2 to point element of C_2. (a) $h_R(\theta)$ when C_2 is a corner contact. (b) $h_R(\theta)$ when C_2 is a side contact.

For any two fixed contact pairs (C_1, C_2) with $C_i = (A_i, B_i)$ for $i = 1, 2$ such that $A_1 \neq A_2$ and $B_1 \neq B_2$, we count the combinatorial changes involving (C_1, C_2) and other two contact pairs C, C' given in counterclockwise order C_1, C_2, C, and C' along the boundary of P. The combinatorial changes for the cases that $A_1 = A_2$ or $B_1 = B_2$ will be counted for other choices of fixed contact pairs.

We use (C, I) to denote a contact pair C *restricted to* an interval I of θ. For (C_1, C_2), let $\mathcal{R}$ be the set of *restricted contact pairs* (C, I) such that $C(T) = \{C_1, C_2, C\}$ and $I = I(T)$ for a face $T \in \mathcal{T}$, and C_1, C_2, C appear in counterclockwise order along the boundary of P. For a fixed restricted contact pair $R = (C, I) \in \mathcal{R}$ and $\theta \in I$, let $\mathcal{P}_{R,\theta}$ denote the homothet of P_θ which satisfies C_1, C_2, and C. Let $h_R(\theta)$ be the function that denotes the distance from the clockwise endpoint of the side element of C_2 (with respect to the ray from the point element of C_1 to the point element of C_2) to the point element of C_2 with respect to $\mathcal{P}_{R,\theta}$. Observe that h_R is a partially defined continuous function on $R \in \mathcal{R}$. See Figure 5.

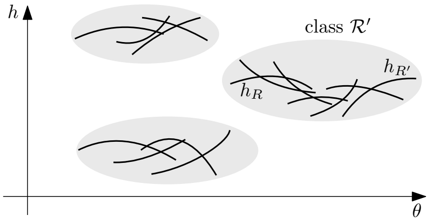


Figure 6 Partitioning $\mathcal{R}$ into classes using the graphs of functions in $\mathcal{F} = \{h_R \mid R \in \mathcal{R}\}$. For two pairs R, R' in class $\mathcal{R}'$, h_R and $h_{R'}$ are connected in the union of the function graphs of $\mathcal{F}$.

Let $\mathcal{F} = \{h_R \mid R \in \mathcal{R}\}$. We partition $\mathcal{R}$ into classes such that two restricted contact pairs R, R' belong to the same class if and only if h_R and $h_{R'}$ are connected in the union of the function graphs of $\mathcal{F}$. Figure 6 illustrates the classes of $\mathcal{R}$.

If a combinatorial change occurs by C_1, C_2, C, and C' at θ, we have $h_R(\theta) = h_{R'}(\theta)$ for two distinct restricted contact pairs $R = (C, I)$ and $R' = (C', I')$. Let $\mathcal{R}'$ be a class of $\mathcal{R}$ and let $\mathcal{F}' = \{h_R \mid R \in \mathcal{R}'\}$. We verify that if $\mathcal{P}_{R,\theta}$ is feasible, then $h_R(\theta)$ appears in the lower envelope or upper envelope of $\mathcal{F}'$.

▶ **Lemma 8.** *Let $R, R', R'' \in \mathcal{R}$ be the restricted contact pairs in the same class. If $h_{R'}(\theta) < h_R(\theta) < h_{R''}(\theta)$, then $\mathcal{P}_{R,\theta}$ is not feasible.*

For a vertex v_i of P, let $\mathcal{R}'_{1i}$ be the set consisting of restricted contact pairs $(C, I) \in \mathcal{R}'$ such that $C = (e, v_i)$ is a side contact pair for some edge $e \in Q$. For an edge e_i of P, let $\mathcal{R}'_{2i}$ be the set consisting of restricted contact pairs $(C, I) \in \mathcal{R}'$ such that $C = (v, e_i)$ is a corner contact pair for some vertex $v \in Q$. Let $|\mathcal{R}'| = m$, and let $|\mathcal{R}'_{1i}| = m_{1i}$ and $|\mathcal{R}'_{2i}| = m_{2i}$ for $i = 1, \ldots, k$. Let $\mathcal{F}'_{ji} = \{h_R \mid R \in \mathcal{R}'_{ji}\}$ for $j = 1, 2$. Recall that for a set X of functions, $\mathsf{B}(X)$ denotes the number of breakpoints on the lower envelope of the functions in X. We have $\mathsf{B}(\mathcal{F}'_{1i}) = O(m_{1i})$ since h_R and $h_{R'}$ intersect each other only at the boundaries of their intervals for $R, R' \in \mathcal{R}'_{1i}$. Let d_i be the number of intersections of the function graphs of $\mathcal{F}'_{2i}$, and let $d = \sum_{i=1}^{k} d_i$. Then $\mathsf{B}(\mathcal{F}'_{2i}) = O(m_{2i} + d_i)$.

▶ **Lemma 9.** $\mathsf{B}(\mathcal{F}'_{1i}) = O(m_{1i})$ *and* $\mathsf{B}(\mathcal{F}'_{2i}) = O(m_{2i} + d_i)$ *for each* $i = 1, \ldots, k$.

Observe that each intersection of the function graphs of $\mathcal{F}'_{2i}$ corresponds to a combinatorial change to eDT for the convex hull of B_1, B_2, and e_i. See Figure 7(a). By Lemma 8, every combinatorial change appears in the lower envelope or upper envelope of $\mathcal{F}'$. Here, we describe the case for the lower envelope of $\mathcal{F}'$. We count the breakpoints of certain types on the lower envelope of $\mathcal{F}'$. We use (a, b)-*change* to denote a combinatorial change induced by a side contact pairs and b corner contact pairs.

Two side contact pairs. We count only $(4, 0)$-changes in this case. We count $(3, 1)$ and $(2, 2)$-changes appearing in the lower envelope in other choices of two pairs, one side contact pair and one corner contact pair. See Figure 7(b). For $\mathcal{F}'_1 = \{f_i \mid i = 1, \ldots, k\}$ such that f_i

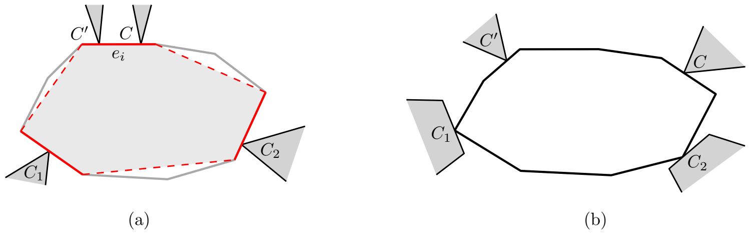


Figure 7 (a) For $R = (C, I)$ and $R' = (C', I')$ in $\mathcal{R}'_{2i}$, and orientation θ with $h_R(\theta) = h_{R'}(\theta)$, there exists a rotated and scaled copy of the convex hull of B_1, B_2, e_i that is feasible and satisfies C_1, C_2, C, and C'. The intersection $h_R(\theta) = h_{R'}(\theta)$ corresponds to a combinatorial change to eDT_θ for the convex hull of B_1, B_2, and e_i. (b) The $(2, 2)$-change induced by C_1, C_2, C and C' is counted when C_2 and C are chosen as the fixed pair.

is the lower envelope of $\mathcal{F}'_{1i}$, $\mathsf{B}(\mathcal{F}'_1) = \sum_{i=1}^{k} O(m_{1i}\lambda_4(k)/k) = O(m\lambda_4(k)/k)$ by Lemmas 2 and 9. We show that any two continuous pieces in $\mathcal{F}'_1$ intersect each other in at most two points in Section 4 of the full version. Since each $(4, 0)$-change corresponds to a breakpoint on the lower envelope of $\mathcal{F}'_1$, the number of $(4, 0)$-changes is $O(m\lambda_4(k)/k)$.

Two corner contact pairs. We count only $(0, 4)$-combinatorial changes in this case. Other changes are counted for other choices of the fixed pairs. For $\mathcal{F}'_2 = \{f_i \mid i = 1, \ldots, k\}$ such that f_i is the lower envelope of $\mathcal{F}'_{2i}$, $\mathsf{B}(\mathcal{F}'_2) = \sum_{i=1}^{k} O((m_{2i} + d_i)\lambda_4(k)/k) = O((m + d)\lambda_4(k)/k)$ by Lemmas 2 and 9. We show that any two continuous pieces in $\mathcal{F}'_2$ intersect each other in at most two points in Section 4 of the full version. Since each $(0, 4)$-change corresponds to a breakpoint of the lower envelope of $\mathcal{F}'_2$, the number of $(0, 4)$-changes is $O((m + d)\lambda_4(k)/k)$.

One side contact pair and one corner contact pair. We count all combinatorial changes other than $(4, 0)$-changes and $(0, 4)$-changes. First, we count the breakpoints on the lower envelope of $\mathcal{F}'_1 = \{h_R \mid R \in \bigcup_{i=1}^{k} \mathcal{R}'_{1i}\}$ and on the lower envelope of $\mathcal{F}'_2 = \{h_R \mid R \in \bigcup_{i=1}^{k} \mathcal{R}'_{2i}\}$, and then count the breakpoints on the lower envelope of $\mathcal{F}'_1 \cup \mathcal{F}'_2$. We show that any two continuous pieces, both from either $\mathcal{F}'_1$ or $\mathcal{F}'_2$, intersect each other in at most two points in Section 4 of the full version. We can compute $\mathsf{B}(\mathcal{F}'_1) = O(m\lambda_4(k)/k)$ and $\mathsf{B}(\mathcal{F}'_2) = O((m+d)\lambda_4(k)/k)$ in the same way as for counting $(4, 0)$-changes and $(0, 4)$-changes, respectively. $\mathsf{B}(\mathcal{F}') = O((m+d)\lambda_4(k)/k)$ since both $\mathsf{B}(\mathcal{F}'_1)$ and $\mathsf{B}(\mathcal{F}'_2)$ are $O((m+d)\lambda_4(k)/k)$, and any two continuous pieces, one from $\mathcal{F}'_1$ and one from $\mathcal{F}'_2$, intersect each other in at most four points. Details are in Section 4 in the full version. Thus, the number the combinatorial changes for the fixed contact pair is $O((m + d)\lambda_4(k)/k)$.

Consider the sum σ of the complexities $|\mathcal{F}'|$ of $\mathcal{F}' = \{h_R \mid R \in \mathcal{R}'\}$ over all classes $\mathcal{R}'$ for a fixed pair. The total sum of σ's for all enumerations of fixed pairs is $O(k^3n^2)$ since $|\mathcal{T}| = O(k^3n^2)$. Similarly, consider the sum ξ of the numbers of intersections (d in the complexities in the previous paragraphs) over all classes for a fixed pair. The total sum of ξ's for all enumerations of fixed pairs is $O(k^3n^2)$ since ξ is bounded by the number of combinatorial changes to eDT_θ for the convex hulls of three elements of P.

▶ **Theorem 10.** *For a polygonal domain Q with n vertices and a convex k-gon P, the number of combinatorial changes to the edge Delaunay triangulation of Q under P_θ-distance for θ increasing from 0 to 2π is $O(k^2n^2\lambda_4(k))$.*

Theorem 10 directly improves upon the algorithm by Chew and Kedem.

▶ **Corollary 11.** *Given a polygonal domain Q with n vertices and a convex k-gon P, we can find a largest similar copy of P inscribed in Q in $O(k^2n^2\lambda_4(k)\log n)$ time using $O(kn)$ space.*

High-clearance motion planning. For a convex polygonal robot P with k vertices and a polygonal domain Q with n vertices in the plane, we want to find a path of P from an initial position to a final position such that the *clearance* of the path exceeds a given value Δ. The clearance of a path of P is the minimum of P-distance from the boundaries of Q throughout translations and rotations of P moving along the path. Chew and Kedem [9] gave an $O(k^4n\lambda_3(n)\log n)$-time algorithm for the high-clearance motion planning. The running time is dominated by the number of combinatorial changes, and our result directly improves the running time.

▶ **Corollary 12.** *Given a convex polygonal robot P with k vertices, a polygonal domain Q with n vertices, initial and final positions of P in the plane, and a clearance Δ, we can find a path of clearance exceeding Δ for P in Q in $O(k^2n^2\lambda_4(k)\log n)$ time using $O(k^2n^2\lambda_4(k))$ space.*

4 The number of critical orientations for four contact pairs

An orientation θ is a *critical orientation* if a combinatorial change to eDT_θ occurs at θ. We consider the critical orientations θ at which $\mathcal{P}_\theta$ has contact with four contact pairs. Recall that an (a,b)-change is a combinatorial change induced by a side contact pairs and b corner contact pairs. We count the number of critical orientations for each (a,b)-change type. The number of critical orientations of $(0,4)$-change is 2, which is shown in Appendix B of [9]. So we count the critical orientations for the other types of combinatorial changes. The following table summarizes the results. Details can be found in the full version.

Types of (a,b)-change	$(4,0)$	$(3,1)$	$(2,2)$	$(1,3)$	$(0,4)$
Number of critical orientations	1	2	4	2	2

References

1 Pankaj K. Agarwal, Nina Amenta, and Micha Sharir. Largest placement of one convex polygon inside another. *Discrete & Computational Geometry*, 19(1):95–104, 1998.

2 Pankaj K. Agarwal, Boris Aronov, and Micha Sharir. Motion planning for a convex polygon in a polygonal environment. *Discrete & Computational Geometry*, 22(2):201–221, 1999.

3 Pankaj K. Agarwal, Haim Kaplan, and Natan Rubin. Kinetic Voronoi diagrams and Delaunay triangulations under polygonal distance functions. *Discrete & Computational Geometry*, 54:871–904, 2015.

4 Mikhail J. Atallah. Dynamic computational geometry. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS 1983)*, pages 92–99. IEEE, 1983.

5 Francis Avnaim and Jean Daniel Boissonnat. Polygon placement under translation and rotation. In Robert Cori and Martin Wirsing, editors, *STACS 88*, pages 322–333, 1988.

6 Sang Won Bae and Sang Duk Yoon. Empty squares in arbitrary orientation among points. In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

7 Bernard Chazelle. The polygon containment problem. In F.P. Preparata, editor, *Advances in Computing Research, Vol I: Computational Geometry*, pages 1–33. JAI Press Inc., 1983.

8 L Paul Chew and Klara Kedem. Placing the largest similar copy of a convex polygon among polygonal obstacles. In *Proceedings of the 5th Annual Symposium on Computational Geometry (SoCG 1989)*, pages 167–173, 1989.

9 L Paul Chew and Klara Kedem. A convex polygon among polygonal obstacles: Placement and high-clearance motion. *Computational Geometry*, 3(2):59–89, 1993.

10 Rudolf Fleischer, Kurt Mehlhorn, Günter Rote, Emo Welzl, and Chee Yap. Simultaneous inner and outer approximation of shapes. *Algorithmica*, 8(1):365, 1992.

11 Steven Fortune. A fast algorithm for polygon containment by translation. In *Proceedings of the 12th International Colloquium on Automata, Languages, and Programming (ICALP 1985)*, pages 189–198, 1985.

12 Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors. *Handbook of Discrete and Computational Geometry*. CRC Press LLC, 3rd edition, 2017.

13 Klara Kedem and Micha Sharir. An efficient motion-planning algorithm for a convex polygonal object in two-dimensional polygonal space. *Discrete & Computational Geometry*, 5:43–75, 1990.

14 Seungjun Lee, Taekang Eom, and Hee-Kap Ahn. Largest triangles in a polygon. *Computational Geometry*, 98:101792, 2021.

15 Daniel Leven and Micha Sharir. Planning a purely translational motion for a convex object in two-dimensional space using generalized Voronoi diagrams. *Discrete & Computational Geometry*, 2:9–31, 1987.

16 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

17 Colm Ó'Dúnlaing and Chee K Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6(1):104–111, 1985.

18 Natan Rubin. On kinetic Delaunay triangulations: A near quadratic bound for unit speed motions. *Journal of the ACM*, 62(3):25:1–25:85, 2015.

19 Micha Sharir and Sivan Toledo. Extremal polygon containment problems. *Computational Geometry*, 4(2):99–118, 1994.

20 Endre Szemerédi. On a problem by Davenport and Schinzel. *Acta Arithmetica*, 25:213–224, 1974.

21 Sivan Toledo. Extremal polygon containment problems. In *Proceedings of the 7th Annual Symposium on Computational Geometry (SoCG 1991)*, pages 176–185, 1991.

A Faster Algorithm for Finding Closest Pairs in Hamming Metric

Andre Esser ✉
Cryptography Research Center, Technology Innovation Institute, Abu Dhabi, UAE

Robert Kübler ✉
Metro AG, Düsseldorf, Germany

Floyd Zweydinger ✉
Ruhr Universität Bochum, Germany

Abstract

We study the Closest Pair Problem in Hamming metric, which asks to find the pair with the smallest Hamming distance in a collection of binary vectors. We give a new randomized algorithm for the problem on uniformly random input outperforming previous approaches whenever the dimension of input points is small compared to the dataset size. For moderate to large dimensions, our algorithm matches the time complexity of the previously best-known locality sensitive hashing based algorithms. Technically our algorithm follows similar design principles as Dubiner (IEEE Trans. Inf. Theory 2010) and May-Ozerov (Eurocrypt 2015). Besides improving the time complexity in the aforementioned areas, we significantly simplify the analysis of these previous works. We give a modular analysis, which allows us to investigate the performance of the algorithm also on non-uniform input distributions. Furthermore, we give a proof of concept implementation of our algorithm which performs well in comparison to a quadratic search baseline. This is the first step towards answering an open question raised by May and Ozerov regarding the practicability of algorithms following these design principles.

2012 ACM Subject Classification Theory of computation → Nearest neighbor algorithms

Keywords and phrases closest pair problem, LSH, nearest neighbor

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.20

Related Version *Full Version*: `https://arxiv.org/abs/2102.02597`

1 Introduction

Finding closest pairs in a given dataset of binary points is a fundamental problem in theoretical computer sciences with numerous applications in data science, machine learning, computer vision, cryptography, and many others.

Image data for example is often represented via compact binary codes to allow for efficient closest pair search in applications like similarity search in images or facial recognition systems [8, 15, 20]. The usage of binary codes also allows decoding the represented data to common codewords. Here, the most efficient algorithms known for decoding such random binary linear codes also heavily benefit from improved algorithms for the Closest Pair Problem [6, 17]. Another common application lies in the field of bioinformatics, where the analysis of genomes involves closest pair search on large datasets to identify the most correlated genetic markers [16, 19].

To be more precise, the Closest Pair Problem asks to find the pair of vectors with the minimal Hamming distance among n given binary vectors. While the general version of this problem does not make any restrictions on the distribution of input points, several settings imply a uniform distribution of dataset elements [6, 16, 17, 19]. Usually, in such settings, there is a planted pair, which attains relative distance $\gamma \in [0, \frac{1}{2}]$, which has to be found. This uniform version is also known as the *light bulb problem* [22]. The problem can be solved in

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 20; pp. 20:1–20:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

time linearly in the dataset size[1] as long as the dimension of vectors is constant [5,14]. As soon as the dimension is non-constant an effect occurs known as *curse of dimensionality*, which lets the problem become much harder.

The most common framework to assess the problem is based on *locality-sensitive hashing* (LSH), whose research was initiated in the pioneering work of Indyk and Motwani [12]. Roughly speaking, a locality-sensitive hash function is more likely to hash points that are close to each other to the same value, rather than points that are far apart. To solve the Closest Pair Problem leveraging an LSH family one chooses a random hash function of the family and computes the hash value of all points in the dataset. In a next step, one computes the pairwise distance only for those pairs, hashing to the same value. This process is then repeated for different hash functions until the closest pair is found. The initial algorithm by Indyk-Motwani achieves a time complexity of $n^{\log_2(\frac{2}{1-\gamma})}$. In general, a time lower bound of $n^{\frac{1}{1-\gamma}}$ is known for LSH based algorithms [9,18]. In [9] Dubiner also gives an abstract idea of an algorithm achieving this lower bound.[2] Later May and Ozerov [17] gave the first concrete algorithmic description following similar design principles, also achieving the mentioned lower bound. Additionally, current data-dependent hashing schemes [2], where the hash function depends also on the actual points in the dataset, improve on the initial idea by Indyk-Motwani and also match the time lower bound of [9,18].

In the uniform setting Valiant [21] was able to circumvent the lower bound by leveraging fast matrix multiplication and hence breaking out of the LSH framework to give an algorithm that runs in time $n^{1.63}\text{poly}(d)$. Remarkably, the complexity exponent of Valiant's algorithm does not depend on the relative distance γ at all. Later this bound was improved to $n^{1.58}\text{poly}(d)$ by Karpa et al. [13] and simplified in an elegant algorithm by Alman [1] achieving the same complexity.

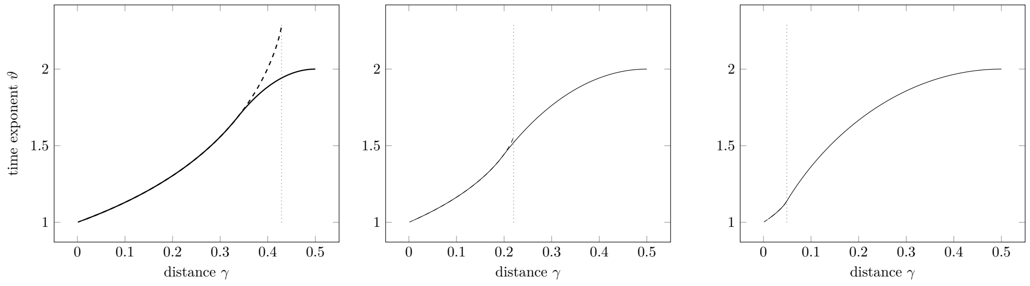
All mentioned algorithms have in common, that they assume a dimension of $d = c(n)\log(n)$, where $c(n)$ is at least a big constant. The explicit size of those constants is usually not stated, instead an asymptotic argument yields their existence. Moreover, the results by [2,9,21] for instance assume even larger dimensions where $\frac{1}{c(n)} = o(1)$. Here, the algorithm by May-Ozerov forms an exception by being applicable for any $c(n) \geq \frac{1}{1-H(\frac{\gamma}{2})}$, where $H(\cdot)$ denotes the binary entropy function. Nevertheless, the mentioned lower bound is only achieved for $c(n)$ approaching infinity. Recently, Xie, Xu and Xu [23] proposed a new algorithm based on decoding the points of the data set according to some random code, exploiting that close vectors are more likely to be decoded to the same word. Their algorithm is also applicable for any $c(n)$ that allows to bound the number of pairs attaining relative distance γ to a constant number with high probability. The authors are able to derandomize their approach and, thus, obtain the fastest known deterministic algorithm for small constants $c(n)$. However, if one also considers probabilistic procedures, their method is inferior to the one by May-Ozerov.

1.1 Our Contribution

We design a randomized algorithm, which achieves the best-known running time for solving the Closest Pair Problem on uniformly random input, when the dimension d is small, which means $c(n)$ being a small constant. Additionally, our algorithm matches the running time of

[1] here we ignore polylogarithmic factors in the dataset size

[2] A precise algorithmic description and a proof of the running time in the case where the vector length is restricted (referred to as *limited amount of data case* in his work) is missing.



(a) Dimension $d = 4\log_2(n)$. **(b)** Dimension $d = 2\log_2(n)$. **(c)** Dimension $d = 1.2\log_2(n)$.

Figure 1 Time complexity exponent ϑ as a function of the relative distance γ of the closest pair for different dimensions. The running time is of the form $n^{\vartheta} \cdot \mathrm{poly}(d)$, where the dashed line represents May-Ozerov's algorithm and the solid line depicts the exponent of our new algorithm. The dotted line gives the maximal γ for which the algorithm by May-Ozerov is still applicable.

the best known LSH algorithms for larger values of $c(n)$ and still matches the time lower bound for LSH based schemes if $\frac{1}{c(n)} = o(1)$. To quantify we give in Figure 1 the achieved runtime exponent for $c(n) \in \{1.2, 2, 4\}$ of our algorithm in comparison to May-Ozerov. As indicated by the graphics, our algorithm can be seen as a natural extension of the May-Ozerov algorithm to higher distances. Moreover, we show that for large distances our algorithm is indeed optimal. Note that apart from the May-Ozerov algorithm *none* of the previously mentioned algorithms is applicable for those choices of $c(n)$. A detailed comparison to the result of May and Ozerov is given right after Theorem 3.

Our improvements over previous work lie in the high density regime, which implies multiple solutions to the Closest Vector Problem. Since the distance alone does not allow to distinguish the planted pair in such cases at least a non-negligible fraction of those pairs needs to be reported, to find the planted pair. The relevance of this setting is mostly given by cryptographic [4, 17] and coding-theoretic [7, 10, 11] applications, precisely the decoding of linear codes. Here, the searched error-vector has known weight and is usually constructed in a tree-wise meet-in-the-middle fashion. Even though the error-vector is usually unique the tree-wise decomposition of the problem introduces multiple solution candidates, such that the lists in the tree can even hold exponentially many pairs with relative distance smaller than γ. However, in such settings, only the elements attaining relative distance γ can possibly sum to the searched error-vector. In the algorithm of Both and May [7], which is the fastest known for decoding random binary linear codes, the authors had to define naive fallback routines for the high density case, for which the May-Ozerov algorithm is not applicable. Here our result allows at least for a unified analysis of the algorithm without the need of fallback routines and in the best case leads to runtime improvements. Also, the generalization of the May-Ozerov nearest neighbor algorithm to $\mathbb{F}_q$ by Hirose [11] suffers similar limitations regarding the high density regime, while also forming the basis for the fastest known decoding algorithm for random linear codes over $\mathbb{F}_q$ [10].

Technically our algorithm follows similar design principles as [9, 17]. At its core, these algorithms group the elements of the given datasets recursively into buckets according to some criterion, which fulfills properties that are similar to those of locality-sensitive hash functions. As the buckets in the recursion are decreasing in size, at the end of the recursion they become small enough to compute the pairwise distance of all contained elements naively.

In contrast to previous works, we exchange the used bucket criteria, which allows us to significantly simplify the algorithms' analysis as well as improve for the mentioned parameter regimes. Also, our approach is applicable for any $c(n)$, thus we are able to remove the restriction $c(n) \geq \frac{1}{1-H(\frac{\gamma}{2})}$.

Following May-Ozerov and Dubiner, we study the bichromatic version of the Closest Pair Problem, which takes as input two datasets rather than one and the goal is to find the closest pair between those given datasets. Obviously, there exists a randomized reduction between the Closest Pair Problem and its bichromatic version, but our algorithm can also be easily adapted to the single dataset case. However, May and Ozerov require the elements within each dataset to be pairwise independent of each other, as a minor contribution we get rid of this restriction, too.

Also, we investigate the algorithms' performance on different input distributions. Therefore we give a modular analysis, which allows for an easy exchange of dataset distribution as well as the choice of bucketing criterion. We also give numerical upper bounds for the algorithm's complexity exponent on some exemplary input distributions. These examples suggest that the chosen criterion is well suited as long as the distance between input elements concentrates around $\frac{d}{2}$ (as in the case of random input lists), while being non-optimal as soon as the expected distance decreases.

We also address an open research question regarding the practical applicability of algorithms following the design of [9,17] raised by May and Ozerov. As their algorithm inherits a huge polynomial overhead in time and space, they left it as an open problem to give a more practical algorithm following a similar design. While our analysis first suggests an equally high overhead, we are able to give an efficient implementation of our algorithm, which requires in addition to the input dataset only constant space. Also, our practical experiments show that most of the overhead of our algorithm is an artifact of the analysis and can be circumvented in practice so that our algorithm performs well compared to a quadratic search baseline.

2 Preliminaries

2.1 Notation

For $a, b \in \mathbb{N}$, $a \leq b$ we denote $[a, b] := \{a, a+1, \ldots, b-1, b\}$. In particular, let $[b] := [1, b]$. For a vector $\mathbf{v} \in \mathbb{F}_2^d$ and $I \in [d]$ let $\mathbf{v}_I$ be the projection of $\mathbf{v}$ onto the coordinates indexed by I, i.e. for $\mathbf{v} = (v_1, v_2, \ldots, v_d)$ and $I = \{i_1, i_2, \ldots, i_k\}$ we have $\mathbf{v}_I = (v_{i_1}, \ldots, v_{i_k}) \in \mathbb{F}_2^k$. We denote the uniform distribution on $\mathbb{F}_2^d$ as $\mathcal{U}\left(\mathbb{F}_2^d\right)$. We define $f(n) = \tilde{\mathcal{O}}\left(g(n)\right) :\Leftrightarrow \exists i \in \mathbb{N}\colon f(n) = \mathcal{O}\left(g(n) \cdot \log^i(g(n))\right)$, i.e. the tilde additionally suppresses polylogarithmic factors in comparison to the standard Landau notation $\mathcal{O}$.

Furthermore, we consider all logarithms having base 2. Define the binary entropy function as $H(x) = -x\log(x) - (1-x)\log(1-x)$ for $x \in (0,1)$, and additionally $H(0) = H(1) := 0$. Using this together with Stirling's formula $n! = \Theta\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)$ we obtain $\binom{n}{\gamma n} = \tilde{\Theta}\left(2^{H(\gamma)n}\right)$. We additionally define $H^{-1}\colon [0,1] \to [0, \frac{1}{2}]$ to be the inverse of the left branch of H.

2.2 Closest Pair Definition

In this work, we consider the Bichromatic Closest Pair Problem in Hamming metric. Here, the inputs are two lists of equal size containing elements drawn uniformly at random from $\mathbb{F}_2^d$ plus a planted pair, whose Hamming distance is γd for some known γ. More formally, we state the problem in the following definition. To allow for easy comparison to the result of May-Ozerov, we follow their notation using the dimension as the primary difficulty parameter. Thus we let the list sizes be $n := 2^{\lambda d}$, which means $\lambda = \frac{1}{c(n)}$, where $d = c(n)\log n$.

▶ **Definition 1** (Bichromatic Closest Pair Problem). *Let $d \in \mathbb{N}$, $\gamma \in \left[0, \frac{1}{2}\right]$ and $\lambda \in (0, 1]$. Let $L_1 = (\mathbf{v}_i)_{i \in [2^{\lambda d}]}, L_2 = (\mathbf{w}_i)_{i \in [2^{\lambda d}]} \in \left(\mathbb{F}_2^d\right)^{2^{\lambda d}}$ be two lists containing elements uniformly drawn at random, together with a distinguished pair $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$ with $\mathrm{wt}(\mathbf{x} + \mathbf{y}) = \gamma d$. We further assume that for each i, j the vectors $\mathbf{v}_i$ and $\mathbf{w}_j$ are pairwise stochastically independent. The* Closest Pair Problem *$\mathcal{CP}_{d,\lambda,\gamma}$ asks to find this* closest pair *$(\mathbf{x}, \mathbf{y})$ given L_1, L_2 and the weight parameter γ. We call $(\mathbf{x}, \mathbf{y})$ the* solution *of the $\mathcal{CP}_{d,\lambda,\gamma}$ problem.*

First, note that $\lambda \leq 1$ is not a real restriction since for $\lambda > 1$ the lists must contain duplicates, which can be safely removed, giving us a problem instance with $\lambda \leq 1$. We also consider the Closest Pair Problem on input lists whose elements are distributed according to some distribution $\mathcal{D}$ different from the uniform one used in Definition 1. To indicate this, we refer to the $\mathcal{CP}_{d,\lambda,\gamma}$ *over distribution* $\mathcal{D}$. Note that in this case, the meaningful upper bound for λ is the entropy of $\mathcal{D}$.

Technically speaking, it is also not necessary to know the value of γ, as the time complexity of appropriate algorithms to solve the $\mathcal{CP}_{d,\lambda,\gamma}$ problem is solely increasing in γ. Thus if γ is unknown, one would apply the algorithm for each $\gamma d = 0, 1, 2, \ldots$ until the solution is found, which results at most in an overhead polynomial in d.

It is well known, that any LSH based algorithm solving the problem of Definition 1 with non-negligible probability needs at least time complexity $|L_1|^{\frac{1}{1-\gamma}} = 2^{\frac{\lambda d}{1-\gamma}}$ [9, 18]. However, this lower bound assumes the promised pair to be uniquely distinguishable from all other pairs in $L_1 \times L_2$. Obviously, if the relation of γ and λ lets us expect more than the promised pair of distance γd in the input lists, an algorithm solving the Closest Pair Problem needs to find all (or at least a non-negligible fraction) of these closest pairs.[3] Hence, if the input lists contain E closest pairs the time complexity of any algorithm solving the problem is lower bounded by

$$\tilde{\Omega}\left(\max(2^{\frac{\lambda d}{1-\gamma}},\ E)\right)$$

Let $(\mathbf{v}, \mathbf{w}) \in L_1 \times L_2 \setminus \{(\mathbf{x}, \mathbf{y})\}$ be arbitrary list elements. If the elements are chosen independently and uniformly at random, as stated in Definition 1 we expect E to be of size

$$\begin{aligned}\mathbb{E}[E] &= (|L_1 \times L_2| - 1) \cdot \Pr\left[\mathrm{wt}(\mathbf{v} + \mathbf{w}) = \gamma d\right] + \underbrace{1}_{\text{from } (\mathbf{x},\mathbf{y})} \\ &= \left(2^{2\lambda d} - 1\right) \cdot \frac{\binom{d}{\gamma d}}{2^d} + 1 \\ &= \tilde{\Theta}\left(2^{(2\lambda + H(\gamma) - 1)d}\right) \ ,\end{aligned}$$

and, thus, the time complexity to solve the $\mathcal{CP}_{d,\lambda,\gamma}$ is lower bounded by

$$T_{\mathrm{opt}} = \tilde{\Omega}\left(\max\left(2^{\frac{\lambda d}{1-\gamma}},\ 2^{(2\lambda + H(\gamma) - 1)d}\right)\right) \ . \qquad (1)$$

3 Our new Algorithm

Our algorithm groups the input elements according to some criterion into several buckets, each one representing a new closest pair instance with smaller list size. We then apply this bucketing procedure recursively until the buckets contain few enough elements to eventually solve the Closest Pair Problem represented by them via a naive quadratic search algorithm, the exhaustive search.

[3] Note that in such a scenario the searched $(\mathbf{x}, \mathbf{y})$ is probably not the pair with the smallest Hamming distance, however, we still refer to elements attaining Hamming distance γd as *closest pairs*.

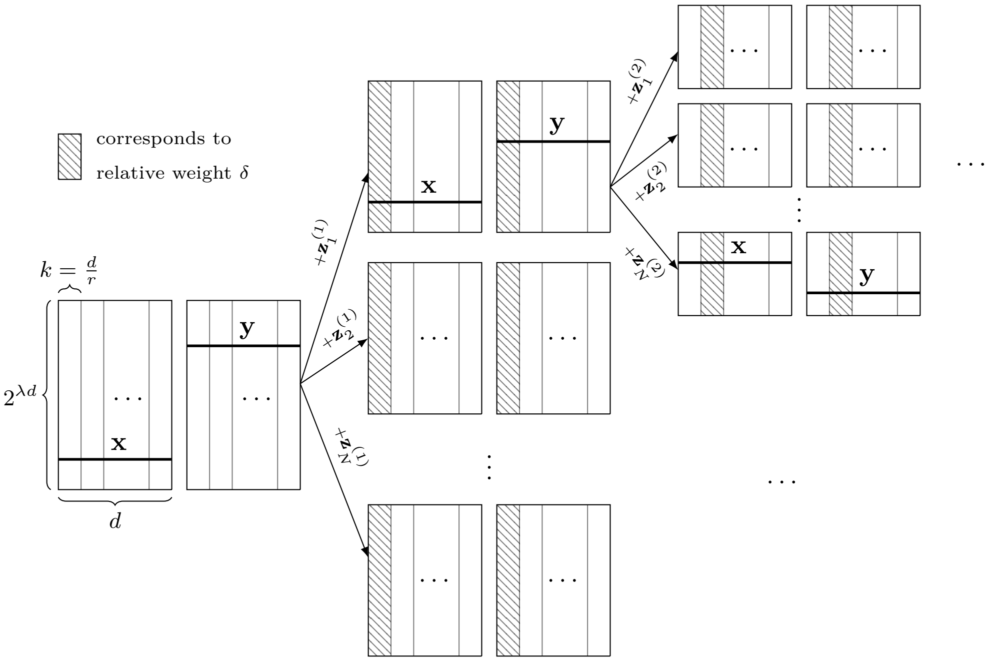


Figure 2 We start off on the left side of the illustration with the two input lists L_1, L_2 containing the closest pair $(\mathbf{x}, \mathbf{y})$. Going right, in each iteration of the algorithm, N different $\mathbf{z}_i^{(j)}$ are randomly chosen and all of the list elements are tested if they fulfill the bucketing criterion. The crosshatched pattern indicates the parts where the bucket criterion is fulfilled, i.e. the list vectors differ from $\mathbf{z}_i^{(j)}$ in δk positions.

As a bucketing criterion, we choose the weight of the vectors after adding a randomly drawn vector $\mathbf{z}$ from $\mathbb{F}_2^d$. Thus, each bucket is represented by a vector $\mathbf{z}$ and only those elements $\mathbf{v}$ are added to the bucket, which satisfy $\mathrm{wt}(\mathbf{v} + \mathbf{z}) = \delta d$, where δ is determined later.

More precisely in each recursive iteration, our algorithm works only on equally large blocks of the input vectors and not on the full d coordinates, i.e. the weight condition is only checked on the current block. This is a technical necessity to obtain independence of vectors in the same bucket on fresh blocks. Let us formally define the notion of blocks.

▶ **Definition 2** (Block). *Let $d, r \in \mathbb{N}$ with $r \mid d$ and $i \in [r]$. Then we denote the i-th* block *of $[d]$ as*

$$B_{i,r}^d := \left[(i-1)\frac{d}{r} + 1,\ i\frac{d}{r}\right] .$$

Note that $[d] = \bigcup_{i \in [r]} B_{i,r}^d$ and $\left|B_{i,r}^d\right| = \frac{d}{r}$ for each $i \in [r]$. Furthermore, the blocks are disjoint. For a leaner notation and since the role of d does not change in the course of this paper, we omit the index d in the following, thus we write $B_{i,r} := B_{i,r}^d$.

Note that May and Ozerov choose the weight of the vectors on random projections as a criterion. In comparison to our variant, their approach involves more parameters and requires extensive re-randomizations of the instance to achieve good success probabilities which together complicates analysis considerably. We cannot rule out the possibility that a different analysis of the May-Ozerov algorithm would allow for an application in the high density regime. However, the complexities of this hypothetical variant are unclear, while our version allows for easy analysis and yields provably optimal complexities in this regime.

In each iteration, we choose the number N of buckets in such a way that with overwhelming probability the closest pair lands in at least one of the buckets. Hence, our algorithm creates a tree with branching factor N with the distinguished pair being contained in one of the leaves. The deeper we get into the tree, the smaller and, hence, the easier the closest pair instances get. An algorithmic description of the whole procedure is given in pseudocode in Algorithm 1. For convenience, a summary of all parameter choices made in line 1 of the algorithm can be found in Equation (8) at the end of Section 3.

Algorithm 1 CLOSEST-PAIR(L_1, L_2, γ).

```
Input: lists $L_1, L_2 \in \left(\mathbb{F}_2^d\right)^{2^{\lambda d}}$, weight parameter $\gamma \in \left[0, \frac{1}{2}\right]$
Output: list $L$ containing the solution $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$ to the $\mathcal{CP}_{d,\lambda,\gamma}$
 1: Set $r, P, N \in \mathbb{N}$, $\delta \in \left[0, \frac{1}{2}\right]$ properly and define $k := \frac{d}{r}$
 2: for $P$ permutations $\pi$ do                                ▷ permutation on the bit positions
 3:     Stack $S := [(\pi(L_1), \pi(L_2), 0)]$
 4:     $L \leftarrow \emptyset$
 5:     while $S$ is not empty do
 6:         $(A, B, i) \leftarrow S$.pop()
 7:         if $i < r$ then
 8:             for $N$ randomly chosen $\mathbf{z} \in \mathbb{F}_2^k$ do
 9:                 $A' \leftarrow (\mathbf{v} \in A \mid \mathrm{wt}((\mathbf{v} + \mathbf{z})_{B_{i+1,r}}) = \delta k)$
10:                 $B' \leftarrow (\mathbf{w} \in B \mid \mathrm{wt}((\mathbf{w} + \mathbf{z})_{B_{i+1,r}}) = \delta k)$
11:                 $S$.push$((A', B', i+1))$
12:         else
13:             for $\mathbf{v} \in A, \mathbf{w} \in B$ do                       ▷ Naive search
14:                 if $\mathrm{wt}(\mathbf{v} + \mathbf{w}) = \gamma d$ then
15:                     $L \leftarrow L \cup \{(\mathbf{v}, \mathbf{w})\}$
16:     return $L$
```

The following theorem gives the time complexity of our algorithm to solve the $\mathcal{CP}_{d,\lambda,\gamma}$.

▶ **Theorem 3.** *Let $\gamma \in \left[0, \frac{1}{2}\right]$ and $\lambda \in [0, 1]$. Then Algorithm 1 solves the $\mathcal{CP}_{d,\lambda,\gamma}$ problem with overwhelming success probability in expected time $2^{\vartheta d(1+o(1))}$, where*

$$\vartheta = \begin{cases} (1-\gamma)\left(1 - H\left(\frac{\delta^\star - \frac{\gamma}{2}}{1-\gamma}\right)\right) & \text{for } \gamma \leq \gamma^\star \\ 2\lambda + H(\gamma) - 1 & \text{for } \gamma > \gamma^\star \end{cases},$$

with $\delta^\star := H^{-1}(1-\lambda)$ and $\gamma^\star := 2\delta^\star(1-\delta^\star)$.

Note that the case distinction marks the transition to the high density regime. Precisely, the transition happens when the amount of closest pairs becomes larger than the running time in the first case. In this first case, where $\gamma \leq \gamma^\star$ our algorithm exactly matches the running time of the May-Ozerov algorithm, which itself is shown to match the lower bound for LSH based approaches whenever λ approaches zero [17] (see also Lemma 8). In the second case, where $\gamma > \gamma^\star$ the running time of our algorithm becomes linear in the number of closest pairs, hence it matches the lower bound from Equation (1), while the running time of May-Ozerov stays as in the first case. Our algorithm hence optimally extends the May-Ozerov algorithm to the high density regime.

We establish the proof of Theorem 3 in a series of lemmata and theorems. Note that any bucketing algorithm heavily depends on two probabilities specific to the chosen bucketing criterion. First, the probability that any element falls into a bucket, which we call p in the remainder of this work. This probability is mainly responsible for the lists' sizes throughout the algorithm. The second relevant probability, which we call q describes the event of both, $\mathbf{x}$ and $\mathbf{y}$, falling into the same bucket, where $(\mathbf{x}, \mathbf{y})$ is the solution to the $\mathcal{CP}_{d,\lambda,\gamma}$ problem. This is the probability of $(\mathbf{x}, \mathbf{y})$ *surviving* one iteration meaning that q determines the success probability of the algorithm. In summary, for our choice of bucketing criteria, we get

$$\begin{aligned} p &:= \Pr_{\mathbf{z}}\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}}) = \delta k\right] \text{ for any } \mathbf{v} \in \mathbb{F}_2^k \text{ and} \\ q &:= \Pr_{\mathbf{z}}\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}}) = \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}}) = \delta k\right] \quad , \end{aligned} \tag{2}$$

where $k = \frac{d}{r}$ is the block width. If we assume that the γd differing coordinates of $\mathbf{x}$ and $\mathbf{y}$ distribute evenly into the r blocks, i.e. $\mathrm{wt}((\mathbf{x}+\mathbf{y})_{B_{i,r}}) = \gamma k$ for each i, these probabilities are independent of i for δk fixed. This property is ensured for at least one of the P permutations in Algorithm 1 with overwhelming probability, as we will see in the proof of Theorem 4.

We determine the exact form of p and q later. First, we are going to prove the following statement about the expected running time of Algorithm 1 in dependence on both probabilities.

▶ **Theorem 4.** *Let p and q be as defined in Equation (2), $\gamma \in \left[0, \frac{1}{2}\right]$, $\lambda \in [0,1]$ and $r = \frac{\lambda d}{\log^2 d}$. Then Algorithm 1 solves the $\mathcal{CP}_{d,\lambda,\gamma}$ problem in expected time*

$$\max\left(q^{-r}, \frac{2^{\lambda d} \cdot p^{r-1}}{q^r}, \frac{\left(2^{\lambda d} \cdot p^r\right)^2}{q^r}\right)^{1+o(1)}$$

with a success probability overwhelming in d.

Proof. First, we are going to prove the statement about the time complexity.

The algorithm maintains a stack, containing list pairs together with an associated counter. In every iteration of the loop in line 5, one element is removed from the stack and if the counter i associated with this element is smaller than r, N additional elements $(A', B', i+1)$ are pushed to the stack in line 11. Let us consider the elements on the stack as nodes in a tree of depth r, where all elements with associated counter i are siblings on level i of the tree. Also, depict the elements pushed to the stack in line 11 as child nodes of the currently processed node (A, B, i). Then the total number of elements with associated counter i pushed to the stack is bounded by the number of nodes on level i in a tree with branching factor N, which is N^i.

Next, let us determine the lists' sizes on level i of that tree. Therefore, let the expected size of lists on level i be $\mathcal{L}_i$. As these lists are constructed from the lists of the previous level by testing the weight condition in line 9 and 10, it holds that

$$\mathcal{L}_i = \mathcal{L}_{i-1} \cdot \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}})) = \delta k\right] := \mathcal{L}_{i-1} \cdot p \quad ,$$

where $i > 0$ and by construction $\mathcal{L}_0 = |L_1|$. By substitution we get

$$\mathcal{L}_i = |L_1| \cdot p^i \quad , \text{ for } i = 0, \dots, r.$$

Now, we are able to compute the time needed to create the nodes on level i of the tree. Observe that for the creation of a level-i node we need to linearly scan through the larger lists of a node on level $i-1$ to check the weight conditions. Thus, to construct all N^i nodes of level i we need a total time of

$$T_i = \tilde{\mathcal{O}}\left(\mathcal{L}_{i-1} \cdot N^i\right) = \tilde{\mathcal{O}}\left(|L_1| \cdot p^{i-1} \cdot N^i\right) \quad ,$$

for each $0 < i \leq r$. Eventually, the list pairs on level r are matched by a naive search with quadratic runtime resulting in

$$T_{r+1} = \tilde{\mathcal{O}}\left(N^r \cdot \mathbb{E}[|A_r| \cdot |B_r|]\right) \ ,$$

where A_r, B_r describe the lists of a level-r node.

The expected value of the product, now, depends on the chosen input distribution. We next argue that for the given input distribution we have

$$\mathbb{E}[|A_r| \cdot |B_r|] = \mathcal{O}\big(\mathbb{E}[|A_r|] \cdot \mathbb{E}[|B_r|]\big) = \mathcal{O}(\mathcal{L}_r^2) \ .$$

To see this, first note that for $\mathbf{v}, \mathbf{w}, \mathbf{z}$ independent and uniform, $\mathbf{v}+\mathbf{z}$ and $\mathbf{w}+\mathbf{z}$ are also independent and uniform. This in turn implies

$$\begin{aligned} &\Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}})) = \delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}})) = \delta k\right] \\ =&\Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}})) = \delta k\right] \cdot \Pr\left[\mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}})) = \delta k\right] \\ =&p^2 \end{aligned}$$

since deterministic functions of independent random variables are still independent. This also works for either $\mathbf{v} = \mathbf{x}$ or $\mathbf{w} = \mathbf{y}$, but not for $(\mathbf{v}, \mathbf{w}) = (\mathbf{x}, \mathbf{y})$. In this case, however, we have $\Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}})) = \delta k, \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}})) = \delta k\right] = q$ by definition. With this insight, we can express $\mathbb{E}[|A_i| \cdot |B_i|]$ in terms of $\mathbb{E}[|A_{i-1}| \cdot |B_{i-1}|]$ for each i via

$$\begin{aligned} \mathbb{E}[|A_i| \cdot |B_i| \mid A_{i-1}, B_{i-1}] = &\sum_{\substack{\mathbf{v} \in A_{i-1},\ \mathbf{w} \in B_{i-1} \\ (\mathbf{v},\mathbf{w}) \neq (\mathbf{x},\mathbf{y})}} \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}}) = \delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}}) = \delta k\right] \\ &+ \Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}})) = \delta k, \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}})) = \delta k\right] \\ &= (|A_{i-1}| \cdot |B_{i-1}| - 1)p^2 + q \\ &\leq |A_{i-1}| \cdot |B_{i-1}| \cdot p^2 + 1 \ , \end{aligned}$$

Applying the Law of total Expectation we obtain

$$\mathbb{E}[|A_i| \cdot |B_i|] = \mathbb{E}[\mathbb{E}[|A_i| \cdot |B_i| \mid A_{i-1}, B_{i-1}]] \leq \mathbb{E}[|A_{i-1}| \cdot |B_{i-1}|] \cdot p^2 + 1 \tag{3}$$

Successive application of Equation (3) yields

$$\mathbb{E}[|A_r| \cdot |B_r|] \leq \mathbb{E}[|L_1| \cdot |L_2|] \cdot p^{2r} + r = 2^{2\lambda d} p^{2r} + r = \mathcal{O}(\mathcal{L}_r^2) \tag{4}$$

Finally, the algorithm is repeated for P different permutations on the bit positions of elements in L_1, L_2. In summary, the expected time complexity to build all list becomes the sum of the T_i multiplied by P, thus, by choosing $N := \frac{d}{q}$ and $P = (d+1)^{r+1}$ we get

$$\begin{aligned} T' = P \cdot \sum_{i=1}^{r+1} T_i &\leq (d+1)^{r+1} \cdot \left(\sum_{i=1}^{r} N^i \cdot |L_1| \cdot p^{i-1} + (|L_1| \cdot p^r)^2 \cdot N^r\right) \\ &= (d+1)^{r+1} \cdot \left(\sum_{i=1}^{r} \frac{|L_1| \cdot d^i}{q} \cdot \left(\frac{p}{q}\right)^{i-1} + \frac{(|L_1| \cdot p^r)^2 \cdot d^r}{q^r}\right) \\ &\leq (d+1)^{2r+1} \cdot \left(\frac{r \cdot |L_1| \cdot p^{r-1}}{q^r} + \frac{(|L_1| \cdot p^r)^2}{q^r}\right) \\ &= \max\left(\frac{2^{\lambda d} \cdot p^{r-1}}{q^r}, \frac{(2^{\lambda d} \cdot p^r)^2}{q^r}\right)^{1+o(1)} \ , \end{aligned}$$

where the inequality follows from the fact that $\frac{p}{q} \geq 1$ since

$$\begin{aligned} q &= \Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}}) = \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}}) = \delta k\right] \\ &\leq \Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}}) = \delta k\right] \\ &= p \ , \end{aligned}$$

and the final equality stems from the fact that $|L_1| = 2^{\lambda d}$ and $r = o(\frac{\lambda d}{\log d})$ as given in the theorem.

Note that T' disregards the fact that no matter how small the lists in the tree become, the algorithm needs to traverse all

$$T'' = \tilde{\mathcal{O}}\left(N^r\right) = \tilde{\mathcal{O}}\left(\left(\frac{d}{q}\right)^r\right)$$

nodes of the tree. Hence, the expected time complexity of the whole algorithm is

$$T = \max(T', T'') \ ,$$

which proves the claim.

Let us now consider the success probability of the algorithm. Therefore, we assume that the chosen permutation distributes the weight on $\mathbf{x}+\mathbf{y}$ such that in every block of length r the weight is equal to $\frac{\gamma d}{r}$, which we describe as a *good* permutation. The probability of a random permutation π distributing the weight in such a way is

$$\Pr\left[\text{good } \pi\right] = \Pr\left[\mathrm{wt}\left(\pi(\mathbf{x}+\mathbf{y})_{B_{i,r}}\right) = \frac{\gamma d}{r}, \text{ for } i = 1, \ldots, r\right] = \frac{\binom{\frac{d}{r}}{\frac{\gamma d}{r}}^r}{\binom{d}{\gamma}} \geq \left(\frac{d}{r}+1\right)^{-r} \ .$$

Thus, the probability of at least one out of $(d+1)^{r+1}$ chosen permutations being good is

$$\begin{aligned} p_1 &:= \Pr\left[\text{at least one good } \pi\right] \\ &= 1 - \left(1 - \Pr\left[\text{good } \pi\right]\right)^{(d+1)^{r+1}} = 1 - \left(1 - \left(\frac{d}{r}+1\right)^{-r}\right)^{(d+1)^{r+1}} \geq 1 - e^{-d} \ . \end{aligned}$$

The algorithm succeeds, whenever there exists a leaf node in the tree, containing the distinguished pair $(\mathbf{x}, \mathbf{y})$. As every node in the tree is constructed based on its parent, it follows that all nodes on the path from the root to that leaf need to contain $(\mathbf{x}, \mathbf{y})$. By definition the probability of $\mathbf{x}$ and $\mathbf{y}$ satisfying the bucket criterion at the same time (thus for the same $\mathbf{z}$) is q and since we condition on a good permutation, q is equal for every considered block. Let us define indicator variables X_j for the first level, where $X_j = 1$ iff the j-th node contains $(\mathbf{x}, \mathbf{y})$. Observe that the X_j for independent choices of $\mathbf{z}$ are independent. Thus, clearly the number of trials until $(\mathbf{x}, \mathbf{y})$ is contained in any node on level one is distributed geometrically with parameter q. Hence, the probability of the solution being contained in at least one node on the first level is

$$\begin{aligned} p_2 &:= \Pr\left[\exists (A, B, 1) \in S : (\mathbf{x}, \mathbf{y}) \in A \times B\right] \\ &= 1 - (1-q)^N = 1 - (1-q)^{\frac{d}{q}} \geq 1 - e^{-d} \ . \end{aligned}$$

Now, imagine the pair being contained in some level-i node. Considering that node, we have with the same probability p_2 again that at least one child contains the solution, and the same argument holds until we reach the leaves. Also, by the independent choices of $\mathbf{z}$ the events remain independent which implies that the probability of $(\mathbf{x}, \mathbf{y})$ being contained in a level-r list is p_2^r. In summary, the success probability is

$$\Pr[\text{success}] = p_1 \cdot p_2^r \geq (1 - e^{-d})^{r+1} \geq 1 - \frac{r+1}{e^d} \geq 1 - \frac{d}{e^d} \ . \qquad ◀$$

The proof of Theorem 4 already shows, how different distributions may affect the complexity of the algorithm by changing the expected value $\mathbb{E}[|A_r| \cdot |B_r|]$. This influence on the algorithms complexity by different input distributions is further investigated in Section 4.

In the next two lemmata, we will determine the exact forms of p and q to conduct the run time analysis.

▶ **Lemma 5.** *Let $k \in \mathbb{N}$, $\delta \in [0, 1]$. If $\mathbf{x} \in \mathbb{F}_2^k$ and $\mathbf{z} \sim \mathcal{U}(\mathbb{F}_2^k)$ then*

$$\Pr_{\mathbf{z}}[\text{wt}(\mathbf{x} + \mathbf{z}) = \delta k] = \binom{k}{\delta k} \left(\frac{1}{2}\right)^k .$$

Proof. Since $\mathbf{z} \sim \mathcal{U}(\mathbb{F}_2^k)$, the probability is

$$\frac{\left|\{\mathbf{z} \in \mathbb{F}_2^k \mid \text{wt}(\mathbf{x} + \mathbf{z}) = \delta k\}\right|}{\left|\mathbb{F}_2^k\right|} .$$

To compute the numerator, note that $\text{wt}(\mathbf{x} + \mathbf{z}) = \delta k$ means that $\mathbf{x}$ and $\mathbf{z}$ differ in δk out of k coordinates, for which there are $\binom{k}{\delta k}$ possibilities. Using $\left|\mathbb{F}_2^k\right| = 2^k$, the lemma follows. ◀

Before we continue, let us make a small definition.

▶ **Definition 6.** *Let $k \in \mathbb{N}$ and $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^k$. Then we define $D(\mathbf{x}, \mathbf{y}) \subseteq [k]$ to be the set of coordinates where $\mathbf{x}$ and $\mathbf{y}$ differ, i.e.*

$$D(\mathbf{x}, \mathbf{y}) := \{i \in [k] \mid \mathbf{x}_i \neq \mathbf{y}_i\}.$$

Furthermore, let $S(\mathbf{x}, \mathbf{y}) := [k] \setminus D(\mathbf{x}, \mathbf{y})$ be the set of coordinates where they are the same.

Now we derive the exact form of the probability q of a pair with difference γk falling into the same bucket.

▶ **Lemma 7.** *Let $k \in \mathbb{N}$, $\delta \in [0, 1]$. If $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^k$ with $\text{wt}(\mathbf{x} + \mathbf{y}) = \gamma k$ and $\mathbf{z} \sim \mathcal{U}(\mathbb{F}_2^k)$. Then*

$$\Pr_{\mathbf{z}}[\text{wt}(\mathbf{x} + \mathbf{z}) = \text{wt}(\mathbf{y} + \mathbf{z}) = \delta k] = \binom{\gamma k}{\frac{1}{2}\gamma k} \binom{(1-\gamma)k}{\left(\delta - \frac{\gamma}{2}\right) k} \left(\frac{1}{2}\right)^k .$$

Proof. Let

$$A := \{\mathbf{z} \in \mathbb{F}_2^k \mid \text{wt}(\mathbf{x} + \mathbf{z}) = \text{wt}(\mathbf{y} + \mathbf{z}) = \delta k\}.$$

In analogy to Lemma 5, the probability we search for is $\frac{|A|}{|\mathbb{F}_2^k|} = |A| \cdot \left(\frac{1}{2}\right)^k$.

In the following, let $\gamma_{\mathbf{x}} := \text{wt}(\mathbf{x} + \mathbf{z})$ and analogously $\gamma_{\mathbf{y}} := \text{wt}(\mathbf{y} + \mathbf{z})$. Now observe that every coordinate z_i of $\mathbf{z}$ with $i \in S(\mathbf{x}, \mathbf{y})$, so belonging to the set of equal coordinates between $\mathbf{x}$ and $\mathbf{y}$, either contributes to both $\gamma_{\mathbf{x}}$ *and* $\gamma_{\mathbf{y}}$ with one or does not affect either one of them. Let us define the amount of the z_i's with $i \in S(\mathbf{x}, \mathbf{y})$ that contribute to the weight as $a := |S(\mathbf{x}, \mathbf{y}) \cap D(\mathbf{x}, \mathbf{z})|$.

Now consider the z_i's with $i \in D(\mathbf{x}, \mathbf{y})$. Clearly, any such z_i contributes *either* to $\gamma_{\mathbf{x}}$ *or* to $\gamma_{\mathbf{y}}$. Thus, let us define the number of those z_i with $i \in D(\mathbf{x}, \mathbf{y})$ that contribute to $\gamma_{\mathbf{x}}$ as $b_{\mathbf{x}} := |D(\mathbf{x}, \mathbf{y}) \cap D(\mathbf{x}, \mathbf{z})|$ and analogously those which contribute to $\gamma_{\mathbf{y}}$ as $b_{\mathbf{y}} := |D(\mathbf{x}, \mathbf{y}) \cap D(\mathbf{y}, \mathbf{z})|$. Obviously we have

$$b_{\mathbf{x}} + b_{\mathbf{y}} = |D(\mathbf{x}, \mathbf{y})| = \gamma k \tag{5}$$

On the other hand we are only interested in those $\mathbf{z}$ for which $\gamma_{\mathbf{x}} = \gamma_{\mathbf{y}} = \delta k$, which yields the two equations

$$\gamma_{\mathbf{x}} = a + b_{\mathbf{x}} = \delta k \tag{6}$$
$$\gamma_{\mathbf{y}} = a + b_{\mathbf{y}} = \delta k \tag{7}$$

All three equations together yield the unique solution

$$b_{\mathbf{x}} = b_{\mathbf{y}} = \frac{\gamma k}{2} \text{ and } a = \left(\delta - \frac{\gamma}{2}\right) k \enspace .$$

This shows the following: If $\mathbf{z} \in A$, it is necessary that $\mathbf{z}$ differs from $\mathbf{x}$ (analogously $\mathbf{y}$) in exactly

- $\frac{\gamma}{2} k$ out of γk coordinates of $D(\mathbf{x}, \mathbf{y})$ and
- $\left(\delta - \frac{\gamma}{2}\right) k$ out of $(1-\gamma)k$ coordinates of $S(\mathbf{x}, \mathbf{y})$.

Thus, because we can freely combine both conditions, in total there are

$$|A| = \binom{\gamma k}{\frac{\gamma}{2} k} \binom{(1-\gamma)k}{\left(\delta - \frac{\gamma}{2}\right) k}$$

different values for $\mathbf{z}$, finishing the proof. ◀

Now we are ready to prove Theorem 3 about the time complexity of Algorithm 1 for solving the $\mathcal{CP}_{d,\lambda,\gamma}$ problem. For convenience, we restate the theorem here.

▶ **Theorem 3.** *Let $\gamma \in \left[0, \frac{1}{2}\right]$ and $\lambda \in [0, 1]$. Then Algorithm 1 solves the $\mathcal{CP}_{d,\lambda,\gamma}$ problem with overwhelming success probability in expected time $2^{\vartheta d(1+o(1))}$, where*

$$\vartheta = \begin{cases} (1-\gamma)\left(1 - H\left(\frac{\delta^\star - \frac{\gamma}{2}}{1-\gamma}\right)\right) & \text{for } \gamma \leq \gamma^\star \\ 2\lambda + H(\gamma) - 1 & \text{for } \gamma > \gamma^\star \enspace , \end{cases}$$

with $\delta^\star := H^{-1}(1-\lambda)$ and $\gamma^\star := 2\delta^\star(1-\delta^\star)$.

Proof. First let us give the exact form of $\log p$ and $\log q$ using Stirling's formula to approximate the binomial coefficients in Lemma 5 and 7. By setting the block width $k = \frac{d}{r}$ we get

$$\log q \ = (1-\gamma)\left(H\left(\frac{\delta - \frac{\gamma}{2}}{1-\gamma}\right) - 1\right)\frac{d}{r}\big(1 + o(1)\big), \quad \log p = \big(H(\delta) - 1\big)\frac{d}{r}\big(1 + o(1)\big) \enspace .$$

Now, let us reconsider the running time given in Theorem 4 as

$$T = \max\Bigg(\underbrace{\frac{1}{q^r}}_{(a)}, \underbrace{\frac{2^{\lambda d} \cdot p^{r-1}}{q^r}}_{(b)}, \underbrace{\frac{\left(2^{\lambda d} \cdot p^r\right)^2}{q^r}}_{(c)}\Bigg)^{1+o(1)} \enspace ,$$

where $r = \frac{\lambda d}{\log^2 d}$.

We now show that the running time for all values of $\delta \geq \delta^\star := H^{-1}(1-\lambda)$ is solely dominated by (c). Observe that we have $(c) \geq (b)$, whenever

$$\begin{aligned} & 2^{\lambda d} \cdot p^{2r} \geq p^{r-1} \\ \Leftrightarrow \quad & H(\delta) \geq 1 - \frac{\lambda r}{r+1} \\ \Leftrightarrow \quad & \delta \geq H^{-1}\Big(1 - \frac{\lambda}{1+\frac{1}{r}}\Big) \to H^{-1}(1-\lambda) = \delta^\star \ , \end{aligned}$$

since $\frac{1}{r} = o(1)$. Also we have $(c) \geq (a)$ for the same choice of delta, as

$$\begin{aligned} & 2^{2\lambda d} \cdot p^{2r} \geq 1 \\ \Leftrightarrow \quad & \delta \geq H^{-1}(1-\lambda) = \delta^\star \ . \end{aligned}$$

Thus, for all choices of $\delta \geq \delta^\star$ the running time is $(T_\delta)^{(1+o(1))}$ with

$$\vartheta^\star(\delta) := \frac{\log T_\delta}{d} = 2(\lambda + H(\delta) - 1) + (1-\gamma)\bigg(1 - H\Big(\frac{\delta - \frac{\gamma}{2}}{1-\gamma}\Big)\bigg) \ .$$

Now, minimizing $\vartheta^\star$ yields a global minimum at $\delta_{\min} = \frac{1}{2}(1 - \sqrt{1-2\gamma})$ attaining a value of

$$\vartheta^\star(\delta_{\min}) = 2\lambda + H(\gamma) - 1 \ .$$

As we are restricted to values for δ which are larger than $\delta^\star$ solving $\delta_{\min} \geq \delta^\star$ for γ yields

$$\begin{aligned} & \delta_{\min} \geq \delta^\star \\ \Leftrightarrow \quad & \gamma \geq 2\delta^\star(1-\delta^\star) = \gamma^\star \ . \end{aligned}$$

This proves the claim of the theorem whenever $\gamma > \gamma^\star$. For all other values of γ we simply choose $\delta = \delta^\star$, which yields

$$\vartheta = \vartheta^\star(\delta^\star) = (1-\gamma)\bigg(1 - H\Big(\frac{\delta^\star - \frac{\gamma}{2}}{1-\gamma}\Big)\bigg) \text{ for } \gamma \leq \gamma^\star$$

as claimed.

Now to boost the expected running time $2^{\vartheta d(1+o(1))}$ of the algorithm to actually being obtained with overwhelming probability we use a standard Markov argument. Let X denote the random variable describing the running time of the algorithm. Then the probability that the algorithm needs more time than $2^{\sqrt{d}}E[X]$ to finish is

$$\Pr\Big[X \geq 2^{\sqrt{d}} \cdot E[X]\Big] \leq \frac{E[X]}{2^{\sqrt{d}} \cdot E[X]} = 2^{-\sqrt{d}} \ ,$$

or equivalently the algorithm finishes in less time than $2^{\sqrt{d}}E[X] = 2^{\vartheta d(1+o(1))}$ with overwhelming probability. Also, a standard application of the union bound yields that the intersection of the algorithm finishing within the claimed time and the algorithm having success in finding the solution is still overwhelming. ◀

The theorem shows that whenever $\gamma > \gamma^*$ our algorithm obtains the optimal time complexity for uniformly random lists as given in Equation (1). Additionally, our algorithm reaches the time lower bound for locality-sensitive hashing based algorithms for all values of γ, whenever the input list sizes are subexponential in the dimension d, which is shown in the following lemma.

▶ **Lemma 8.** *Let $\gamma \in \left[0, \frac{1}{2}\right]$, and ϑ as defined in Theorem 3. Then we have*

$$\lim_{\lambda \to 0} \frac{\vartheta}{\lambda} = \frac{1}{1-\gamma} \ .$$

Proof. Note that for λ converging zero, $\delta^\star = H^{-1}(1-\lambda)$ approaches $\frac{1}{2}$. This implies $\gamma^\star := 2\delta^\star(1-\delta^\star) = \frac{1}{2}$ and hence for all choices of γ we have

$$\vartheta = (1-\gamma)\left(1 - H\left(\frac{\delta - \frac{\gamma}{2}}{1-\gamma}\right)\right) \ .$$

Now, for this choice of ϑ, May and Ozerov [17, Corollary 1] already showed the statement of this lemma, by applying L'Hopital's rule twice. ◀

For convenience, we restate all parameter choices of Algorithm 1 for solving the $\mathcal{CP}_{d,\lambda,\gamma}$ in the following overview:

$$r = \frac{d}{\log^2 d},\ P = (d+1)^{r+1},\ k = \frac{d}{r}$$
$$N = \frac{d}{q},\ \text{where } q = \binom{\gamma k}{\frac{1}{2}\gamma k}\binom{(1-\gamma)k}{\left(\delta - \frac{\gamma}{2}\right)k}\left(\frac{1}{2}\right)^k$$
$$\delta = \begin{cases} \delta^\star & \text{for } \gamma \leq 2\delta^\star(1-\delta^\star) \\ \frac{1}{2}(1-\sqrt{1-2\gamma}) & \text{else} \end{cases}\ ,\ \text{with } \delta^\star := H^{-1}(1-\lambda) \tag{8}$$

4 Different Input Distributions

In this section, we show how to adapt the analysis of Algorithm 1 to variable input distributions. Therefore, we first reformulate Theorem 4 in Corollary 9 for the case of considering the $\mathcal{CP}_{d,\lambda,\gamma}$ over an arbitrary distribution $\mathcal{D}$. As already indicated in the proof of Theorem 4, this reformulation depends on the expected value $\mathcal{E}$ of the cost of the naive search at the bottom of the computation tree, which is highly influenced by the distribution $\mathcal{D}$. Then, we show how to compute $\mathcal{E}$ and how to upper bound it effectively. Finally, we give upper bounds for the time complexity of the algorithm to solve the $\mathcal{CP}_{d,\lambda,\gamma}$ over some generic distributions. These examples suggest that the algorithm is best suited for distributions $\mathcal{D}$, where the weight of the sum $\mathbf{v}+\mathbf{w}$ of elements $\mathbf{v}, \mathbf{w} \sim \mathcal{D}$ concentrates at $\frac{d}{2}$.[4]

Let us start with the reformulation of the theorem.

▶ **Corollary 9.** *Let $\mathcal{D}$ be some distribution over $\mathbb{F}_2^d$, q and p be as defined in Equation (2), $\gamma \in \left[0, \frac{1}{2}\right]$, $\lambda \in [0,1]$ and $r = \frac{\lambda d}{\log^2 d}$. Also let $\mathcal{E} = \mathbb{E}[|A| \cdot |B|]$ for A and B in line 13 of Algorithm 1 (where the expectation is taken over the distribution of input lists and the random choices of the algorithm). Then Algorithm 1 solves the $\mathcal{CP}_{d,\lambda,\gamma}$ problem over $\mathcal{D}$ in time*

$$\max\left(q^{-r}, \frac{2^{\lambda d} \cdot p^{r-1}}{q^r}, \frac{\mathcal{E}}{q^r}\right)^{1+o(1)}$$

with success probability overwhelming in d.

[4] This behavior seems quite natural as in this case, the solution is most distinguishable from random input pairs.

Proof. The proof follows along the lines of the proof of Theorem 4, by observing that $T_{r+1} = N^r \cdot \mathcal{E}$ and the expected time complexity is again amplified to being obtained with overwhelming probability by using a Markov argument similar to the proof of Theorem 3. ◀

In the next lemma, we show how to upper bound the value of $\mathcal{E}$.

▶ **Lemma 10** (Expectation of Naive Search). *Let $\mathcal{D}$ be some distribution over $\mathbb{F}_2^d$, $\gamma \in \left[0, \frac{1}{2}\right]$, $\lambda \in [0,1]$ and $r = \frac{\lambda d}{\log^2 d}$. Also let $\mathcal{E} = \mathbb{E}[|A| \cdot |B|]$ for A and B in line 13 of Algorithm 1 when solving some instance of the $\mathcal{CP}_{d,\lambda,\gamma}$ over $\mathcal{D}$ (where the expectation is taken over the distribution of input lists and the random choices of the algorithm). Then we have*

$$\mathcal{E} \leq 2^{2\lambda d} \prod_{i=1}^{r} \alpha_i + 4r \cdot 2^{\lambda d} \cdot p^r$$

where $\alpha_i := \Pr_{\mathbf{v},\mathbf{w} \sim \mathcal{D}} \left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}}) = \delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}}) = \delta k \right]$.

Proof. See Appendix A.1. ◀

While Lemma 10 gives an upper bound on the required expectation, it is not very handy. In the next lemma, we show how to further bound this expectation and how it affects the running time of the algorithm.

▶ **Lemma 11** (Complexity for Arbitrary Distributions). *Let $\mathcal{D}$ be some distribution over $\mathbb{F}_2^d$, $r := \frac{\lambda d}{\log^2 d}$, $\gamma \in \left[0, \frac{1}{2}\right]$ and $\lambda \in [0,1]$. Also let $\mathcal{E} = \mathbb{E}[|A| \cdot |B|]$ for A and B in line 13 of Algorithm 1 when solving some instance of the $\mathcal{CP}_{d,\lambda,\gamma}$ over $\mathcal{D}$ (where the expectation is taken over the distribution of input lists and the random choices of the algorithm). Then Algorithm 1 solves the $\mathcal{CP}_{d,\lambda,\gamma}$ over $\mathcal{D}$ in time*

$$\max\left(q^{-r}, \frac{2^{\lambda d} \cdot p^{r-1}}{q^r}, \frac{2^{\varepsilon d}}{q^r} \right)^{1+o(1)} ,$$

where

$$\varepsilon = 2\lambda - \min_{\substack{i \in [r] \\ \eta \in [0,1]}} (1-\eta)\left(1 - H\left(\frac{\delta - \frac{\eta}{2}}{1-\eta}\right)\right) - \frac{r \cdot \log p_{i,\eta k}}{d}$$

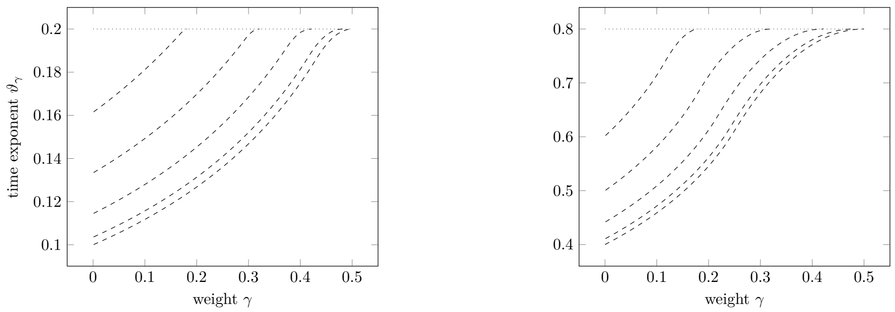
with $p_{i,\eta k} := \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = \eta k\right]$.

Proof. See Appendix A.2. ◀

Note that if it further holds that for $\mathbf{v} \sim \mathcal{D}$ each of the r blocks of $\mathbf{v}$ is identically distributed we can further simplify the term of ε from Lemma 11. In this case, we have $p_{i,\eta k}^r \leq \Pr\left[\mathrm{wt}(\mathbf{v}+\mathbf{w}) = \eta d\right] := p_{\eta d}$, thus we get

$$\varepsilon = 2\lambda - \min_{\eta \in [0,1]} (1-\eta)\left(1 - H\left(\frac{\delta - \frac{\eta}{2}}{1-\eta}\right)\right) - \frac{\log p_{\eta d}}{d} .$$

Now if we are given an arbitrary distribution $\mathcal{D}$ we can maximize ε according to η. Then we can similar to the proof of Theorem 3 derive a value for δ minimizing the overall time complexity.



(a) List sizes $|L_1| = |L_2| = 2^{0.1d}$.

(b) List sizes $|L_1| = |L_2| = 2^{0.4d}$.

Figure 3 Time complexity exponents as a function of the weight of the closest pair for different input list distributions, where the expected weight of input elements is equal to $0.1d$, $0.2d$, $0.3d$, $0.4d$, $0.5d$ from left to right.

We performed this maximization and optimization numerically for some generic input distributions. We considered distributions, where the weight of input vectors is distributed *binomially*, chosen according to a *Poisson* distribution or *fixed* to a specific value. This means, first a weight is sampled according to the chosen distribution and then a vector of that weight is selected uniformly among all vectors of that weight.

The running time of Algorithm 1 for solving the $\mathcal{CP}_{d,\lambda,\gamma}$ over the considered distributions seems to be only dependent on the expected weight of vectors contained in the input lists. That means the time complexity for input lists containing random vectors whose weight is either fixed to ηd or binomially or Poisson distributed with expectation ηd is equal. This can possibly be explained by the low variance of all these distributions, which implies a high concentration around this expected weight.

We see in Figure 3, that the value for γ, from where on the complexity becomes quadratic in the lists sizes shifts to the left. This behavior stems from the fact, that the expected weight of a sum of elements is no longer $\frac{d}{2}$, but roughly $2\eta(1-\eta)d$. What also stands out is, that the complexity for $\gamma = 0$ is no longer linear in the lists sizes. The reason for this is that the probability of random pairs falling into the same bucket and the probability of the closest pair falling into the same bucket converge for decreasing weight of input list elements. This indicates that for input distributions with smaller expected weight a different bucketing criterion might be beneficial. We pose this as an open question for further research.

References

1 Josh Alman. An illuminating algorithm for the light bulb problem. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, volume 69 of *OASICS*, pages 2:1–2:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASIcs.SOSA.2019.2.

2 Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 793–801, 2015.

3 Yoshinori Aono, Phong Q Nguyen, Takenobu Seito, and Junji Shikata. Lower bounds on lattice enumeration with extreme pruning. In *Annual International Cryptology Conference*, pages 608–637. Springer, 2018.

4 Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 10–24, Arlington, VA, USA, January 10–12 2016. ACM-SIAM. doi:10.1137/1.9781611974331.ch2.

5 Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

6 Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 25–46, Fort Lauderdale, Florida, United States, April 9–11 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-79063-3_2.

7 Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for lpn security. In *International Conference on Post-Quantum Cryptography*, pages 25–46. Springer, 2018.

8 Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

9 Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Transactions on Information Theory*, 56(8):4166–4179, 2010.

10 Cheikh Thiécoumba Gueye, Jean Belo Klamti, and Shoichi Hirose. Generalization of bjmm-isd using may-ozerov nearest neighbor algorithm over an arbitrary finite field $\mathbb{F}_q$. In *International Conference on Codes, Cryptology, and Information Security*, pages 96–109. Springer, 2017.

11 Shoichi Hirose. May-ozerov algorithm for nearest-neighbor problem over $\mathbb{F}_q$ and its application to information set decoding. In *International Conference for Information Technology and Communications*, pages 115–126. Springer, 2016.

12 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *30th Annual ACM Symposium on Theory of Computing*, pages 604–613, Dallas, TX, USA, May 23–26, 1998. ACM Press. doi:10.1145/276698.276876.

13 Matti Karppa, Petteri Kaski, and Jukka Kohonen. A faster subquadratic algorithm for finding outlier correlations. In Robert Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1288–1305, Arlington, VA, USA, January 10–12, 2016. ACM-SIAM. doi:10.1137/1.9781611974331.ch90.

14 Samir Khuller and Yossi Matias. A simple randomized sieve algorithm for the closest-pair problem. *Information and Computation*, 118(1):34–37, 1995.

15 Jiwen Lu, Venice Erin Liong, Xiuzhuang Zhou, and Jie Zhou. Learning compact binary face descriptor for face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(10):2041–2056, 2015.

16 Jonathan Marchini, Peter Donnelly, and Lon R Cardon. Genome-wide strategies for detecting multiple loci that influence complex diseases. *Nature genetics*, 37(4):413–417, 2005.

17 Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46800-5_9.

18 Rajeev Motwani, Assaf Naor, and Rina Panigrahi. Lower bounds on locality sensitive hashing. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 154–157, 2006.

19 Solomon K Musani, Daniel Shriner, Nianjun Liu, Rui Feng, Christopher S Coffey, Nengjun Yi, Hemant K Tiwari, and David B Allison. Detection of gene× gene interactions in genome-wide association studies of human population data. *Human heredity*, 63(2):67–84, 2007.

20 Christoph Strecha, Alex Bronstein, Michael Bronstein, and Pascal Fua. Ldahash: Improved matching with smaller descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 34(1):66–78, 2011.

21 Gregory Valiant. Finding correlations in subquadratic time, with applications to learning parities and juntas. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 11–20. IEEE, 2012.

22 Leslie G Valiant. Functionality in neural nets. In *COLT*, volume 88, pages 28–39, 1988.

23 Ning Xie, Shuai Xu, and Yekun Xu. A new coding-based algorithm for finding closest pair of vectors. *Theoretical Computer Science*, 782:129–144, 2019.

A Proofs Regarding Different Input Distributions

In this section, we give the proofs for the lemmata regarding the performance and correctness of our algorithm applied to different input distributions that were omitted in the main body.

A.1 Proof of Lemma 10

Similar to the proof of Theorem 4, let us bound $\mathbb{E}[|A_i|\cdot|B_i|]$ in terms of $\mathbb{E}[|A_{i-1}|\cdot|B_{i-1}|]$, $\mathbb{E}[|A_{i-1}|]$ and $\mathbb{E}[|B_{i-1}|]$ for each i.

$$\begin{aligned}\mathbb{E}[|A_i|\cdot|B_i| \mid A_{i-1},B_{i-1}] = {} & \sum_{\substack{\mathbf{v}\in A_{i-1}\setminus\{\mathbf{x}\}\\ \mathbf{w}\in B_{i-1}\setminus\{\mathbf{y}\}}} \underbrace{\Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}})=\delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}})=\delta k\right]}_{=:\alpha_i} \\ & + \sum_{\mathbf{v}\in A_{i-1}} \underbrace{\Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}})=\delta k, \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}})=\delta k\right]}_{\le p} \\ & + \sum_{\mathbf{w}\in B_{i-1}} \underbrace{\Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}})=\delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}})=\delta k\right]}_{\le p} \\ & + \Pr\left[\mathrm{wt}((\mathbf{x}+\mathbf{z})_{B_{i,r}}))=\delta k, \mathrm{wt}((\mathbf{y}+\mathbf{z})_{B_{i,r}}))=\delta k\right] \\ & \le \alpha_i\cdot|A_{i-1}|\cdot|B_{i-1}| + p\cdot(|A_{i-1}|+|B_{i-1}|+1)\end{aligned}$$

and hence $\mathbb{E}[|A_i|\cdot|B_i|] \le \alpha_i\cdot\mathbb{E}[|A_{i-1}|\cdot|B_{i-1}|]+p\cdot(\mathbb{E}[|A_{i-1}|]+\mathbb{E}[|B_{i-1}|]+1)$. Again, applying this equation successively, we obtain

$$\mathcal{E}=\mathbb{E}[|A_r|\cdot|B_r|] \le 2^{2\lambda d}\prod_{i=1}^{r}\alpha_i + 4\cdot 2^{\lambda d}\cdot\sum_{i=1}^{r}\left(\prod_{j=0}^{i-2}\alpha_{r-j}\right)p^{r-i+1} \le 2^{2\lambda d}\prod_{i=1}^{r}\alpha_i + 4r\cdot 2^{\lambda d}\cdot p^r \ . \quad ◀$$

A.2 Proof of Lemma 11

Taking the result for $\mathcal{E}$ from Lemma 10 and plugging into the run time formula from Corollary 9 we get that the $\mathcal{CP}_{d,\lambda,\gamma}$ problem over $\mathcal{D}$ can be solved with probability overwhelming in d in time

$$\max\left(q^{-r}, \frac{2^{\lambda d}\cdot p^{r-1}}{q^r}, \frac{\mathcal{E}}{q^r}\right)^{1+o(1)} \le \max\left(q^{-r}, \frac{2^{\lambda d}\cdot p^{r-1}}{q^r}, \frac{2^{2\lambda d}\prod_{i=1}^{r}\alpha_i}{q^r}\right)^{1+o(1)}$$

since the right summand $\frac{4r\cdot 2^{\lambda d}\cdot p^r}{q^r}$ of $\frac{\mathcal{E}}{q^r}$ is asymptotically smaller than the second entry in the max, i.e. $\frac{2^{\lambda d}\cdot p^{r-1}}{q^r}$. Thus, is suffices to find an easier upper bound for the first summand $S := 2^{2\lambda d}\prod_{i=1}^{r}\alpha_i$. Remembering $\alpha_i = \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}})=\delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}})=\delta k\right]$ we receive

$$\begin{aligned}
S &\leq 2^{2\lambda d} \cdot \left(\max_{i \in [r]} \alpha_i \right)^r \\
&= 2^{2\lambda d} \cdot \left(\max_{i \in [r]} \sum_{j=0}^{k} q_{i,j} \cdot \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = j \right] \right)^r \\
&\leq 2^{2\lambda d + o(d)} \cdot \left(\max_{i \in [r],\, j \in [k] \cup \{0\}} q_{i,j} \cdot \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = j \right] \right)^r \\
&= 2^{2\lambda d + o(d)} \cdot \left(\max_{i \in [r],\, \eta \in [0,1]} q_{i,\eta k} \cdot \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = \eta k \right] \right)^r \quad ,
\end{aligned}$$

where $q_{i,\eta k} = \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{z})_{B_{i,r}}) = \delta k, \mathrm{wt}((\mathbf{w}+\mathbf{z})_{B_{i,r}}) = \delta k \mid \mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = \eta k \right]$. Lemma 7 lets us rewrite this probability as

$$q_{i,\eta k} = \binom{\eta k}{\frac{1}{2}\eta k} \binom{(1-\eta)k}{\left(\delta - \frac{\eta}{2}\right)k} \left(\frac{1}{2}\right)^k \leq 2^{-\left(1 - H\left(\frac{\delta - \frac{\eta}{2}}{1-\eta}\right)\right)(1-\eta)k} \quad .$$

We end up with

$$\begin{aligned}
S &\leq 2^{2\lambda d + r \cdot \max\limits_{i \in [r],\, \eta \in [0,1]} -\left(1 - H\left(\frac{\delta - \frac{\eta}{2}}{1-\eta}\right)\right)(1-\eta)k + \log p_{i,\eta k} + o(d)} \\
&= 2^{\left(2\lambda + \max\limits_{i \in [r],\, \eta \in [0,1]} -\left(1 - H\left(\frac{\delta - \frac{\eta}{2}}{1-\eta}\right)\right)(1-\eta) + \frac{r}{d} \cdot \log p_{i,\eta k}\right) d + o(d)} \\
&= 2^{\left(2\lambda - \min\limits_{i \in [r],\, \eta \in [0,1]} \left(1 - H\left(\frac{\delta - \frac{\eta}{2}}{1-\eta}\right)\right)(1-\eta) - \frac{r}{d} \cdot \log p_{i,\eta k}\right) d + o(d)}
\end{aligned}$$

with $p_{i,\eta k} := \Pr\left[\mathrm{wt}((\mathbf{v}+\mathbf{w})_{B_{i,r}}) = \eta k \right]$, which proves the claim. ◀

B Practical Experiments

In this section, we give experimental results of the performance of a proof of concept implementation of our new algorithm. These experiments verify the performance gain of our algorithm over a naive quadratic search approach. We also verify the numerical estimates of the algorithm's performance on different input distributions from the previous section and give some practical related improvements to our algorithm. Our implementation is publicly available at `https://github.com/FloydZ/NNAlgorithm`.

Before discussing the benchmark results let us first briefly describe some of the practical improvements we introduced in our implementation, which differ from the description in Section 3. We implemented a true depth-first search rather than the iterative description given previously. The iterative description just allowed for a more convenient analysis. Thus, our algorithm needs to store only the lists of a single path from the root to a leaf node at any time. Also, as all lists of subsequent levels are subsets of previous ones, we do not create r different lists. We rather rearrange the elements of the input list such that elements belonging to the list of the subsequent level are consecutive, making it sufficient to just memorize the range of elements that belong to the next level list. This way, we only need to store the input list plus two integer markers for each level. Also, it turns out that in practice often a small depth of the tree (not exceeding 8 in our experiments) is already sufficient to achieve good runtime results. Regarding the branching factor N of the tree, we achieve

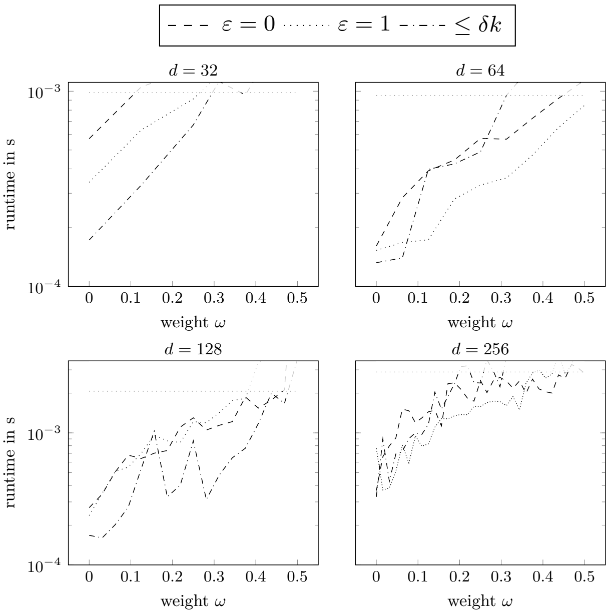


Figure 4 Runtime in seconds in logarithmic scale (y-axis) as a function of distance ω of the closest pair (x-axis) on random lists of size 2^{10}. Dotted, dashed and dash-dotted lines indicate results for different bucketing, straight horizontal line is the time used by a naive quadratic search.

optimal results either for values close to its expectation $\frac{1}{q}$ as given by the analysis or values being significantly smaller. The case of using a very small branching factor can be seen as a pruning strategy, similar to the one used in lattice enumeration algorithms for shortest vector search [3]. Additionally, we benchmarked three different strategies for the weight criteria:

1. Strictly enforcing a weight of δk in each block, as described in our algorithm.
2. Allowing for a small deviation $\pm\varepsilon$ around δk.
3. Allowing for weights of *at most* δk.

Further, we introduced a threshold for the size of the lists in the tree, below which the computation of further leaves is aborted and naive search is used instead.

Figure 4 shows the runtime results for the different bucket criteria on small input lists of size 2^{10} containing random elements. Here, each data point was averaged over 50 measurements. The experimental results clearly indicate a significant gain over the quadratic search approach. The less significant gain for small dimension d is due to the reduced amount of possible blocks or equivalently the low depth of the computation tree, which lets the algorithm not reach its full potential. In the case of small input lists, we observe that a bucketing strategy that allows a deviation of $\varepsilon = 1$ from δk is beneficial for most values of d.

Figure 5 shows the same experiments performed on larger input lists of size 2^{15}. Besides a more significant improvement over the naive search, we can observe that the bucketing criterion that uses δk as an upper bound becomes more beneficial for nearly all values of γ and d.

Eventually, Figure 6 shows the experimental runtime results on input lists, whose elements are drawn from a different input distribution, analyzed in Section 4. Here the distribution is the uniformly random distribution over vectors of weight ηd. One can observe that for growing d the shape of the graph resembles the theoretical results from Figure 3.

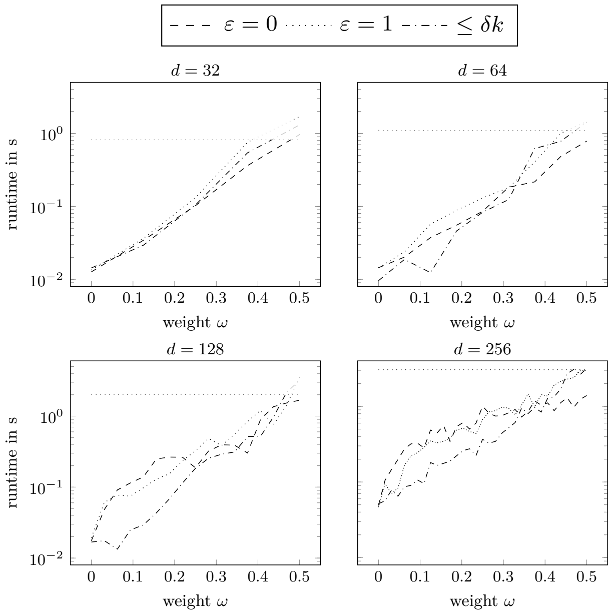


Figure 5 Runtime in seconds in logarithmic scale (y-axis) as a function of distance ω of the closest pair (x-axis) on random lists of size 2^{15}. Dotted, dashed and dash-dotted lines indicate results for different bucketing strategies, straight horizontal line is time used by a naive quadratic search.

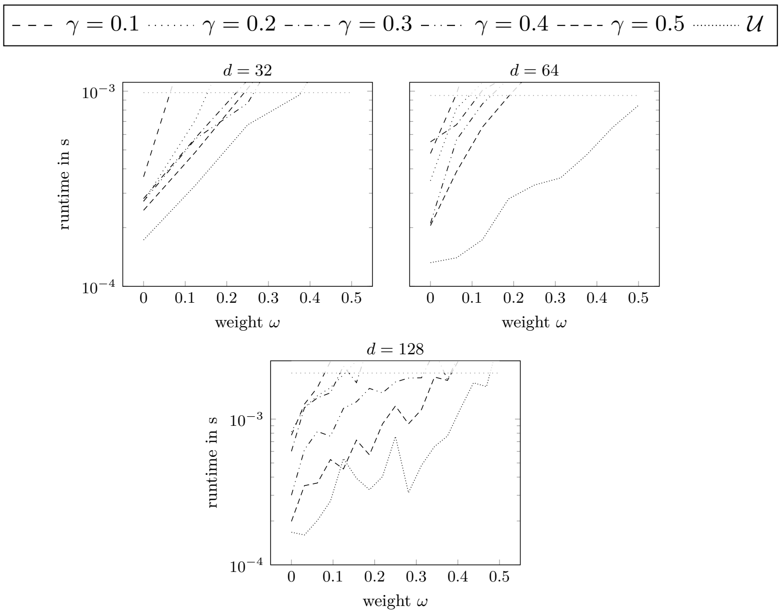


Figure 6 Runtime in seconds in logarithmic scale (y-axis) as a function of distance ω of the closest pair (x-axis) on lists of size 2^{10} containing random elements of weight γd. The densely dashed line ($\mathcal{U}$) indicates the runtime on uniformly random lists.

ETH Tight Algorithms for Geometric Intersection Graphs: Now in Polynomial Space

Fedor V. Fomin
University of Bergen, Norway

Petr A. Golovach
University of Bergen, Norway

Tanmay Inamdar
University of Bergen, Norway

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

Abstract

De Berg et al. in [SICOMP 2020] gave an algorithmic framework for subexponential algorithms on geometric graphs with tight (up to ETH) running times. This framework is based on dynamic programming on graphs of weighted treewidth resulting in algorithms that use super-polynomial space. We introduce the notion of weighted treedepth and use it to refine the framework of de Berg et al. for obtaining polynomial space (with tight running times) on geometric graphs. As a result, we prove that for any fixed dimension $d \geq 2$ on intersection graphs of similarly-sized fat objects many well-known graph problems including Independent Set, r-Dominating Set for constant r, Cycle Cover, Hamiltonian Cycle, Hamiltonian Path, Steiner Tree, Connected Vertex Cover, Feedback Vertex Set, and (Connected) Odd Cycle Transversal are solvable in time $2^{\mathcal{O}(n^{1-1/d})}$ and within polynomial space.

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

Keywords and phrases Subexponential Algorithms, Geometric Intersection Graphs, Treedepth, Treewidth

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.21

Related Version *Full Version*: `https://arxiv.org/abs/2107.06715`

Funding The research leading to these results has received funding from the Research Council of Norway via the project "'MULTIVAL" (grant no. 263317) and the European Research Council (ERC) via grant LOPPRE, reference 819416.

1 Introduction

Most of the fundamental NP-complete problems on graphs like Independent Set, Feedback Vertex Set, or Hamiltonian Cycle do not admit algorithms of running times $2^{o(n)}$ on general graphs unless the Exponential Time Hypothesis (ETH) fails. However, on planar graphs, H-minor-free graphs, and several classes of geometric graphs, such problems admit *subexponential* time algorithms. There are several general frameworks for obtaining subexponential algorithms [4, 6, 8]. The majority of these frameworks utilize dynamic programming algorithms over graphs of bounded treewidth. Consequently, the subexponential algorithms derived within these frameworks use prohibitively large (exponential) space.

We consider another related graph parameter, namely, *treedepth*. Given a graph $G = (V, E)$, a pair (F, φ) is a treedepth decomposition of G, if F is a rooted forest, and $\varphi : V(F) \to V(G)$ is a mapping such that the neighbors in G are mapped to vertices in F that

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 21; pp. 21:1–21:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

have an ancestor-descendant relationship. Then, the treedepth of G is the minimum height of the forest over all treedepth decompositions. Alternatively, the treedepth of G can be thought of as the elimination distance to the family of edgeless graphs (see the book of Nesetril and de Mendez [17] for more details). Recently, algorithms on graphs of bounded treedepth attracted significant attention [9, 10, 15]. The advantage of these algorithms over dynamic programming used for treewidth is that they use polynomial space. Our work is motivated by the following natural question

> Could the treedepth find applications in the design of (polynomial space) subexponential algorithms?

The problem is that the treedepth of a graph could be significantly larger than its treewidth. For example, the treewidth of an n-vertex path is one, while the treedepth is of order $\log n$. It creates problems in using treedepth in frameworks like bidimensionality that strongly exploit the existence of large grid minors in graphs of large treewidth. Despite that, we show the usefulness of treedepth for obtaining polynomial space subexponential algorithms on intersection graphs of some geometrical objects.

In [4], de Berg et al. developed a generic framework facilitating the construction of subexponential algorithms on large classes of geometric graphs. By applying their framework on intersection graphs of similarly-sized fat objects in dimension $d \geq 2$, de Berg et al. obtained algorithms with running time $2^{\mathcal{O}(n^{1-1/d})}$ for many well-known graph problems, including Independent Set, r-Dominating Set for constant r, Hamiltonian Cycle, Hamiltonian Path, Feedback Vertex Set, Connected Dominating Set, and Steiner Tree.

The primary tool introduced by de Berg et al. is the weighted treewidth. They show that solving many optimization problems on intersection graph of n similarly-sized fat objects can be reduced to solving these problems on graphs of weighted treewidth of order $\mathcal{O}(n^{1-1/d})$. Combined with single-exponential algorithms on graphs of bounded weighted treewidth, this yields subexponential algorithms for several problems.

The running times $2^{\mathcal{O}(n^{1-1/d})}$ are tight – de Berg et al. accompanied their algorithmic upper bounds with matching conditional complexity (under ETH) bounds. However, as most of the treewidth-based algorithms, the algorithms of Berg et al. are dynamic programming over tree decompositions. As a result, they require super-polynomial space. Thus a concrete question here is *whether running times $2^{\mathcal{O}(n^{1-1/d})}$ could be achieved using polynomial space.*

We answer this question affirmatively by developing polynomial space algorithms that in time $2^{\mathcal{O}(n^{1-1/d})}$ solve all problems on intersection graphs of similarly-sized fat objects from the paper of de Berg et al. except for Connected Dominating Set. The primary tool in our work is the *weighted treedepth.* To the best of our knowledge, this notion is new.

The Cut&Count technique was introduced by Cygan et al. [3], who gave the first single-exponential (randomized) algorithms parameterized by the treewidth for many problems using this technique. We note that at the heart of these algorithms is a dynamic programming over the tree decomposition, and thus require exponential space. However, unweighted treedepth was recently used by several authors in the design of parameterized algorithms using polynomial space [9, 10, 15]. Some of these works adapt the Cut&Count technique for the treedepth decomposition.

Our main insight is that in the framework of de Berg et al. [4] for most of the problems the weighted treedepth can replace the weighted treewidth. Pipelined with branching algorithms over graphs of small weighted treedepth, this new insight brings us to many tight (up to ETH) polynomial space algorithms on geometric graphs.

Our results. To explain our strategy of "replacing" the weighted treewidth with the weighted treedepth, we need to provide an overview of the framework of de Berg et al. [4]. It has two main ingredients. First, for an intersection graph of n similarly-sized fat objects (we postpone technical definitions to the next section), we construct an auxiliary weighted graph $G_{\mathcal{P}}$. (Roughly speaking, to create $G_{\mathcal{P}}$, we contract some cliques of G and assign weights to the new vertices.) Then the combinatorial theorem of de Berg et al. states that the weighted treewidth of $G_{\mathcal{P}}$ is $\mathcal{O}(n^{1-1/d})$. Second, to solve problems on G in time $2^{\mathcal{O}(n^{1-1/d})}$, one uses a tree decomposition of $G_{\mathcal{P}}$. This part is problem-dependent and, for some problems, could be pretty non-trivial.

To plug in the treedepth into this framework, we first prove that the weighted treedepth of $G_{\mathcal{P}}$ is $\mathcal{O}(n^{1-1/d})$. Moreover, we give an algorithm computing a treedepth decomposition in time $2^{\mathcal{O}(n^{1-1/d})}$ and polynomial space. For Independent Set, a simple branching algorithm over the treedepth decomposition can solve the problem in time $2^{\mathcal{O}(n^{1-1/d})}$ and polynomial space. We also get a similar time and space bounds for Dominating Set, and more generally, r-Dominating Set for constant r; however, we need to use a slightly different kind of recursive algorithm.

Next, we consider connectivity problems like Steiner Tree, Connected Vertex Cover, Feedback Vertex Set, and (Connected) Odd Cycle Transversal. For these problems, we are able to adapt the single exponential FPT algorithms parameterized by (unweighted) treedepth given by Hegerfeld and Kratsch [10], into the framework of weighted treedepth decomposition. Thus, we get $2^{\mathcal{O}(n^{1-1/d})}$ time, polynomial space algorithms for these problems.

Finally, we consider Cycle Cover, which is a generalization of Hamiltonian Cycle. Here, we are able to "compress" the given graph into a new graph, such that the (unweighted) treedepth of the new graph is $\mathcal{O}(n^{1-1/d})$. We can also compute the corresponding treedepth decomposition in $2^{\mathcal{O}(n^{1-1/d})}$ time, and polynomial space. Then, we can use a result by Nederlof et al. [15] as a black box, which is a Cut&Count based a single exponential FPT algorithms parameterized by treedepth, that uses polynomial space. Thus, we get $2^{\mathcal{O}(n^{1-1/d})}$ time, polynomial space algorithms for Cycle Cover, Hamiltonian Cycle, and also for Hamiltonian Path.

We note that the results in the previous two paragraphs are based on the Cut&Count technique, and are randomized. We also note that all of our algorithms, except for Cycle Cover and related problems can work even without the geometric representation of the similarly-sized fat objects. For Cycle Cover and related problems, however, we require the geometric representation. This is in line with similar requirements for these problems from [4].

Organization. In Section 2, we define some of the basic concepts including the weighted treedepth, and then prove our main result about the same. At the end of the section, we give a warm-up example of an algorithm for Independent Set using this framework. Then, in Section 3 we describe the Cut&Count algorithm, and its application for Steiner Tree using the notion of weighted treedepth. Finally, we give a rough sketch of our approach for Cycle Cover in Section 4. The algorithms for the remaining problems are omitted from the main version due to page limit, but they can be found in the full version of the paper.

2 Geometric Graphs and Weighted Treedepth

In this section we define the weighted treedepth, prove a combinatorial bound on the treedepth of certain geometric graphs and provide a generic algorithm and provide an abstract theorem modeling at a high level our subexponential time and polynomial space algorithms. But first, we need some definitions.

Graphs. We consider only undirected simple graphs and use the standard graph theoretic terminology; we refer to the book of Diestel [7] for basic notions. We write $|G|$ to denote $|V(G)|$, and throughout the paper we use n for the number of vertices if it does not create confusion. For a set of vertices $S \subseteq V(G)$, we denote by $G[S]$ the subgraph of G induced by the vertices from S and write $G-S$ to denote the graph obtained by deleting the vertices of S. For a vertex v, $N_G(v)$ denotes the *open neighborhood* of v, that is, the set of vertices adjacent to v, and $N_G[v] = \{v\} \cup N_G(v)$ is the *closed neighborhood.* For a vertex v, $d_G(v) = |N_G(v)|$ denotes the *degree* of v. We may omit subscripts if it does not create confusion. For two distinct vertices u and v of a graph G, a set $S \subseteq V(G)$ is a (u,v)*-separator* if $G - S$ has no (u,v)-path and S is a *separator* if S is a (u,v)-separator for some vertices u and v. A pair of vertex subsets (A,B) is called a separation if $A \cup B = V(G)$, and there are no edges between $A \setminus B$ and $B \setminus A$, that is, $S = A \cap B$ is a (u,v)-separator for $u \in A \setminus B$ and $v \in B \setminus A$. We say that a subset $S \subseteq V(G)$ is an α-balanced separator for a constant $\alpha \in (0,1)$ if there exists a separation (A,B) such that $A \cap B = S$, and $\max\{|A|,|B|\} \leq \alpha n$.

κ-partition Let $\mathcal{P} = \{V_1, V_2, \ldots, V_t\}$ be a partition of $V(G)$ for some $t \geq 1$, such that any $V_i \in \mathcal{P}$ satisfies the following properties: (1) $G[V_i]$ is connected, and (2) V_i is a union of at most κ cliques in G (not necessarily disjoint). Then, we say that $\mathcal{P}$ is a κ-partition of G. Furthermore, given a κ-partition $\mathcal{P} = \{V_1, V_2, \ldots, V_t\}$ of G, we define the graph $G_{\mathcal{P}}$, the graph induced by $\mathcal{P}$, as the undirected graph obtained by contracting each V_i to a vertex, and removing self-loops and multiple edges.

Treedepth and Weighted Treedepth. We introduce *weighted treedepth* of a graph as a generalization of the well-known notion of treedepth (see e.g. the book of Nesetril and de Mendez [17]). There are different ways to define treedepth but it is convenient for us to deal with the definition via *treedepth decompositions* or *elimination forests.* We say that a forest F supplied with one selected node (it is convenient for us to use the term "node" instead of "vertex" in such a forest) in each connected component, called a *root*, a *rooted forest.* The choice of roots defines the natural parent–child relation on the nodes of a rooted forest. Let G be a graph and let $\omega\colon V(G) \to \mathbb{R}$ be a weight function. A *treedepth decomposition* of G is a pair (F, φ), where F is a rooted forest and $\varphi\colon V(F) \to V(G)$ is a bijective mapping such that for every edge $uv \in E(G)$, either $\varphi^{-1}(u)$ is an ancestor of $\varphi^{-1}(v)$ in F or $\varphi^{-1}(v)$ is an ancestor of $\varphi^{-1}(u)$. Then the *depth* of the decomposition is the depth of F, that is, the maximum number of nodes in a path from a root to a leaf. The *treedepth* of G, denoted $\mathtt{td}(G)$, is the minimum depth of a treedepth decomposition of G. We define the *weighted depth* of a treedepth decomposition as the maximum $\sum_{v \in V(P)} \omega(\varphi(v))$ taken over all paths P between roots and leaves. Respectively, the *weighted treedepth* $\mathtt{wtd}(G)$ is the minimum weighted depth of a treedepth decomposition. For our applications, we assume without loss of generality that G is connected, which implies that the forest F in a (weighted) treedepth decomposition is actually a tree.

Weighted Treewidth. We assume basic familiarity with the notion of treewidth and tree decomposition of a graph – see a textbook such as [2], for example. Similar to the previous paragraph, de Berg et al. [4] define the *weighted treewidth* of a graph. Given an undirected graph $G = (V, E)$ with weights $\omega \colon V(G) \to \mathbb{R}$, the weighted width of a tree decomposition (T, β), is defined to be the maximum over bags, the sum of the weights of vertices in the bag. The weighted treewidth of a graph is the minimum weighted width over all tree decompositions of the graph.

It is useful to observe that we consider treedepth and tree decompositions of the graphs $G_{\mathcal{P}}$ constructed for graphs G with given κ-partition $\mathcal{P} = \{V_1, V_2, \dots, V_t\}$. Then the treedeph decomposition of $G_{\mathcal{P}}$ can be seen as a pair (F, φ), where F is a rooted forest and φ is a bijective mapping of $V(F)$ to $\mathcal{P}$. Similarly, in a tree decomposition (T, β) of $G_{\mathcal{P}}$, corresponding to every node $t \in V(T)$, the bag $\beta(t)$ is a subset of $\mathcal{P}$. Finally, we observe that the results of [4] regarding weighted treewidth – thus our results for weighted treedepth – hold for any weight functions $\omega : \mathcal{P} \to \mathbb{R}^+$, provided that $\omega(\ell) = \mathcal{O}(\ell^{1-1/d-\epsilon})$, for any $\epsilon > 0$. However, as in [4], we will fix the weight function to be $\omega(\ell) \coloneqq \log(1 + \ell)$ throughout the rest of the paper. For the simplicity of notation, we use the shorthand $\omega(u_i) \coloneqq \omega(|V_i|)$, where $\varphi(u_i) = V_i$, and for any $u_i \in V(F)$, and $\omega(S) \coloneqq \sum_{u_i \in S} \omega(u_i)$ for any subset $S \subseteq V(F)$.

Geometric Definitions. Given a set F of objects in $\mathbb{R}^d$, we define the corresponding intersection graph $G[F] = (V, E)$, where there is a bijection between an object in F and $V(G)$, and $uv \in E(G)$ iff the corresponding objects in F have a non-empty intersection. It is sometimes convenient to erase the distinction between F with $V(G)$, and to say that each vertex is a geometric object from F.

We consider the geometric intersection graphs of *fat objects*. A geometric object $g \subset \mathbb{R}^d$ is said to be α-fat for some $\alpha \geq 1$, if there exist balls $B_{\text{in}}, B_{\text{out}}$ such that $B_{\text{in}} \subseteq g \subseteq B_{\text{out}}$, such that the ratio of the radius of B_{out} to that of B_{in} is at most α. We say that a set F of objects is *fat* if there exists a constant $\alpha \geq 1$ such that every geometric object in F is α-fat. Furthermore, we say that F is a set of *similarly-sized* fat objects, if the ratio of the largest diameter of an object in F, to the smallest diameter of an object in F is at most a fixed constant. Finally, observe that if F is a set of *similarly-sized fat objects*, then the ratio of the largest out-radius to the smallest in-radius of an object is also upper bounded by a constant. de Berg et al. [4] prove the following two results regarding the intersection graphs of similarly sized fat objects.

▶ **Lemma 1** ([4]). *Fix dimension $d \geq 2$. There exist constants κ and Δ, such that for any intersection graph $G = (V, E)$ of an (unknown) set of n similarly-sized fat objects in $\mathbb{R}^d$, a κ-partition $\mathcal{P}$ for which $G_{\mathcal{P}}$ has maximum degree Δ can be computed in time polynomial in n.*

In the following, we will use the tuple $(G, d, \mathcal{P}, G_{\mathcal{P}})$ to indicate that $G = (V, E)$ is the intersection graph of n similarly-sized fat objects in $\mathbb{R}^d$, $\mathcal{P}$ is a κ-partition of G such that $G_{\mathcal{P}}$ has maximum degree Δ, where κ, Δ are constants, as guaranteed by Lemma 1.

▶ **Lemma 2** ([4]). *For any $(G, d, \mathcal{P}, G_{\mathcal{P}})$, the weighted treewidth of $G_{\mathcal{P}}$ is $\mathcal{O}(n^{1-1/d})$.*

Now we are ready to prove the following result about the intersection graphs of similarly sized fat objects. This result is at the heart of the subexponential algorithms designed in the following sections.

▶ **Theorem 3.** *There is a polynomial space algorithm that for a given $(G, d, \mathcal{P}, G_{\mathcal{P}})$, computes in time $2^{\mathcal{O}(n^{1-1/d})}$ a weighted treedepth decomposition (F, φ) of $G_{\mathcal{P}}$ of weighted treedepth $\mathcal{O}(n^{1-1/d})$.*

Proof. We use the approximation algorithm from [18] to compute a weighted tree decomposition (T, β) of $G_{\mathcal{P}}$ (see the later part of the proof for a detailed explanation). Using the standard properties of the tree decomposition (e.g., see [2]), there exists a node $t \in V(T)$, such that $V_B := \bigcup_{V_i \in \beta(t)} V_i$ is an α-balanced separator for G, for some $\alpha \leq 2/3$. Let $B := \beta(t)$. Note that $B \subseteq \mathcal{P}$.

Now we construct a part of the forest F, and the associated bijection φ in the weighted treedepth decomposition (F, φ) of $G_{\mathcal{P}}$. We create a path $\pi = (u_1, u_2, \ldots, u_{|B|})$, and arbitrarily assign $\varphi(u_i)$ to some $V_i \in B$ such that it is a bijection. We set u_1, the first vertex on π, to be the root of a tree in F. We also set the weight $\omega(u_i) = \log(1 + |V_i|)$, where $\varphi(u_i) = V_i$. Note that the $\omega(\pi) = \omega(B) = \mathcal{O}(n^{1-1/d})$.

Let (Y_1, Y_2) be the separation of G, corresponding to the separator $\bigcup_{V_i \in \beta(t)} V_i$. Analogously, let $(\mathcal{P}'_1, \mathcal{P}'_2)$ denote the separation of $G_{\mathcal{P}}$, corresponding to the separator B. Furthermore, let $X_i := Y_i \setminus V_B$, and $\mathcal{P}_i := \mathcal{P}'_i \setminus B$ for $i = 1, 2$. Note that $X_1 \setminus V_B, X_2 \subseteq V(G)$ are disjoint, $\max\{|X_1|, |X_2|\} \leq \alpha n$, and there is no edge from a vertex in X_1, to a vertex in X_2. Furthermore, $\mathcal{P}_1$ is a κ-partition of $G[X_1]$, and $\mathcal{P}_2$ is a κ-partition of $G[X_2]$.

Now, we recursively construct weighted treedepth decomposition (F_1, φ_1) of $G_{\mathcal{P}}[\mathcal{P}_1]$. Note that φ_1 is a bijection between $V(F_1)$ and $\mathcal{P}_1 \subseteq \mathcal{P}$. Let R_1 denote the set of roots of the trees in forest F_1. We add an edge from the last vertex $u_{|B|}$ on the path π, to each root in R_1. In other words, we attach every tree in F_1 as a subtree below $u_{|B|}$. The bijection φ is extended to $\mathcal{P}_1$ using φ_1. Now we consider a weighted treedepth decomposition (F_2, φ_2) of $G_{\mathcal{P}}[\mathcal{P}_2]$, and use it to extend (F, φ) in a similar manner. This completes the construction of (F, φ).

Let us first analyze the weighted treedepth of (F, φ). Let us use $q := 1 - 1/d$ for simplicity. For a path π in F, let $\omega(\pi)$ denote the sum of weights of vertices along the path π. Recall that the weight of any root-leaf path π in F is at most $\mathcal{O}(n^q)$. More generally, let $c' \geq 0$ be a universal constant (independent of the path π, or its level in F) such that the weight of a path corresponding to a separator computed at level j, is at most $c' \cdot (\alpha^{j-1} n)^q$. Since $\max\{|X_1|, |X_2|\} \leq \alpha n$, we inductively assume that the weighted treedepth of (F_1, φ_1), and that of (F_2, φ_2) is at most $\mathcal{O}(\alpha^q \cdot n^q)$. More specifically, we assume that there exists a universal constant $c \geq c'$, such that the sum of the weights along any root-leaf path in F_1 is upper bounded by $c \cdot \frac{(\alpha n)^q}{1-\alpha^q}$. The same inductive assumption holds for any root-leaf path in F_2. Therefore, the weight of any root-leaf path in F is upper bounded by

$$\omega(\pi) + c \cdot \frac{\alpha^q n^q}{1 - \alpha^q} \leq c n^q \left(1 + \frac{\alpha^q}{1 - \alpha^q}\right) = \frac{c n^q}{1 - \alpha^q}.$$

Therefore, we have the desired bound on the weighted treedepth by induction.

Now we look the treewidth construction part of the algorithm in order to sketch the claims about bounds on time and space. Given the graph $G_{\mathcal{P}}$, we construct a graph H by replacing every vertex V_i with a (new) clique C_i of size $\log(1+|V_i|)$. If $V_i V_j \in E(G_{\mathcal{P}})$, we also add edges from every vertex in C_i to every vertex in C_j. As shown in [4], the weighted width of $G_{\mathcal{P}}$ is equal to the treewidth of H, plus 1. Note that $|V(H)| = \sum_{V_i \in \mathcal{P}} \log(1 + |V_i|) \leq n$, since $\mathcal{P}$ is a partition of $V(G)$.

The algorithm from [18] (see also Section 7.6.2 in the Parameterized Algorithms book [2]) for approximating treewidth of a graph H works as follows. Suppose the treewidth of a graph is k, which is known. At the heart of this algorithm is a procedure $\texttt{decompose}(W, S)$, where $S \subsetneq W \subseteq V(H)$, and $|S| \leq 3k + 4$. This procedure tries to decompose the subgraph $H[W]$ in such a way that S is completely contained in one bag of the tree decomposition. The first step is to compute a partition (S_A, S_B) of S, such that the size of the separator separating S_A and S_B in $H[W]$ is at most $k+1$. This is done by exhaustively guessing all partitions, which takes $2^{\mathcal{O}(k)}$ time. For each such guess of (S_A, S_B), we run a polynomial time algorithm to check whether the bound on the separator size holds. Once such a partition is found, a set $\hat{S} \supsetneq S$ is found by augmenting S in a particular way. Finally, we recursively run the procedure $\texttt{decompose}(N_H[D], N_H(D))$, for each connected component D in $H[W \setminus \hat{S}]$. Finally, the tree decomposition of $H[W]$ is computed by augmenting the tree decompositions computed by the recursive procedure for its children, with the root bag containing $\hat{S}$. It is shown that this algorithm computes a tree decomposition of width $\mathcal{O}(\texttt{tw})$ in time $2^{\mathcal{O}(\texttt{tw})} \cdot n^{\mathcal{O}(1)}$. Furthermore, it can also be observed that it only uses polynomial space.

Therefore, computing a tree decomposition of $G_{\mathcal{P}}$ of weighted treewidth $\mathcal{O}(n^{1-1/d})$ takes $2^{\mathcal{O}(n^{1-1/d})}$ time and polynomial space, corresponding to the original graph G with n vertices. The treewidth computation algorithm is called at most n times, and there is additional polynomial processing at every step. This implies the time and space bounds as claimed. ◀

We note that de Berg et al. [4] show the existence of a balanced separator of weight $\mathcal{O}(n^{1-1/d})$, which is then used to show the same bound on weighted treewidth (Theorem 2). This separator can be computed in $\mathcal{O}(n^{d+2})$ time if we are also given the geometric representation of the underlying objects in $\mathbb{R}^d$. However, without geometric representation it is not clear whether this separator can be directly computed. Therefore, we first compute an approximate weighted treewidth decomposition, and then retrieve the separator bag in the proof of Theorem 3. We state the following abstract theorem that models at a high level our subexponential algorithms that use polynomial space. The proof of this theorem follows from Theorem 3, and can be found the full version.

▶ **Theorem 4.** *Let $\mathcal{A}$ be an algorithm for solving a problem on graph G, that takes input $(G, d, \mathcal{P}, G_{\mathcal{P}})$, and a weighted treedepth decomposition (F, φ) of $G_{\mathcal{P}}$ of weighted depth $\mathcal{O}(n^{1-1/d})$ (and optionally additional inputs of polynomial size). Suppose $\mathcal{A}$ is a recursive algorithm, that at every node $u \in V(F)$, spends time proportional to $2^{\mathcal{O}(\omega(u))} \cdot n^{\mathcal{O}(1)}$, uses polynomial space, and makes at most $2^{\mathcal{O}(\omega(u))}$ recursive calls on the children of u. Then, the algorithm $\mathcal{A}$ runs in time $2^{\mathcal{O}(n^{1-1/d})}$, and uses polynomial space.*

Independent Set. As a warm-up example for using the weighted treedepth decomposition, we describe an application for Independent Set. Given $(G, d, \mathcal{P}, G_{\mathcal{P}})$, we first observe that every $V_i \in \mathcal{P}$ is a union of at most κ cliques, which implies that the intersection of an independent set with any V_i is bounded by κ. A recursive algorithm for Independent Set works with the weighted treedepth decomposition (F, φ) computed via Theorem 3. When the algorithm is at a node $u_i \in V(F)$, we make a recursive call to the children of u_i, corresponding to each independent subset $U_i \subseteq \varphi(u_i) = V_i$ of size at most κ, that is independent. We recursively compute a Maximum Independent Set in the subgraph of G, corresponding to the subtree rooted at each children of u_i, with the vertices in $N(U_i)$ removed. We return the maximum independent set found over all choices of the subset U_i. Finally, we observe that the number of subsets of V_i of size at most κ is at most $(1 + |V_i|)^\kappa = 2^{\mathcal{O}(\log(1+|V_i|))} = 2^{\mathcal{O}(\omega(u_i))}$, where we use the fact that $\kappa = \mathcal{O}(1)$. A more formal description of the algorithm can be found in the full version.

▶ **Theorem 5.** *There exists a $2^{\mathcal{O}(n^{1-1/d})}$ time, polynomial space algorithm to compute a maximum (weight) independent set in the intersection graphs of similarly sized fat objects in $\mathbb{R}^d$.*

r-Dominating Set. For a fixed $r \geq 1$, r-Dominating Set asks for a minimum-size vertex subset $D \subseteq V(G)$, such that for every $v \in V(G)$, there exists some $u \in D$ such that $dist_G(u, v) \leq r$, where $dist_G(u, v)$ is the number of edges on the shortest path in G between

Algorithm	Space	Similarly-sized	Convex	Robust
Corollary 2.4 in [4]	Poly	Yes	Yes	No
Theorem 2.13 in [4]	$2^{\mathcal{O}(n^{1-1/d})}$	Yes	No	Yes
Theorem 5 (this paper)	Poly	Yes	No	Yes

Figure 1 Comparison of three $2^{\mathcal{O}(n^{1-1/d})}$-time algorithms for Independent Set on the intersection graphs of fat objects. "Robust" in the last column means that the algorithm does not require the geometric representation of the objects. In terms of technique, our algorithm (third row) is closely related to the one in the first row, albeit we use the framework of weighted treedepth.

u and v. Following [4], there are the two important ingredients in our algorithm for r-Dominating Set. First, de Berg et al. [4] show that it can be assumed that $|V_i \cap D| \leq \kappa^2(1+\Delta)$ for any $V_i \in \mathcal{P}$. However, this property alone is not sufficient to obtain a recursive algorithm that runs in subexponential time and polynomial space.

Consider a similar recursive algorithm that is processing a vertex $V_i \in \mathcal{P}$ using a weighted treedepth decomposition (F, φ) of $G_{\mathcal{P}}$. There are three possibilities for a vertex $u \in V_i$ – (i) it is in the dominating set, (ii) it is already being dominated by a vertex that was added to the dominating set at an earlier stage of recursion, or (iii) it will be dominated by a vertex $v \in V_j$ with $dist_G(u, v) \leq r$, where V_j belongs to the subtree of F rooted at V_i. To handle case (iii), we need to enumerate partial solutions from *all* V_j's such that $dist_{G_{\mathcal{P}}}(V_i, V_j) \leq r$. However, the weighted treedepth bound given in Theorem 3 is not sufficient, and we need a strengthened version of the theorem. Such a result appears in [4], and we reprove it in the full version for completeness. Loosely speaking, this result bounds the total weight of *all* the bags that appear within the r-neighborhood of the α-balanced separator obtained via the weighted treewidth decomposition. Armed with this result, the recursive algorithm "guesses" the set of vertices from the balanced separator bag, and for each vertex u type (iii), it also guesses a vertex v from the subtree that dominates u. The stronger theorem implies that the number of recursive calls made from the i-th level of recursion can still be bounded by $2^{\mathcal{O}((\alpha^{i-1}n)^{1-1/d})}$. A formal description and analysis of this algorithm can be found in the full version of the paper. We summarize our result in the following theorem.

▶ **Theorem 6.** *For any fixed $r \geq 1$, there exists a $2^{\mathcal{O}(n^{1-1/d})}$ time, polynomial space algorithm to compute a minimum r-dominating set in the intersection graphs of similarly sized fat objects in $\mathbb{R}^d$.*

3 Cut&Count Algorithms

Hegerfeld and Kratsch [10] adapt the Cut&Count technique to give FPT algorithms for various connectivity based subset problems, parameterized by (unweighted) treedepth. In particular, these algorithms are randomized, have running times of the form $2^{\mathcal{O}(\mathtt{td})} \cdot n^{\mathcal{O}(1)}$, and use polynomial space. In their work, they consider Connected Vertex Cover, Feedback Vertex Set, Connected Dominating Set, Steiner Tree, and Connected Odd Cycle Transversal problems. We are able to adapt their technique for all of these problems, except for Connected Dominating Set. For the rest of the problems, we will extend their ideas to the more general case of *weighted treedepth*, and use it to give $2^{\mathcal{O}(n^{1-1/d})}$ time, polynomial space, randomized algorithms. In the following, we select Steiner Tree as a representative problem, which is explained in detail. For the remaining problems, we give only a brief sketch highlighting the differences from the Steiner Tree algorithm, and defer the formal details to the full version.

3.1 Setup

We adopt the following notation from Hegerfeld and Kratsch [10]. Let $\texttt{cc}(G)$ denote the number of connected components in G. A cut of $X \subseteq V(G)$ is a pair (X_L, X_R), where $X_L \cap X_R = \emptyset$, $X_L \cup X_R = X$. We refer to X_L, X_R as the left and the right side of the cut (X_L, X_R) respectively. A cut (X_L, X_R) of $G[X]$ is consistent, if for any $u \in X_L$ and $v \in X_R$, $uv \notin E(G[X])$. A *consistently cut subgraph* of G is a pair $(X, (X_L, X_R))$, such that $X \subseteq V(G)$, and (X_L, X_R) is a consistent cut of $G[X]$. Finally, for $X \subseteq V(G)$, we denote the set of consistently cut subgraphs of $G[X]$ by $\mathcal{C}(X)$.

For $n \in \mathbb{N}$, let $[n]$ denote the set of integers from 1 to n. For integers a, b, we write $a \equiv b$ to indicate equality modulo 2. We use Iverson's bracket notation: for a boolean predicate p, $[p]$ is equal to 1 if p is true, otherwise $[p]$ is equal to 0.

Consider a function $f : A \to S$. For every $s \in S$ and a set X, we define the set $X(f, s) \coloneqq X \cap f^{-1}(s)$ – note that $X(f, s)$ may be empty for some or all $s \in S$. Furthermore, observe that the sets $\{A(f, s)\}_{s \in S}$ define a partition of A. For two functions $g : A \to S$, $f : B \to S$, we define the new function $g \oplus f : (A \cup B) \to S$ as follows. $(g \oplus f)(e) = f(e)$ for $e \in B$, and $(g \oplus f)(e) = g(e)$ for $e \in (A \setminus B)$. That is, $(g \oplus f)$ behaves like g and f on the exclusive domains, but in case of a conflict, the function f takes the priority.

Recall that we work with $(G, d, \mathcal{P}, G_{\mathcal{P}})$, and the corresponding weighted treedepth decomposition (F, φ) of G. Here, φ is a bijection between $V(F)$ and $\mathcal{P}$. For a node u_i, we will use $V_i \coloneqq \varphi(u_i)$, i.e., we use the same indices in the subscript to identify a node of F and the corresponding part in $\mathcal{P}$. We denote the set of children of u_i by $\texttt{child}(u_i)$. Additionally,

$$\texttt{tail}[u_i] = \bigcup_{u_j \text{ is an ancestsor of } u_i} V_j \; ; \qquad \texttt{tail}(u_i) = \texttt{tail}[u_i] \setminus V_i$$

$$\texttt{tree}[u_i] = \bigcup_{u_j \text{ is a descendant of } u_i} V_j \; ; \qquad \texttt{tree}(u_i) = \texttt{tree}[u_i] \setminus V_i$$

$$\texttt{broom}[u_i] = \texttt{tail}[u_i] \cup \texttt{tree}(u_i)$$

Isolation Lemma

▶ **Definition 7.** *Let U be a finite set, and $\mathcal{F} \subseteq 2^U$ be a family of subsets of U. We say that a weight function $\mathbf{w} : U \to \mathbb{Z}$* isolates *the family $\mathcal{F}$ if there exists a unique set $S' \in \mathcal{F}$ such that $\mathbf{w}(S') = \min_{S \in \mathcal{F}} \mathbf{w}(S)$, where $\mathbf{w}(X) \coloneqq \sum_{x \in X} \mathbf{w}(x)$ for any subset $X \subseteq U$.*

The following isolation lemma due to Mulmuley et al. [14] is at the heart of all Cut&Count algorithms.

▶ **Lemma 8** ([14])**.** *Let $\mathcal{F} \subseteq 2^U$ be a non-empty family of subsets of a finite ground set U. Let $N \in \mathbb{N}$, and suppose $\mathbf{w}(u)$ is chosen uniformly and independently at random from $[N]$ for every $u \in U$. Then, $\Pr(\mathbf{w}$ isolates $\mathcal{F}) \geq 1 - |U|/N$.*

General Idea

Fix a problem involving connectivity constraints. Let U be the ground set that is related to the graph G, such that $\mathcal{S} \subseteq 2^U$, where $\mathcal{S}$ denotes the set of solutions to the problem. At a high level, a Cut&Count based algorithm contains the following two parts.

- **The Cut part:** We obtain a set $\mathcal{R}$ by relaxing the connectivity requirements on the solutions, such that $\mathcal{S} \subseteq \mathcal{R} \subseteq 2^U$. The set $\mathcal{Q}$ will contain pairs (X, C), where $X \in \mathcal{R}$ is a candidate solution, and C is a consistent cut of X. Note that since $X \in \mathcal{R}$, X may be possibly disconnected.

- **The Count part:** We compute $|\mathcal{Q}| \mod 2$ using an algorithm. The consistent cuts are defined carefully, in order that the non-connected solutions from $\mathcal{R} \setminus \mathcal{S}$ cancel while counting modulo 2, since they are consistent with an even number of cuts.

Note that if $|\mathcal{S}|$ is even, then the procedure counting $|\mathcal{Q}| \mod 2$ will return 0, which will be inconclusive. Therefore, we initially sample a random weight function $\mathbf{w} : U \to [N]$ for some large integer $N \geq 2|U|$, and count $|\mathcal{Q}_w| \mod 2$ (where $\mathcal{Q}_w$ is the subset of $\mathcal{Q}$ such that the corresponding X has weight *exactly* w), for all values of $w \in [2|U|^2]$. Using Lemma 8, it can be argued that with at least probability 1/2, if $\mathcal{S} \neq \emptyset$, then for some weight $w \in [2|U|^2]$, the procedure counting $|\mathcal{Q}_w| \mod 2$ outputs 1. Finally, we guess an arbitrary vertex $v_1 \in V(G)$ in the solution, and force it to be on the left side of the consistent cuts. That is, we count the number of consistent cuts in which v_1 is forced to belong to the left side. This breaks the left-right symmetry. We first have the following two results from [10, 3].

▶ **Lemma 9** ([10, 3]). *Let $X \subseteq V(G)$ such that $v_1 \in X$. The number of consistently cut subgraphs $(X, (X_L, X_R))$ such that $v_1 \in X_L$ is equal to $2^{\mathrm{cc}(G[X])-1}$.*

▶ **Corollary 10** ([10, 3]). *Let $\mathcal{S} \subseteq 2^U$, and $\mathcal{Q} \subseteq 2^{U \times (V \times V)}$, such that for every $\mathbf{w} : U \to [2|U|]$, and a target weight $w \in [2|U|^2]$, the following two properties hold.*

1. *$|\{(X, C) \in \mathcal{Q} : \mathbf{w}(X) = w\}| = |\{X \in \mathcal{S} : \mathbf{w}(X) = w\}|$, and*
2. *There is an algorithm **CountC**$(\mathbf{w}, w, (G, d, \mathcal{P}, G_{\mathcal{P}}), (F, \varphi))$, where (F, φ) is a weighted treedepth decomposition of $(G, d, \mathcal{P}, G_{\mathcal{P}})$, such that: **CountC**$(\mathbf{w}, w, (G, d, \mathcal{P}, G_{\mathcal{P}}), (F, \varphi)) \equiv |\{(X, C \in \mathcal{Q} : \mathbf{w}(X) = w)\}|$.*

*Then, Algorithm 1 returns **false** if $\mathcal{S} = \emptyset$, and returns **true** with probability at least $\frac{1}{2}$ otherwise.*

Proof. Plugging in $\mathcal{F} = \mathcal{S}$ and $N = 2|U|$ in Lemma 8, we know that if $\mathcal{S} \neq \emptyset$, then with probability at least 1/2, there exists a weight $w \in [2|U|^2]$ such that $|\{X \in \mathcal{S} : \mathbf{w}(X) = w\}| = 1$. Then, Algorithm 1 returns **true** with probability at least 1/2.

On the other hand, if $\mathcal{S} = \emptyset$, then by the first property, and the definition of `CountC`, for any choice of $\mathbf{w}$ and w, the procedure `CountC` returns **false**. Therefore, Algorithm 1 returns **false**. ◀

Algorithm 1 Cut&Count$(U, (G, d, \mathcal{P}, G_{\mathcal{P}}), (F, \varphi), \texttt{CountC})$.

```
Input: A set $U$, $(G, d, \mathcal{P}, G_{\mathcal{P}})$, associated weighted treedepth decomposition $(F, \varphi)$, a procedure CountC that takes $\mathbf{w} : U \to [N], w \in \mathbb{N}$
Choose $\mathbf{w}(u)$ independently and uniformly at random from $[2|U|]$ for each $u \in U$
for $w = 1, 2, \ldots, 2|U|^2$ do
    if CountC$((G, d, \mathcal{P}, G_{\mathcal{P}}), (F, \varphi), \mathbf{w}, w) \equiv 1$ return true
end for
return false
```

3.2 Steiner Tree

▶ **Definition 11** (Steiner Tree).
Input: An undirected graph $G = (V, E)$, a set of terminals $K \subseteq V(G)$, and an integer k.
Question: Is there a subset $X \subseteq V(G)$, with $|X| \leq k$, such that $G[X]$ is connected, and $K \subseteq X$?

Fix $(G, d, \mathcal{P}, G_{\mathcal{P}})$ via Lemma 1. Recall that $\mathcal{P}$ is a κ-partition of G, such that the corresponding graph $G_{\mathcal{P}}$ has maximum degree $\Delta = \mathcal{O}(1)$. We first have the following lemma.

▶ **Lemma 12** ([4]). *Suppose X is a minimal solution for Steiner Tree (i.e., no proper subset of X is also a solution) for a given $(G, d, \mathcal{P}, G_{\mathcal{P}})$, and a set of terminals K. Then $|X \cap (V_i \setminus K)| \leq \kappa^2(\Delta + 1)$ for any $V_i \in \mathcal{P}$.*

Let $k' = |K| + \kappa^2(\Delta+1) \cdot |\mathcal{P}|$. Note that using Lemma 12, we may assume that $k \leq k'$ – if $k \geq k'$, then (G, k) is a "yes-instance" iff (G, k') is a "yes-instance". For any $X \subseteq V(G)$, we say that X is $\mathcal{P}$-restricted if for any $V_i \in \mathcal{P}$, $|X \cap (V_i \setminus K)| \leq \kappa^2(\Delta + 1)$. Note that this definition of a $\mathcal{P}$-restricted set (and later, that of a $\mathcal{P}$-restricted function) is specific to the Steiner Tree problem. For different problems, we need to define this notion differently, albeit the main idea is to use a problem-specific version of Lemma 12.

We will run the following algorithm for all values of $k \leq k'$. Let $t_1 \in K$ be an arbitrary terminal that we will fix to be on the left side of consistent cuts, as discussed previously. Now we give the formal definitions of the sets $\mathcal{R}, \mathcal{S}, \mathcal{Q}$ that were abstractly defined in the setup. We also define weight-restricted versions $\mathcal{R}_w, \mathcal{S}_w, \mathcal{Q}_w$ of these sets, where $w \in \mathbb{N}$.

$$\mathcal{R} = \{X \subseteq V(G) : X \text{ is } \mathcal{P}\text{-restricted}, K \subseteq X, |X| = k\}; \quad \mathcal{R}_w = \{X \in \mathcal{R} : \mathbf{w}(X) = w\}$$
$$\mathcal{S} = \{X \in \mathcal{R} : G[X] \text{ is connected}\}; \quad \mathcal{S}_w = \{X \in \mathcal{S} : \mathbf{w}(X) = w\}$$
$$\mathcal{Q} = \{(X, (X_L, X_R)) \in \mathcal{C}(V) : X \in \mathcal{R} \text{ and } t_1 \in X_L\}; \quad \mathcal{Q}_w = \{(X, (X_L, X_R)) \in \mathcal{Q} : \mathbf{w}(X) = w\}$$

▶ **Lemma 13.** *Let $\mathbf{w} : V(G) \to [N]$ be a weight function. Then, for every $w \in \mathbb{N}$, $|\mathcal{S}_w| \equiv |\mathcal{Q}_w|$.*

Proof. From Lemma 9, $|\mathcal{Q}_w| = \sum_{X \in \mathcal{R}_w} 2^{\mathtt{cc}(G[X])-1}$.
Thus, $|\mathcal{Q}_w| \equiv |\{X \in \mathcal{R}_w : \mathtt{cc}(G[X]) = 1\}| = |\mathcal{S}_w|$. Recall that $\equiv$ is equality modulo 2. ◀

The goal of the rest of this subsection is to explain how the procedure `CountC` works.

First, we drop the cardinality constraints and define the following candidates and candidate cut-pairs for induced subgraphs $G[V']$, where $V' \subseteq V(G)$.

$$\hat{\mathcal{R}}(V') = \{X \subseteq V' : X \text{ is } \mathcal{P}\text{-restricted, and } K \cap V' \subseteq X\}$$
$$\hat{\mathcal{Q}}(V') = \{(X, (X_L, X_R)) \in \mathcal{C}(V') : X \in \mathcal{R}(V') \text{ and } t_1 \in V' \Longrightarrow t_1 \in X_L\}$$

Recall that each node $u_i \in V(F)$ is bijectively mapped to a $V_i \in \mathcal{P}$. The algorithm will assign a value to every vertex $v \in V_i$ from the set $\mathtt{states} \coloneqq \{\mathbf{1}_L, \mathbf{1}_R, \mathbf{0}\}$, with the condition that if $v \in K \cap V_i$, then it cannot be assigned $\mathbf{0}$. The interpretation of the states $\mathbf{1}_L$ and $\mathbf{1}_R$ for a vertex $v \in V_i$ is that v is part of a candidate Steiner Tree solution, and is part of the left and the right side of the consistent cut, respectively. On the other hand, the vertices that are not part of a candidate Steiner Tree solution have the state $\mathbf{0}$. Next, we define an important notion of $\mathcal{P}$-restricted functions, which will be crucial for pruning the number of recursive calls.

▶ **Definition 14.** *Let $f : X \to \mathtt{states}$ be a function, where $X \subseteq V(G)$. We say that f is $\mathcal{P}$-restricted, if the following properties hold:*

- *$f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\})$ is $\mathcal{P}$-restricted, and*
- *$(X \cap K) \subseteq f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\})$, and if $t_1 \in X$, then $f(t_1) = \mathbf{1}_L$.*

The algorithm will be recursive, and it will compute a multivariate polynomial in the variables Z_W and Z_X, where the coefficient of the term $Z_W^w Z_X^i$ is equal to the cardinality of $\hat{Q}_w^i(V') \coloneqq \left\{(X, C) \in \hat{Q}(V') : \mathbf{w}(X) = w, |X| = i\right\}$, modulo 2. That is, the formal variables will keep track of the weight and the size of the solutions. The polynomial is computed by using a recursive algorithm that uses the weighted treedepth decomposition to guide recursion. The algorithm starts at the root r and proceeds towards the leaves.

Consider a node $u_i \in V(F)$, and a $\mathcal{P}$-restricted function $f : \mathtt{tail}[u_i] \to \mathtt{states}$, we define the set of partial solutions at u_i, but excluding any subset of V_i, that respect f by

$$\begin{aligned}\mathcal{C}_{(u_i)}(f) \coloneqq \Big\{(X,(X_L,X_R)) \in \hat{Q}(\mathtt{tree}(u_i)) :& X' = X \cup f^{-1}(\{\mathbf{1}_L,\mathbf{1}_R\}),\\ & C' = (X_L \cup f^{-1}(\mathbf{1}_L), X_R \cup f^{-1}(\mathbf{1}_R)),\\ & (X',C') \in \hat{Q}(\mathtt{broom}[u_i])\Big\} \qquad (1)\end{aligned}$$

That is, the partial solutions in $\mathcal{C}_{(u_i)}(f)$ are given by consistently cut subgraphs of $G[\mathtt{tree}(u_i)]$, that are extended to the candidate-cut-pairs for $G[\mathtt{broom}[u_i]]$ by f, i.e., consistently cut subgraphs of $G[\mathtt{broom}[u_i]]$ that contain all terminals in $\mathtt{broom}[u_i]$.

Similarly, for a node $u_i \in V(F)$, and a $\mathcal{P}$-restricted function $g : \mathtt{tail}(u_i) \to \mathtt{states}$, we define the set of partial solutions at u_i, but possibly including a subset of V_i, that respect g by $\mathcal{C}_{[u_i]}(g)$, whose definition is identical to (1) (after replacing f by g everywhere), except that the candidate consistently cut subgraph $(X,(X_L,X_R))$ is from the set $\hat{Q}[u_i]$.

With these definitions, the coefficients of the terms $Z_W^w Z_X^k$, for $0 \le w \le 2n^2$ in the polynomial $P_{[r]}(\emptyset)$ at the root node $r \in V(F)$ will give the desired quantities.

Recursively Computing Polynomials. Let $u_i \in V(F)$, and let $f : \mathtt{tail}[u] \to \mathtt{states}$ be a $\mathcal{P}$-restricted function. If u_i is a leaf in F, then

$$\begin{aligned}P_{(u_i)}(f) = &\big[(f^{-1}(\mathbf{1}_L), f^{-1}(\mathbf{1}_R)) \text{ is a consistent cut of } G[f^{-1}(\{\mathbf{1}_L,\mathbf{1}_R\})]\big]\\ \cdot\ &\big[K \cap \mathtt{tail}[u_i] \subseteq f^{-1}(\{\mathbf{1}_L,\mathbf{1}_R\})\big] \cdot [t_1 \in \mathtt{tail}[u_i] \Longrightarrow f(t_1) = \mathbf{1}_L] \qquad (2)\end{aligned}$$

If $u_i \in V(F)$ is not a leaf, then $$P_{(u_i)}(f) = \prod_{u_j \in \mathtt{child}(u_i)} P_{[u_j]}(f) \qquad (3)$$

To define the computation of $P_{[u_i]}(g)$ for a $\mathcal{P}$-restricted function $g : \mathtt{tail}(u_i) \to \mathtt{states}$, we need the following notation. Let $\mathcal{F}(V_i)$ be a set of $\mathcal{P}$-restricted functions (see Definition 14) from $V_i \to \mathtt{states}$ with the following additional property: for all $h \in \mathcal{F}(V_i)$, if $u, v \in h^{-1}(\{\mathbf{1}_L,\mathbf{1}_R\})$ with $uv \in E(G)$, then $h(u) = h(v)$. We refer to this additional property as the function being *cut-respecting.*

Note that g and any $h \in \mathcal{F}(V_i)$ have disjoint domains, and both are $\mathcal{P}$-restricted. Therefore, $g \oplus h$ is also $\mathcal{P}$-restricted for any $h \in \mathcal{F}(V_i)$. We have the following recurrence:

$$P_{[u_i]}(g) = \sum_{h \in \mathcal{F}(V_i)} P_{(u_i)}(g \oplus h) \cdot Z_W^{\mathbf{w}(V_i(h,\mathbf{1}))} Z_X^{|V_i(h,\mathbf{1})|} \qquad (4)$$

Where, we use the shorthand $V_i(h,\mathbf{1})$ for the set $V_i(h,\mathbf{1}_L) \cup V_i(h,\mathbf{1}_R)$.

At a high level, the correctness of the equations (2)-(4) essentially follows from the same arguments as in [10]. However, the details are rather technical because a recursive call made at a vertex $u_i \in V(F)$ corresponds to a function from $\mathcal{F}(V_i)$ that simultaneously assigns $\mathtt{states}$ to all the vertices in V_i. We defer the formal proof of correctness to the appendix.

Given recurrences (2-4), it is straightforward to compute polynomials $P_{(u_i)}(f)$ and $P_{[u_i]}(g)$ using a recursive algorithm. Finally, we return the coefficient of the term $Z_W^w Z_X^k$ in the polynomial $P_{[u_i]}(\emptyset)$ thus computed. The actual description of the algorithm can be found in the appendix.

▶ **Lemma 15.** *For any $V_i \in \mathcal{P}$, $|\mathcal{F}(V_i)| \le (1+|V_i|)^{\mathcal{O}(1)} = 2^{\mathcal{O}(\omega(V_i))}$. Furthermore, the set $\mathcal{F}(V_i)$ can be computed in poly($|\mathcal{F}(V_i)|, n$) time.*

Proof. Let $K_i = V_i \cap K$. Because of the first property from the definition of $\mathcal{P}$-restricted functions, there are at most $(1+|V_i|)^{\kappa^2(1+\Delta)}$ choices for selecting a subset $U_i \subseteq V_i \setminus K$ of size at most $\kappa^2(1+\Delta)$, to be mapped to $\{\mathbf{1}_L, \mathbf{1}_R\}$. Let us fix such a choice U_i. Note that every terminal in $K_i \coloneqq K \cap V_i$ must be assigned to $\{\mathbf{1}_L, \mathbf{1}_R\}$.

Due to the cut-respecting property, if there are two vertices $u, v \in U_i \cup K_i$ that belong to the same clique, then they must belong to the same side of the consistent cut. Since each V_i is a union of at most κ cliques, there are at most 2^κ choices for assigning vertices in $U_i \cup K_i$ to either side of a consistent cut. Therefore, since $\kappa, \Delta = \mathcal{O}(1)$, and $\omega(V_i) = \log(1+|V_i|)$, we have the following:

$$|\mathcal{F}(V_i)| \le (1+|V_i|)^{\kappa^2(1+\Delta)} \cdot 2^\kappa = 2^{\mathcal{O}(\omega(V_i))}.$$

Here we would like to highlight the distinction between the weights $\omega : \mathcal{P} \to \mathbb{R}^+$ from the weighted treedepth decomposition, the weights $\mathbf{w} : V(G) \to \mathbb{N}$ from the Isolation Lemma, and the target weight w for $\mathbf{w}$.

It is relatively straightforward to convert this proof into an algorithm for computing $\mathcal{F}(V_i)$. First, we can use a standard algorithm (e.g., [13]) to generate subsets U_i of size at most $\kappa^2(1+\Delta)$. It is known that this can be done in $|V_i|^{\mathcal{O}(\kappa^2(1+\Delta))}$ time.

Now, fix a particular choice of U_i, and consider the set $U_i \cup K_i$ as defined above. Now we compute an inclusion-wise maximal independent set S_i of $U_i \cup K_i$, e.g., by a greedy algorithm. Since V_i is a union of at most κ cliques, $|S_i| \le \kappa$. Now we consider at most 2^κ choices for assigning $\{\mathbf{1}_L, \mathbf{1}_R\}$ to each vertex in S_i. For any vertex $v \in (V_i \cup K_i) \setminus S_i$, there is a vertex $v' \in S_i$ such that vv' is an edge. Therefore, we set $f(v) = f(v')$. Note that if v has more than one neighbor in S_i, and if a particular choice assigns them different values, then this corresponds to a function that is *not* cut-respecting. In this case, we may move to the next assignment to S_i. Finally, since S_i is a maximal independent set, each cut-respecting function for the fixed choice of V_i will be considered in this manner. Finally, iterating over all choices of V_i, we can compute the set $\mathcal{F}(V_i)$ as claimed. ◀

▶ **Theorem 16.** *There exists a $2^{\mathcal{O}(n^{1-1/d})}$ time, polynomial space, randomized algorithm to solve* Steiner Tree *in the intersection graphs of similarly sized fat objects in $\mathbb{R}^d$.*

Proof. From Lemma 15, it follows that at every $u_i \in F$, the procedure `CountC` spends $2^{\mathcal{O}(\omega(u_i))} \cdot n^{\mathcal{O}(1)}$ time, and makes $2^{\mathcal{O}(\omega(u_i))}$ recursive calls to its children, corresponding to each function in $\mathcal{F}(V_i)$. Furthermore, since the weights defined by $\mathbf{w} : V(G) \to [2n]$ are polynomially bounded, at every node in F the algorithm uses space polynomial in n. We finally observe that the Cut&Count algorithm is a randomized procedure that makes polynomially many calls to `CountC`. The correctness of `CountC` follows from the correctness of recurrence relations, and the bounds on probability follow from Corollary 10. ◀

3.3 Other Problems

We design algorithms for Connected Vertex Cover, Feedback Vertex Set, and (Connected) Odd Cycle Transversal using Cut&Count technique. At a high level, the ideas that are similar to that for Steiner Tree from the previous section. However, there are a few crucial differences that are specific to the problem at hand. Here, we give a very brief sketch of how these differences are handled. A more formal description and analysis can be found in the full version.

Connected Vertex Cover. Recall that the goal is to find a vertex cover C of the smallest size for the given geometric intersection graph G that induces a connected subgraph of G. Since C is a vertex cover, it may leave out at most one vertex from any clique. Thus, $|V_i \setminus C| \leq \kappa$. This is the crucial observation (analogous to 12) that helps us prune the number of recursive calls, via bounding the number of $\mathcal{P}$-restricted functions as in Lemma 15. The details of the Cut&Count computation are very similar to that for Steiner Tree, with appropriate modifications. Indeed, there is a reduction from Connected Vertex Cover to Steiner Tree that increases the treedepth of the graph by at most 1, as observed in [3, 10]. This readily implies a $2^{\mathcal{O}(\mathtt{td})} \cdot n^{\mathcal{O}(1)}$ for Connected Vertex Cover. However, in the resulting instance of Steiner Tree, the resulting graph may not necessarily belong to the class of geometric intersection graphs, and thus may not have weighted treedepth of at most $\mathcal{O}(n^{1-1/d})$. Nevertheless, we are able to adapt the approach of [10] and get a $2^{\mathcal{O}(n^{1-1/d})}$ time randomized algorithm that uses polynomial space.

Feedback Vertex Set. Again, we observe that any feedback vertex set S may leave out at most two vertices from any clique of G, thus, $|V_i \setminus C| \leq 2\kappa$ – otherwise $G \setminus S$ will not be acyclic. Although the high level idea is similar to Steiner Tree and Connected Vertex Cover, the technical details need to be adapted to the peculiarities of the FVS problem.

(Connected) Odd Cycle Transversal. Let us first focus on the connected version. As for FVS, any Connected Odd Cycle transversal C may leave out at most two vertices from a clique – otherwise there will be a triangle in the $G \setminus C$. This implies that $|V_i \setminus C| \leq 2\kappa$ for any $V_i \in \mathcal{P}$ as earlier. This observation, combined with the ideas from [10] gives the desired subexponential time algorithm with polynomial space. Finally, we observe that an instance of OCT can be reduced to Connected OCT by adding a new universal vertex that is adjacent to all the original vertices. Note that the new graph may not necessarily belong to the class of geometric intersection graphs. Nevertheless, we can use the previous algorithm for Connected OCT as follows. The first observation is that the universal vertex can be assumed to be the root of the weighted treedepth decomposition, and there are at most 4 recursive calls made by the algorithm from the root. The rest of the algorithm works with the original graph, which is indeed a geometric intersection graph. Thus, the algorithm for Connected OCT also solves OCT, up to a constant factor increase in the running time.

4 Cycle Cover

▶ **Definition 17** (Cycle Cover).
Input: An undirected graph $G = (V, E)$, and an integer k.
Question: Do there exist at most k vertex-disjoint cycles that span $V(G)$?

Note that the case of $k = 1$ corresponds to determining whether G has a Hamiltonian cycle, that is, to the Hamiltonian Cycle problem.

We briefly sketch our approach for Cycle Cover. In this section, we assume that we are also given the geometric representation of the similarly-sized fat objects involved in the input graph G. With the geometric representation, we can use a stronger result from [4] that computes $(G, d, \mathcal{P}, G_{\mathcal{P}})$ with an additional property that $\mathcal{P}$ is a *clique cover* of G, i.e., $\mathcal{P}$ is a partition of $V(G)$ into cliques. Furthermore, the maximum degree Δ of $G_{\mathcal{P}}$ is a constant.

In the second step, we compute a graph H in polynomial time, such that G has a cycle cover of size k iff H has a cycle cover of size k. For this, we use ideas similar to [1, 11] to argue that the cycles can be rereouted to ensure that the size of the set of “boundary vertices", (i.e., vertices from which a cycle enters or leaves the clique) from each clique $V_i \in \mathcal{P}$, is upper bounded by $\mathcal{O}(\Delta)$. Then, since the degree of each V_i is at most Δ, it can be shown that all but $\mathcal{O}(\Delta^3) = \mathcal{O}(1)$ vertices of V_i can be discarded without changing the answer for Cycle Cover.

In our algorithm, we first construct the weighted treedepth decomposition of G, of weighted depth at most $\mathcal{O}(n^{1-1/d})$. Then, we discard all but $\mathcal{O}(\Delta^3)$ vertices from the graph H. By also deleting the corresponding vertices from the treedepth decomposition of G, it can be observed that the resulting structure can be modified to obtain an *unweighted* treedepth decomposition of H, of depth $\mathcal{O}(n^{1-1/d})$. Then, we can appeal to a $2^{\mathcal{O}(\mathtt{td})} \cdot n^{\mathcal{O}(1)}$ time, polynomial space randomized algorithm by [15] for Cycle Cover that is based on Cut&Count. Thus, we get the following result. We give the formal details in the full version.

▶ **Theorem 18.** *There exists a $2^{\mathcal{O}(n^{1-1/d})}$ time, polynomial space, randomized algorithm, to solve Cycle Cover in the intersection graphs of similarly sized fat objects in $\mathbb{R}^d$. In particular, this implies analogous results for Hamiltonian Cycle and Hamiltonian Path in the intersection graphs of similarly sized fat objects in $\mathbb{R}^d$.*

5 Conclusion and Open Questions

In this paper, following de Berg et al. [4], we consider various graph problems in the intersection graphs of similarly sized fat objects. Our running times for Independent Set, r-Dominating Set, Steiner Tree, Connected Vertex Cover, Feedback Vertex Cover, (Connected) Odd Cycle Transversal, Hamiltonian Cycle are of the form $2^{\mathcal{O}(n^{1-1/d})}$ – matching that in [4] – but we improve the space requirement to be polynomial. Due to some technical reasons, we are not able to achieve a similar result for Connected Dominating Set which is also considered by [4]. We leave this as an open problem.

Kisfaludi-Bak [12] used some of the ideas from [4] in the context of (noisy) unit ball graphs in d-dimensional hyperbolic space. In particular, he gave subexponential and quasi-polynomial time (and space) algorithms for problems such as Independent Set, Steiner Tree, Hamiltonian Cycle using a notion similar to the weighted treedepth. Using our techniques, it should be possible to improve the space requirement of these algorithms to polynomial, while keeping the running time same (up to possibly a multiplicative $\mathcal{O}(\log n)$ factor in the exponent in some cases). Very recently, [5] designed clique-based separators for various geometric intersection graphs that are of sublinear weight. Again, it should be possible to obtain subexponential time, polynomial space for these graph classes. We leave the details of these extensions for a future version.

Finally, our algorithms for the connectivity problems such as Steiner Tree, Connected Vertex Cover, (Connected) Odd Cycle Transversal, and that for Cycle Cover use an adapted version of the Cut&Count technique ([15, 10, 3]). Cut&Count technique crucially uses the Isolation Lemma (cf. Lemma 8), and hence these algorithms are inherently randomized. We note that recently there has been some progress toward derandomizing Cut&Count [16] for problems such as Hamiltonian Cycle on graphs of bounded treedepth. This may also have some consequences for our algorithms.

References

1 Steven Chaplick, Fedor V. Fomin, Petr A. Golovach, Dusan Knop, and Peter Zeman. Kernelization of graph hamiltonicity: Proper h-graphs. In *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 296–310. Springer, 2019. doi:10.1007/978-3-030-24766-9_22.

2 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

3 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Joham MM van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 150–159. IEEE, 2011.

4 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. A framework for exponential-time-hypothesis-tight algorithms and lower bounds in geometric intersection graphs. *SIAM J. Comput.*, 49(6):1291–1331, 2020. doi:10.1137/20M1320870.

5 Mark de Berg, Sándor Kisfaludi-Bak, Morteza Monemizadeh, and Leonidas Theocharous. Clique-based separators for geometric intersection graphs. *CoRR*, abs/2109.09874, 2021. arXiv:2109.09874.

6 Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs. *Journal of the ACM*, 52(6):866–893, 2005.

7 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

8 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Finding, hitting and packing cycles in subexponential time on unit disk graphs. *Discret. Comput. Geom.*, 62(4):879–911, 2019. doi:10.1007/s00454-018-00054-x.

9 Martin Fürer and Huiwen Yu. Space saving by dynamic algebraization based on tree-depth. *Theory Comput. Syst.*, 61(2):283–304, 2017. doi:10.1007/s00224-017-9751-3.

10 Falko Hegerfeld and Stefan Kratsch. Solving connectivity problems parameterized by treedepth in single-exponential time and polynomial space. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.29.

11 Hiro Ito and Masakazu Kadoshita. Tractability and intractability of problems on unit disk graphs parameterized by domain area. In *Proceedings of the 9th International Symposium on Operations Research and Its Applications (ISORA)*, volume 2010. Citeseer, 2010.

12 Sándor Kisfaludi-Bak. Hyperbolic intersection graphs and (quasi)-polynomial time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1621–1638. SIAM, 2020.

13 Donald Ervin Knuth. *The art of computer programming: Generating all combinations and partitions*. Addison-Wesley, 2005.

14 Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 345–354, 1987.

15 Jesper Nederlof, Michal Pilipczuk, Céline M. F. Swennenhuis, and Karol Wegrzycki. Hamiltonian cycle parameterized by treedepth in single exponential time and polynomial space. In *46th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 12301 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2020. doi:10.1007/978-3-030-60440-0_3.

16 Jesper Nederlof, Michal Pilipczuk, Céline M. F. Swennenhuis, and Karol Wegrzycki. Isolation schemes for problems on decomposable graphs. *CoRR*, abs/2105.01465, 2021. arXiv:2105.01465.

17 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

18 Bruce A Reed. Algorithmic aspects of tree width. In *Recent advances in algorithms and combinatorics*, pages 85–107. Springer, 2003.

On Fair and Efficient Allocations of Indivisible Public Goods

Jugal Garg
University of Illinois, Urbana-Champaign, IL, USA

Pooja Kulkarni
University of Illinois, Urbana-Champaign, IL, USA

Aniket Murhekar
University of Illinois, Urbana-Champaign, IL, USA

Abstract

We study fair allocation of indivisible public goods subject to cardinality (budget) constraints. In this model, we have n agents and m available public goods, and we want to select $k \leq m$ goods in a fair and efficient manner. We first establish fundamental connections between the models of private goods, public goods, and public decision making by presenting polynomial-time reductions for the popular solution concepts of maximum Nash welfare (MNW) and leximin. These mechanisms are known to provide remarkable fairness and efficiency guarantees in private goods and public decision making settings. We show that they retain these desirable properties even in the public goods case. We prove that MNW allocations provide fairness guarantees of Proportionality up to one good (Prop1), $1/n$ approximation to Round Robin Share (RRS), and the efficiency guarantee of Pareto Optimality (PO). Further, we show that the problems of finding MNW or leximin-optimal allocations are NP-hard, even in the case of constantly many agents, or binary valuations. This is in sharp contrast to the private goods setting that admits polynomial-time algorithms under binary valuations. We also design pseudo-polynomial time algorithms for computing an exact MNW or leximin-optimal allocation for the cases of (i) constantly many agents, and (ii) constantly many goods with additive valuations. We also present an $O(n)$-factor approximation algorithm for MNW which also satisfies RRS, Prop1, and 1/2-Prop.

2012 ACM Subject Classification Theory of computation → Mathematical optimization

Keywords and phrases Public goods, Nash welfare, Leximin, Proportionality

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.22

Related Version *Full Version*: https://arxiv.org/abs/2107.09871 [23]

Funding Supported by NSF Grant CCF-1942321 (CAREER).

1 Introduction

The problem of fair division was formally introduced by Steinhaus [32], and has since been extensively studied in economics and computer science [10, 28]. Recent work has focused on the problem of fair and efficient allocation of indivisible *private* goods. We label this setting as the PrivateGoods model. Here, goods have to be partitioned among agents, and a good provides utility only to the agent who owns it. However, goods are not always private, and may provide utility to multiple agents simultaneously, e.g., books in a public library. The fair and efficient allocation of such *indivisible public goods* is an important problem.

In this paper we study the setting of PublicGoods, where a set of n agents have to select a set of at most k goods from a set of m given goods. This simple cardinality constraint models several real world scenarios. While previous work has largely focused on the $k < n$ case, e.g., for voting and committee selection [2, 13], there is much less work available for the case of $k \geq n$. This setting is important in its own right. We present a few compelling examples.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 22; pp. 22:1–22:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Example 1.** A *public library* wants to buy k books that adhere to preferences of n people who might use the library. Clearly, the number of books has to be much greater than the number of people using the library, hence $k \gg n$.

▶ **Example 2.** A family (or a group of friends) of size n wants to decide on a list of k movies to watch together for a few months. Here too, $k > n$. Another example of the same flavor is a committee tasked with inviting speakers at a year-long weekly seminar.

▶ **Example 3.** Another important example is that of *diverse search results* for a query. Given a query (say of "computer scientist images") on a database, we would like to output k search results which reflect diversity in terms of n specified features (like "gender, race and nationality"). Once again, $k \geq n$.

A related setting PublicDecisions of public decision making [15] models the scenario in which n agents are faced with m issues with multiple alternatives per issue, and they must arrive at a decision on each issue. Conitzer et al. [15] showed that this model subsumes the PrivateGoods setting.

Connections between the models. A central question motivating this work is:

▶ **Question 1.** *Can we establish fundamental connections between the three models* PrivateGoods*,* PublicGoods*, and* PublicDecisions*?*

To answer this question, we first describe two well-studied solution concepts for allocating goods in the PrivateGoods and PublicDecisions models, namely the *maximum Nash welfare* (MNW) and *leximin* mechanisms. These mechanisms have been shown to produce allocations that are fair and efficient in the models of PrivateGoods and PublicDecisions. The MNW mechanism returns an allocation that maximizes the geometric mean of agents' utilities, and the leximin mechanism returns an allocation that maximizes the minimum utility, and subject to this, maximizes the second minimum utility, and so on. We label the problems of computing the Nash welfare maximizing (resp. leximin optimal) allocation in the three models as PrivateMNW, PublicMNW, DecisionMNW (resp. PrivateLex, PublicLex, DecisionLex).

We answer Question 1 positively by presenting novel polynomial-time reductions from the model of PrivateGoods to PublicGoods, and from PublicGoods to PublicDecisions for the problem of computing a Nash welfare maximizing allocation.

$$\boxed{\mathsf{PrivateMNW} \leq \mathsf{PublicMNW} \leq \mathsf{DecisionMNW}} \tag{1}$$

More notably, these reductions also work for the MNW problem when restricted to binary valuations. Apart from establishing fundamental connections between these models, our reductions also determine the complexity of the MNW problem, as we detail below. We also develop similar reductions between the models for the leximin mechanism, showing:

$$\boxed{\mathsf{PrivateLex} \leq \mathsf{PublicLex} \leq \mathsf{DecisionLex}} \tag{2}$$

Fairness and efficiency considerations. We next describe the fairness and efficiency properties that the MNW and leximin mechanisms have been shown to satisfy in the PrivateGoods and PublicDecisions models.

The standard notion of economic efficiency is Pareto-optimality (PO). An allocation is said to be PO if no other allocation makes an agent better off without making anyone worse off. The classical fairness notion of *proportionality* requires that every agent gets her *proportional value*, i.e., $1/n$-fraction of the maximum value she can obtain in any allocation. However, proportional allocations are not guaranteed to exist.[1] Hence, we study the notion of Proportionality up to one good (Prop1) for PublicGoods. We say an allocation is Prop1 if for every agent i who does not get her proportional value, i gets her proportional value after swapping some unselected good with a selected one. For PrivateGoods and PublicDecisions, Prop1 is defined similarly – in the former, an agent is given an additional good [6, 27]; and in the latter, an agent is allowed to change the decision on a single issue [15]. While Prop1 is an individual fairness notion, it is still important for allocating public goods. For instance, in Example 1, we want allocations in which every agent has some books that cater to her taste, even if her taste differs from the rest of the agents. Likewise, in Example 2, a fair selection of movies must ensure that there are some movies every member can enjoy.

We also consider the fairness notion of Round-Robin Share (RRS) [15], which demands that each agent i receives at least the utility which she would get if agents were allowed to pick goods in a round-robin fashion, with i picking last.

In the PrivateGoods and PublicDecisions models, an MNW allocation satisfies Prop1 in conjunction with PO [11, 15]. Similarly in both these models, the leximin-optimal allocation satisfies RRS and PO [15]. It is therefore natural to ask:

▶ **Question 2.** *What guarantee of fairness and efficiency do the MNW and leximin mechanisms provide in the* PublicGoods *model?*

Answering this question, we show that an MNW allocation satisfies Prop1, $1/n$-approximation to RRS, and is PO. Further, for all agents, a leximin-optimal allocation satisfies RRS[2], Prop1 and PO.

Complexity of computing MNW and leximin-optimal allocations. Given the desirable fairness and efficiency properties of these mechanisms, we investigate the complexity of computing MNW and leximin-optimal allocations in the PublicGoods model. It is known that PrivateMNW is APX-hard [26, 21] (hard to approximate) and DecisionMNW [15] is NP-hard. Likewise, PrivateLex too is NP-hard [9]. Therefore, we ask:

▶ **Question 3.** *What is the complexity of* PublicMNW *and* PublicLex*?*

Since PrivateMNW and PrivateLex are known to be NP-hard, our reductions (1) and (2) immediately show that PublicMNW and PublicLex are NP-hard. However, we show stronger results that PublicMNW and PublicLex remain NP-hard even when the valuations are binary. These results are in stark contrast to the PrivateGoods case, which admits polynomial-time algorithms for binary valuations [16, 20]. Further, our reductions between PublicGoods and PublicDecisions also directly enable us to show NP-hardness of DecisionMNW and DecisionLex. Note that the hardness of these problems is known through the connection between PrivateMNW (PrivateLex) and DecisionMNW (DecisionLex) [15]. However, a feature of our reductions (Observation 8) enables us to shows that DecisionMNW is NP-hard even for binary valuations, highlighting the utility of our reductions. We also show that PublicMNW and PublicLex remain NP-hard even when there are only two agents. We note that for the case

[1] Consider for example, two agents A and B and six public goods $\{g_1, g_2, g_3, g_4, g_5, g_6\}$. Agent A has value 1 for g_1, g_2, g_3 and B has value 1 for g_4, g_5, g_6. All other valuations are 0. Suppose we want to select three of these goods. The proportional share of both agents is 1.5. However, in any allocation, the value of at least one agent is at most 1, implying that proportional allocations need not exist.

[2] Note that here we assume we scale the valuations so that RRS $= 1$ for every agent.

Table 1 Complexity of computing MNW and leximin-optimal allocations.

Problem	PrivateGoods	PublicGoods	PublicDecisions
MNW $\{0,1\}$ valuations	P [8, 16]	NP-hard (Theorem 16)	NP-hard (Corollary 23)
Leximin $\{0,1\}$ valuations	P [8, 16]	NP-hard (Theorem 21)	?
MNW two agents	NP-hard	NP-hard (Theorem 20)	?
Leximin two agents	NP-hard	NP-hard (Theorem 22)	?

of two agents, the NP-hardness of PrivateMNW and PrivateLex does not imply NP-hardness of PublicMNW and PublicLex because our reductions between the models do not preserve the number of agents.

We summarize our results in Table 1.

In light of the above computational hardness, we turn to approximation algorithms and exact algorithms for special cases. We design a polynomial-time algorithm that returns an allocation which approximates the MNW to a $O(n)$-factor when $k \geq n$, and is also Prop1 and satisfies RRS.

Finally, we obtain pseudo-polynomial time algorithms for computing MNW and leximin-optimal allocations for constant n. These are essentially tight in light of the NP-hardness for constant n. In interest of space, we skip some proofs from this version. All these proofs can be found in full version of the paper [23].

1.1 Other related work

Maximum Nash welfare. The problem of approximating maximum Nash welfare for private goods is well-studied, see e.g., [14, 7, 12, 22]. [18] showed that the MNW problem is NP-hard for allocating public goods subject to matroid or packing constraints. It has also been studied in the context of voting, or multi-winner elections [1]. Fluschnik et al. [19] studied the fair multi-agent knapsack problem, wherein each good has an associated budget, and a set of goods is to be selected subject to a budget constraint. In this context, they studied the objective of maximizing the geometric mean of $(1 + u_i)$ where u_i is the utility of the i^{th} agent. They showed that maximizing this objective is NP-hard, even for binary valuations or constantly many agents with equal budgets and presented a pseudo-polynomial time algorithm for constant n.

Leximin. Leximin was developed as a fairness notion in itself [30]. Plaut and Roughgarden [29] showed that for private goods, leximin can be used to construct allocations that are envy-free up to any good. Freeman et al. [20] showed that in the PrivateGoods model the MNW and leximin-optimal allocations coincide when valuations are binary.

Core. Core is a strong property that enforces both PO and proportionality-like fairness guarantees for all subsets of agents. It is well-studied in many settings, including game theory and computer science [31, 25]. The core of indivisible public goods might be empty. Fain et al. [18] proved that under matroid constraints, a 2-additive approximation to core exists. On an individual fairness level, 1-additive core is weaker than Prop1 [18].

Participatory Budgeting. The participatory budgeting problem [3, 4] consists of a set of n agents (or voters), and a set of k projects that require funds, a total available budget, and the preferences of the voters over the projects. The problem is to allocate the budget in a fair and efficient manner. Here typically $k \ll n$. Fain et al. [17] showed that the fractional core outcome is polynomial-time computable. This could be modeled as a public goods problem with goods as the projects.

Voting and Committee Selection. These settings involve selecting a set of k members from a set of m candidates based on the preferences of n agents. Usually, here $k \ll n$ and the fairness notions studied are group fairness like Justified Representation [2], and a core-like notion called *stability* [13].

2 Notation and Preliminaries

Problem setting. For $t \in \mathbb{N}$, let $[t]$ denote $\{1, \ldots, t\}$. An instance of the PublicGoods allocation problem is given by a tuple $\mathcal{I} = (\mathcal{A}, \mathcal{G}, k, \{v_i\}_{i\in\mathcal{A}})$ of a set $\mathcal{A} = [n]$ of $n \in \mathbb{N}$ agents, a set $\mathcal{G} = [m]$ of $m \in \mathbb{N}$ public goods, an integer $0 \le k \le m$, and a set of valuation functions $\{v_i\}_{i\in\mathcal{A}}$, one per agent, where each $v_i : 2^{\mathcal{G}} \to \mathbb{Z}_{\ge 0}$. Unless specified, we assume that $k \ge n$. For a subset of goods $S \subseteq \mathcal{G}$, $v_i(S)$ denotes the utility agent i derives from the goods in S. Unless specified, we assume the valuations are *additive*. In this case, each v_i is specified by m non-negative integers $\{v_{ij}\}_{j\in\mathcal{G}}$, where v_{ij} denotes the value of agent i for good j. Then for $S \subseteq \mathcal{G}$, $v_i(S) = \sum_{j\in S} v_{ij}$. We assume without loss of generality that for every agent i, there is at least one good j with $v_{ij} > 0$. For brevity, we write $v_i(g_1, \ldots, g_r)$ in place of $v_i(\{g_1, \ldots, g_r\})$ for a set $\{g_1, \ldots, g_r\} \subseteq \mathcal{G}$. An *allocation* is a subset $\mathbf{x} \subseteq \mathcal{G}$ of goods which satisfies the cardinality constraint $|\mathbf{x}| \le k$.

Nash welfare. The Nash welfare (NW) of an allocation $\mathbf{x}$ is given by $\mathsf{NW}(\mathbf{x}) = (\prod_{i\in\mathcal{A}} v_i(\mathbf{x}))^{1/n}$. An allocation with the maximum NW is called an MNW allocation or a Nash optimal allocation.[3] We also refer to the product of the agents' utilities as the *Nash product*. An allocation $\mathbf{x}$ *approximates* MNW to a factor of α if $\mathsf{NW}(\mathbf{x}) \ge \alpha \cdot \mathsf{NW}(\mathbf{x}^*)$, where $\mathbf{x}^*$ is an MNW allocation.

Leximin. Given an allocation $\mathbf{x}$, let $\hat{\mathbf{x}}$ denote the vector of agent's utilities under $\mathbf{x}$, sorted in non-decreasing order. For two allocations $\mathbf{x}, \mathbf{y}$, we say $\mathbf{x}$ *leximin-dominates* $\mathbf{y}$ if there exists $i \in [n]$ such that $\hat{\mathbf{x}}_i > \hat{\mathbf{y}}_i$ and $\forall j < i, \hat{\mathbf{x}}_j = \hat{\mathbf{y}}_j$. An allocation is leximin-optimal if no other allocation leximin-dominates it.

Fairness notions. We now discuss fairness notions for the PublicGoods setting. The *proportional share* of an agent i, denoted by Prop_i is a $1/n$-share of the maximum value she can obtain from any allocation. Formally:

$$\mathsf{Prop}_i = \frac{1}{n} \cdot \max_{\mathbf{x}\subseteq\mathcal{G}, |\mathbf{x}|\le k} v_i(\mathbf{x}).$$

The round-robin share of agent i, denoted by RRS_i, is the minimum value an agent can be guaranteed if the agents pick k goods in a round-robin fashion, with i picking last. Therefore, this value equals the maximum value of any $\lfloor k/n \rfloor$ sized subset. Formally:

$$\mathsf{RRS}_i = \max_{\mathbf{x}\subseteq\mathcal{G}, |\mathbf{x}|\le\lfloor k/n\rfloor} v_i(\mathbf{x}).$$

[3] If the NW is 0 for all allocations, MNW allocations are defined as those which give non-zero utility to maximum number of agents, and then maximize the product of utilities for those agents. Note if $k \ge n$, every agent positively values at least one good and thus MNW > 0.

For $\alpha \in (0, 1]$, an allocation $\mathbf{x}$ is said to satisfy:

1. α-Proportionality (α-Prop) if $\forall i \in \mathcal{A}$, $v_i(\mathbf{x}) \geq \alpha \mathsf{Prop}_i$;
2. α-Proportionality up to one good (α-Prop1) if $\forall i \in \mathcal{A}$, $\exists g \in \mathbf{x}, g' \in \mathcal{G}$, such that $v_i((\mathbf{x} \setminus g) \cup g') \geq \alpha \mathsf{Prop}_i$,
3. α-RRS if for all agents $i \in \mathcal{A}$, $v_i(\mathbf{x}) \geq \alpha \mathsf{RRS}_i$.

Due to the cardinality constraints in the PublicGoods model, the notion of Prop1 requires that for every agent, there is a way to swap one preferred unpicked good with one picked good, after which the agent gets her proportional share. Since Prop1 in PrivateGoods requires only giving an extra good, this makes the definition of Prop1 in PublicGoods slightly more demanding than that in PrivateGoods.

Pareto-optimality. An allocation $\mathbf{y}$ is said to Pareto-dominate an allocation $\mathbf{x}$ if for all agents $i \in \mathcal{A}$, $v_i(\mathbf{y}) \geq v_i(\mathbf{x})$, with at least one of the inequalities being strict. We say $\mathbf{x}$ is Pareto-optimal (PO) if there is no allocation that Pareto-dominates $\mathbf{x}$.

Related models.

1. PrivateGoods. The classic problem of *private goods allocation* concerns partitioning a set of goods $\mathcal{G}$ among the set $\mathcal{A}$ of agents. Thus, a feasible allocation $\mathbf{x}$ is an n-partition $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of $\mathcal{G}$, where agent i is allotted $\mathbf{x}_i \subseteq \mathcal{G}$, and derives utility $v_i(\mathbf{x}_i)$ only from $\mathbf{x}_i$.
2. PublicDecisions. In this model, a set $\mathcal{A}$ of agents are required to make *decisions* on a set $\mathcal{G}$ of issues. Each issue $j \in \mathcal{G}$ has a set $\mathcal{G}_j$ of k_j alternatives, given by $\mathcal{G}_j := \{(j, 1), (j, 2), \ldots, (j, k_j)\}$. A feasible allocation or outcome $\mathbf{x} = (x_1, \ldots, x_m)$ comprises of m decisions, where $x_j \in [k_j]$ is the decision made on issue j. Assuming the valuations are additive, each agent has a value $v_i(j, \ell)$ for the ℓ^{th} alternative of issue j. The valuation of the agent for the outcome $\mathbf{x}$ is then $v_i(\mathbf{x}) = \sum_{j \in \mathcal{G}} v_i(j, x_j)$.

3 Relating the models

We first show rigorous mathematical connections between the PrivateGoods, PublicGoods and PublicDecisions models w.r.t. computing optimal MNW and leximin allocations.

▶ **Theorem 4.** PublicMNW *polynomial-time reduces to* DecisionMNW.

Proof. Let $\mathcal{I} = (\mathcal{A}, \mathcal{G}, k, \{v_i\}_{i \in \mathcal{A}})$ be an instance of the PublicGoods model. For $k = m$, the MNW problem is trivial, since we can select all the m goods. For $n \leq k < m$, we can construct an instance $\mathcal{I}' = (\mathcal{A}', \mathcal{G}', \{\mathcal{G}_j\}_{j \in \mathcal{G}'} \{v'_i\}_{i \in \mathcal{A}'})$ of PublicDecisions from $\mathcal{I}$ in polynomial time, such that given an MNW allocation of $\mathcal{I}'$, we can compute an MNW allocation of $\mathcal{I}$ in polynomial time. Let $V = \max_{i,j} v_{ij}$. We create m public issues: corresponding to each good $j \in \mathcal{G}$, we create an issue j with two alternatives $(j, 1)$ and $(j, 2)$. That is, $\mathcal{G}' = [m]$, and $\mathcal{G}_j = \{(j, 1), (j, 2)\}$ for $j \in \mathcal{G}'$. We create $\mathcal{A}' = [n + mT]$, where $T = \lceil 2mn \log mV \rceil$. The first n agents here correspond to the n agents in $\mathcal{I}$. The last mT agents are of two types: kT agents $\{n + 1, \ldots, n + kT\}$ of type A, and $(m - k)T$ agents $\{n + kT + 1, \ldots, n + mT\}$ of type B. The valuations are as follows: each agent $i \in [n]$ values alternative “1” of the issue $j \in \mathcal{G}'$ at v_{ij}, the agents of type A value only alternative “1”, agents of type B value only alternative “2”. Formally, for $i \in \mathcal{A}'$, and an alternative (j, c) of the issue $j \in \mathcal{G}'$, where $c \in \{1, 2\}$:

$$v'_i(j,c) = \begin{cases} v_{ij}, & \text{if } c = 1 \text{ and } i \in [n]; \\ 1, & \text{if } n < i \leq n + kT \text{ and } c = 1; \\ 1, & \text{if } n + kT < i \leq n + mT \text{ and } c = 2; \\ 0, & \text{otherwise.} \end{cases}$$

Let $\mathbf{x}'$ be an allocation for the instance $\mathcal{I}'$. For $c \in \{1,2\}$, let S_c be the set of issues j with decision c in $\mathbf{x}'$. That is, $S_c = \{j \in [m] : \mathbf{x}'_j = c\}$. Let $k' = |S_1|$. Then we have:

$$\mathsf{NW}(\mathbf{x}') = \left(\prod_{i\in[n]} v'_i(\mathbf{x}') \cdot (k')^{kT} \cdot (m-k')^{(m-k)T} \right)^{\frac{1}{n+mT}}.$$

We now relate $\mathbf{x}'$ to the PublicGoods instance $\mathcal{I}$. The decision $(j,1)$ corresponds to selecting the public good j. Let $\mathbf{x} = S_1 \subseteq \mathcal{G}$ be the corresponding set of public goods. Then for any $i \in [n]$ we have that $v_i(\mathbf{x}) = v'_i(\mathbf{x}')$, since $v'_i(j,2) = 0$ for every $j \in [m]$. Thus:

$$\mathsf{NW}(\mathbf{x}') = \left(\mathsf{NW}(\mathbf{x})^n \cdot (k')^{kT} \cdot (m-k')^{(m-k)T}\right)^{\frac{1}{n+mT}}. \tag{3}$$

We now have to prove that $\mathbf{x}$ satisfies $|\mathbf{x}| \leq k$. Let W_ℓ be the Nash product of any MNW allocation for the PublicGoods instance $\mathcal{I}_\ell = (\mathcal{A}, \mathcal{G}, \ell, \{v_i\}_{i\in\mathcal{A}})$, $0 \leq \ell \leq m$. Clearly, $0 = W_0 \leq W_1 \leq \ldots W_m \leq (mV)^n$. As $k \geq n$, $W_k \geq 1$, since we assume every agent has at least one good that she values positively. Define $g : [m] \to \mathbb{Z}$, as $g(a) = a^k(m-a)^{m-k}$. Then if $\mathbf{x}'$ is an MNW allocation for $\mathcal{I}'$, (3) becomes:

$$\mathsf{NW}(\mathbf{x}') = (W_{k'} \cdot g(k')^T)^{1/(n+mT)}. \tag{4}$$

Let G_1 and G_2 denote the largest and second-largest values that g attains over its domain. We observe that g increases in $[0,k]$, and decreases in $[k,m]$. Hence, $G_1 = g(k)$ implying:

$$G_1 = k^k(m-k)^{m-k}; G_2 = \max(g(k-1), g(k+1)).$$

We now claim the following and prove it in Appendix A:

▷ Claim 5. $G_1^T > W_m \cdot G_2^T$.

Using Claim 5, we have for all $k' \in [m] \setminus \{k\}$:

$$W_k \cdot g(k)^T \geq G_1^T > W_m \cdot G_2^T \geq W_{k'} \cdot g(k')^T,$$

Hence, the quantity $W_{k'} \cdot g(k')^T$ is maximized when $k' = k$. Recalling (4), we conclude that for the MNW allocation $\mathbf{x}'$ of $\mathcal{I}'$, the corresponding set $\mathbf{x}$ has cardinality exactly k. Further $\mathbf{x}$ also maximizes the NW among all allocations of the instance $\mathcal{I}$ satisfying this cardinality constraint. Thus, $\mathbf{x}$ in fact is an MNW allocation for $\mathcal{I}$. Finally, it is clear that this is a polynomial time reduction. ◀

We next relate the MNW problem in the PrivateGoods model with the PublicGoods model.

▶ **Theorem 6.** PrivateMNW *polynomial-time reduces to* PublicMNW.

Proof. Let $\mathcal{I} = (\mathcal{A} = [n], \mathcal{G} = [m], V)$ be a PrivateGoods instance, using which we create a PublicGoods instance I' as follows. We create $n + 2m$ agents, i.e. $\mathcal{A}' = [n + 2m]$. The first n agents correspond to the n agents in $\mathcal{I}$. The last $2m$ are dummy agents. We create $n \cdot m$ public goods: for each good $j \in [m]$, we create a set of n copies $S_j = \{j_1, j_2, \ldots, j_n\}$, $\mathcal{G}' = \bigcup_{j \in \mathcal{G}} S_j$. We set $k = m$. The valuations for $i \in \mathcal{A}'$, $j_\ell \in \mathcal{G}'$ are:

$$v'_i(j_\ell) = \begin{cases} v_{ij}, \text{ if } i = \ell \text{ and } i \in [n]; \\ 1, \text{ if } i \in \{n + 2j - 1, n + 2j\}; \\ 0, \text{ otherwise}, \end{cases}$$

i.e. each agent $i \in [n]$ values exactly one copy, j_i for each $j \in \mathcal{G}$ at v_{ij}, and for each good $j \in \mathcal{G}$, there are exactly two dummy agents who value all copies of j.

We use the following claim in our proof. We prove it in Appendix A.

▷ Claim 7. Any MNW allocation $\mathbf{x}'$ of $\mathcal{I}'$ does not select two goods from same $S_j, j \in [m]$.

Consider any MNW allocation $\mathbf{x}'$ of $\mathcal{I}'$. We construct a partition, $\mathbf{x}$ of goods for $\mathcal{I}$ from this in the following way. For $i \in [n]$, $j \in [m]$, define $x_{ij} = 1$ if $j_i \in \mathbf{x}'$, and 0 otherwise. Let $\mathbf{x}_i = \{j \in \mathcal{G} : x_{ij} = 1\}$. Thus, the value that agent i gets in $\mathbf{x}$ is

$$\begin{aligned} v_i(\mathbf{x}_i) = \sum_{j \in \mathcal{G}} v_{ij} x_{ij} &= \sum_{j \in \mathcal{G}} v_{ij} \mathbf{1}(j_i \in \mathbf{x}'), \\ &= \sum_{j \in \mathcal{G}} v'_i(j_i) \mathbf{1}(j_i \in \mathbf{x}'), \\ &= v'_i(\mathbf{x}'). \end{aligned}$$

Thus, if $m \geq n$, $\mathsf{NW}(\mathbf{x}) = \mathsf{NW}(\mathbf{x}')^{(n+2m)/n}$ and the partition corresponding to $\mathbf{x}'$ as defined above gives an MNW solution for $\mathcal{I}$. On the other hand, if $m < n$, then $\mathbf{x}'$ already gives non-zero value to all dummy agents by Claim 7. Thus, to maximize the total number of agents who get non-zero value, it maximizes the number of agents in $[n]$ who get non-zero value. Call this set S^*. Thus partition $\mathbf{x}$ has maximum number of agents getting a non-zero value. Finally, it maximizes the Nash product over $S^* \cup \{n + 1, \ldots, n + 2m\}$. Claim 7 also implies that all dummy agents get value 1. Thus, $\prod_{i \in S^*} v_i(\mathbf{x}_i) = \prod_{i \in S^*} v_i(\mathbf{x}')$. Thus even in this case the allocation $\mathbf{x}$ corresponds to an MNW allocation in $\mathcal{I}$. ◀

▶ **Observation 8.** *A desirable feature of the above reductions for the MNW problem from instance $\mathcal{I} = (\mathcal{A}, \mathcal{G}, V)$ to $\mathcal{I}' = (\mathcal{A}', \mathcal{G}', V')$ is that $V' = V \cup \{0, 1\}$, i.e., the reduction only creates instances $\mathcal{I}'$ which have 0 and 1 as the only potentially additional values as compared to $\mathcal{I}$. We use this feature in establishing the computational complexity of computing an MNW allocation in the* PublicDecisions *model with binary values, see Corollary 23.*

Similar polynomial-time reductions hold between the three models for the problem of computing a leximin-optimal allocation. We give the theorem statements here and the proofs can be found in full version of the paper.

▶ **Theorem 9.** PublicLex *polynomial-time reduces to* DecisionLex.

▶ **Theorem 10.** PrivateLex *polynomial-time reduces to* PublicLex.

4 Properties of MNW and Leximin

We prove that MNW and leximin-optimal allocations satisfy desirable fairness and efficiency properties in the PublicGoods model as well. First, we show some interesting relations between our three fairness notions – Prop, Prop1, and RRS in the PublicGoods model where $k \geq n$.[4] Our results are presented in Table 2.

Table 2 Relations between the fairness notions for $k \geq n$. Each cell (R, C) contains a factor α s.t. any allocation satisfying the row property R implies an α-approximation to the column property C. Cells with $\alpha = 1$ are marked with ✓, and with $\alpha = 0$ are marked with ✗.

	RRS	Prop	Prop1
RRS	✓	$\frac{n}{2n-1}$ (Lem. 12)	✓(Lem. 11)
Prop	$1/n$ (Lem. 13)	✓	✓
Prop1	✗(Ex. 14)	✗(Ex. 14)	✓

▶ **Lemma 11.** *Any allocation that satisfies* RRS *also satisfies* Prop1.

Proof. Fix any agent i. Let $\mathbf{x} = \{h_1, h_2, \ldots, h_k\}$ be any allocation that satisfies RRS. Let $\mathbf{x}_k^* = \{g_1, g_2, \ldots, g_k\}$ denote the top k goods for agent i. We assume that the goods both in $\mathbf{x}$ and $\mathbf{x}_k^*$ are ordered in decreasing order of valuations according to agent i. Now, suppose that top ℓ goods of $\mathbf{x}$ match with top ℓ goods of $\mathbf{x}_k^*$, i.e. $v_i(h_j) = v_i(g_j), \forall j \leq \ell$ and $v_i(h_{\ell+1} < v_i(g_{\ell+1}))$. Note that since $\mathbf{x}_k^*$ is the top k goods of agent i, we cannot have that $v_i(h_j) > v_(g_j)$ for any $j \leq \ell$. We want to prove that RRS implies Prop1. If $\mathbf{x}$ was already satisfying proportionality, it is obvious that $\mathbf{x}$ is Prop1. If $\ell \geq d$, it is again easy to see that $\mathbf{x}$ is Prop1. This is because, if $k = d$ then we already have top k goods, giving a proportional allocation. If $k > d$, then we can remove any good from $h_{d+1}, \ldots, h_k$ and exchange it with g_{d+1} to ensure proportionality, making the original allocation Prop1. Finally, if n divides k then we have proportionality implied by RRS from Lemma 12.

Thus, we now assume that $\ell < d$, $k = nd + r$ with $r \leq n - 1$ and that $\mathbf{x}$ is not already a proportional allocation. We know that $v(h_1, \ldots, h_\ell) = v(g_1, \ldots, g_\ell)$ and $v(h_1, \ldots, h_k) < \frac{1}{n} v(g_1, g_2, \ldots, g_k)$. Thus,

$$v(h_{\ell+1}, \ldots, h_k) < \frac{1}{n} v(g_{\ell+1}, \ldots, g_k) \tag{5}$$

Now, $v(h_k) \leq \frac{1}{k-\ell} v(h_{\ell+1}, \ldots, h_k)$. Thus,

$$v(h_k) \leq \frac{1}{n \cdot (k - \ell)} v(g_{\ell+1}, \ldots, g_k) \tag{6}$$

Now, consider the good $g_{\ell+1}$. It is the good with highest value that is not in $\mathbf{x}$. We prove that removing h_k and adding $g_{\ell+1}$ gives us an allocation that is proportional. Since $\ell < d$, $v_i(g_{\ell+1}) \geq v_i(g_{nd+j}), \forall j \leq r$. Combining with the fact that $r < n$,

$$(n - 1) \cdot v_i(g_{\ell+1}) \geq v_i(g_{nd+1}, \ldots, g_{nd+r}). \tag{7}$$

[4] Note that when $k < n$, RRS is 0. Any agent who gets 0 value satisfies Prop1 when $k < n$ trivially. Thus, RRS and Prop1 coincide when $k < n$. On the other hand, the proportional value will be non-zero even when $k = 1$ if the agent likes at least one good. Thus, there can be no multiplicative relation between RRS and Prop when $k < n$.

Again since the goods are arranged in decreasing order of valuations, $v_i(g_1,\dots,g_d) \geq v_i(g_{jd+1},\dots,g_{(j+1)d}), \forall 1 \leq j \leq (n-1)$. Thus,

$$(n-1)\cdot v_i(g_1,\dots,g_d) \geq v_i(g_{d+1},\dots,g_{nd}). \tag{8}$$

Define, $LHS = (n-1)v_i(g_{\ell+1}) + (n-1)v_i(g_1,\dots,g_d)$. Combining (7) and (8),

$$\begin{aligned} LHS &\geq v_i(g_{nd+1},\dots,g_{nd+r}) + v_i(g_{d+1},\dots,g_{nd}) \\ &= v_i(g_{d+1},\dots,g_k) \\ &= v_i(g_{\ell+1},\dots,g_k) - v_i(g_{\ell+1},\dots,g_d) \end{aligned}$$

Thus we get,

$$(n-1)v_i(g_{\ell+1}) + (n-1)v_i(g_1,\dots,g_\ell) \geq v_i(g_{\ell+1},\dots,g_k) - nv_i(g_{\ell+1},\dots,g_d)$$

Now, $v_i(g_{\ell+1}) \geq \frac{1}{k-\ell}v_i(g_{\ell+1},\dots,g_k)$. Hence,

$$\begin{aligned} nv_i(g_{\ell+1}) + (n-1)v_i(g_1,\dots,g_\ell) &\geq v_i(g_{\ell+1},\dots,g_k) - nv_i(g_{\ell+1},\dots,g_d) + \frac{1}{k-\ell}v_i(g_{\ell+1},\dots,g_k) \\ &\geq v_i(g_{\ell+1},\dots,g_k) - nv_i(h_{\ell+1},\dots,h_k) + nv_i(h_k), \end{aligned}$$

where the second inequality follows because $\mathbf{x}$ is RRS and from (6). Rearranging the above terms and using the fact that $v_i(g_1,\dots,g_\ell) = v_i(h_1,\dots,h_\ell)$, we get

$$nv_i(g_{\ell+1}) + nv_i(h_1,\dots,h_k) - nv_i(h_k) \geq v_i(g_1,\dots,g_k)$$

which implies that $\mathbf{x}$ is Prop1. ◀

▶ **Lemma 12.** *Any allocation that is α-RRS is also $\alpha\cdot\frac{n}{2n-1}$-Prop. Further, when n divides k, α-RRS implies α-Prop.*

Proof. We will prove a stronger result assuming the valuations are monotone and subadditive.

Let $\mathbf{x}$ denote any subset of k items that satisfies $\alpha\cdot$ RRS. Fix any agent i. We have,

$$v_i(\mathbf{x}) \geq \alpha \cdot \max_{|\mathbf{y}|\leq \lfloor k/n \rfloor} v_i(\mathbf{y}).$$

Let $\mathbf{x}^*$ denote the set of top k goods of agent i. Let $k = n*d+r$ where $r<n$. We can partition $\mathbf{x}^*$ by dividing it into n bundles, each of size $\lfloor k/n \rfloor$ and r more bundles, each of size 1. Note that when $k \geq n$, $\lfloor k/n \rfloor \geq 1$ and $r<n$. Thus, we get at most $2n-1$ bundles each of size at most $\lfloor k/n \rfloor$. We denote these bundles by $S_1, S_2, \dots, S_l$, with $l \leq 2n-1$. Thus, we have,

$$\begin{aligned} v_i(\mathbf{x}^*) &= v_i(\cup_{i\in[l]}S_i), \\ &\leq \sum_{i\in[l]} v_i(S_i), \\ &\leq \sum_{i\in[l]} \frac{1}{\alpha}\cdot v_i(\mathbf{x}), \\ &\leq v_i(\mathbf{x})\cdot\frac{2n-1}{\alpha}. \end{aligned} \tag{9}$$

Here the second inequality follows from subadditivity and third follows because $\mathbf{x}$ is RRS. Thus, we have

$$v_i(\mathbf{x}) \geq \frac{\alpha}{2n-1}v_i(\mathbf{x}^*) = \alpha\cdot\frac{n}{2n-1}\mathsf{Prop}_i.$$

Further, when n divides k, $r = 0$ and we get $l = n$ bundles each of size k/n. Thus, we have from (9)

$$v_i(\mathbf{x}^*) \leq \frac{n}{\alpha} \cdot v_i(\mathbf{x}).$$

Thus,

$$v_i(\mathbf{x}) \geq \frac{\alpha}{n} v_i(\mathbf{x}^*) = \alpha \mathsf{Prop}_i. \qquad \blacktriangleleft$$

▶ **Lemma 13.** *Any allocation that satisfies α-*Prop *gives an α/n multiplicative approximation to* RRS*, and this is tight.*

Proof. Suppose a given allocation, $\mathbf{x}$ satisfies α-Prop. Fix any agent i.

$$\begin{aligned} v_i(\mathbf{x}) &\geq \alpha \cdot \frac{1}{n} \cdot \max_{|\mathbf{y}| \leq k} v_i(\mathbf{y}), \\ &\geq \alpha \cdot \frac{1}{n} \cdot \max_{|\mathbf{y}| \leq \lfloor k/n \rfloor} v_i(\mathbf{y}), \\ &= \frac{\alpha}{n} \cdot \mathsf{RRS}. \end{aligned}$$

For the tightness of lemma, consider the following example: We have $n = 2$ agents and $m = 5$ goods. Agent 1 values goods 1 and 2 at 1 each, does not value goods $3, 4, 5$. Agent 2 values all goods at 1. If $k = 4$, the RRS value of agent 1 is 2. Her proportional value is 1. Thus, picking goods $1, 3, 4, 5$ gives agent 1 her Prop share but only ensures $1/n$ of her RRS share. ◀

Finally, we note in the following example that Prop1 will not give any approximation to either Prop or RRS.

▶ **Example 14** (Prop1 does not approximate Prop or RRS)**.** Finally, we note that a Prop1 allocation might not give an α approximation to RRS for any $\alpha > 0$. Consider an instance of public goods allocation with $n = 2$. We have 3 goods. Agent 1 values goods 1, 2 at value of 1 and values good 3 at 0. Agent 2 values goods 1, 2 at 0 and values good 3 at 1. If we want to select $k = 2$ goods, then, selecting goods 1 and 2 gives agent 2 value 0. This allocation is Prop1, but provides no multiplicative approximation to either RRS or Prop for agent 2.

Next, we show that MNW allocations are fair:

▶ **Lemma 15.** *All MNW allocations satisfy Prop1.*

Proof. Suppose there exists an MNW allocation $\mathbf{x}$ that is not Prop1. This implies for some agent $i \in \mathcal{A}$, for all pairs of goods $j \in \mathbf{x}$ and $j' \notin \mathbf{x}$, $v_i((\mathbf{x} \setminus j) \cup j') < \mathsf{Prop}_i$. If $k < n$, $\mathsf{Prop}_i \leq \max_{j \in \mathcal{G}} v_{ij}$, and swapping any good in $\mathbf{x}$ with this good will give her her proportional share.

Consider now $k \geq n$. Since we assume each agent positively values at least one good, the MNW value is non-zero. Since MNW is scale-invariant, we scale the valuations of agents so that $v_h(\mathbf{x}) = 1 \ \forall h \neq i$. Let g' be the highest-valued good of i not in $\mathbf{x}$, i.e., $g' = \mathsf{argmax}_{j \in \mathcal{G} \setminus \mathbf{x}} v_{ij}$. Let $\mathbf{x}_0 = \{j \in \mathbf{x} : v_{ij} < v_{ig'}\}$ be the set of goods in $\mathbf{x}$ that give i strictly lesser value than g'. Since i does not satisfy Prop1, $\mathbf{x}_0 \neq \emptyset$. Suppose we order the goods in $\mathcal{G}$ according to the valuation of i as $\{g_1, \ldots, g_m\}$, where $v_i(g_r) \geq v_i(g_s)$ for $1 \leq r \leq s \leq m$. Then $n \cdot \mathsf{Prop}_i = v_i(g_1, \ldots, g_k)$ by definition. Since g' is the highest-valued good for i not in $\mathbf{x}$, and further since every good in $\mathbf{x}_0$ is valued at less than $v_{ig'}$ by i, we can bound the total value to i of the top k goods $g_1, \ldots, g_k$ as follows: $v_i(g_1, \ldots, g_k) \leq v_i(\mathbf{x} \setminus \mathbf{x}_0) + |\mathbf{x}_0| v_{ig'}$ which, using additivity of v_i, can alternatively be written as:

$$v_i(\mathbf{x}) + \sum_{j \in \mathbf{x}_0} (v_{ig'} - v_{ij}) \geq n\mathsf{Prop}_i. \tag{10}$$

Consider a good g given by[5]:

$$g \in \mathsf{argmin}_{j \in \mathbf{x}_0} \frac{\sum_{h \in \mathcal{A} \setminus \{i\}} v_{hj}}{v_{ig'} - v_{ij}}.$$

Then by definition of g, we have:

$$\begin{aligned} \frac{\sum_{h \in \mathcal{A} \setminus \{i\}} v_{hg}}{v_{ig'} - v_{ig}} &\leq \frac{\sum_{j \in \mathbf{x}_0} \sum_{h \in \mathcal{A} \setminus \{i\}} v_{hj}}{\sum_{j \in \mathbf{x}_0} v_{ig'} - v_{ij}} \quad \leq \frac{\sum_{h \in \mathcal{A} \setminus \{i\}} \sum_{j \in \mathbf{x}_0} v_{hj}}{n\mathsf{Prop}_i - v_i(\mathbf{x})} \\ &\leq \frac{n-1}{n\mathsf{Prop}_i - v_i(\mathbf{x})}, \end{aligned} \tag{11}$$

where the first transition follows by rearranging terms in the numerator, and using (10) in the denominator, and the final transition follows by recalling that $v_h(\mathbf{x}) = 1$ for all $h \neq i$.

Let $\delta = v_{ig'} - v_{ig}$. We know $v_i(\mathbf{x}) + \delta < \mathsf{Prop}_i$. Substituting this in (11), and noting $\delta > 0$ gives:

$$\frac{\sum_{h \in \mathcal{A} \setminus \{i\}} v_{hg}}{\delta} < \frac{1}{v_i(\mathbf{x}) + \delta}. \tag{12}$$

Let us now consider the allocation $\mathbf{x}' = (\mathbf{x} \setminus g) \cup g'$. We show $\mathsf{NW}(\mathbf{x}') > \mathsf{NW}(\mathbf{x})$, thus contradicting the Nash optimality of $\mathbf{x}$. Since for any $h \neq i$, $v_h(\mathbf{x}') \geq v_h(\mathbf{x}) - v_{hg} = 1 - v_{hg}$, we have:

$$\begin{aligned} \prod_{h \in \mathcal{A}} v_h(\mathbf{x}') \geq v_i(\mathbf{x}') \prod_{h \in \mathcal{A} \setminus \{i\}} (1 - v_{hg}) &\geq (v_i(\mathbf{x}) + \delta)\Bigg(1 - \sum_{h \in \mathcal{A} \setminus \{i\}} v_{hg}\Bigg) \\ &> (v_i(\mathbf{x}) + \delta)\left(1 - \frac{\delta}{v_i(\mathbf{x}) + \delta}\right) = v_i(\mathbf{x}), \end{aligned}$$

where the first transition uses Weierstrass' inequality [24], and the second transition uses (12). This leads to $\mathsf{NW}(\mathbf{x}') > \mathsf{NW}(\mathbf{x})$, giving the desired contradiction. Hence any MNW allocation satisfies Prop1. ◀

Besides Prop1, the MNW allocation satisfies several other desirable properties, as our next result shows.

▶ **Theorem 16.** *All MNW allocations satisfy* PO*,* Prop1*, and* $1/n$-RRS. *Further when $k \geq n$, MNW allocation implies $\frac{1}{2n-1}$-Prop.*

Proof. If any MNW allocation did not satisfy Pareto optimality, then at least one of the agents gets a strictly higher value with values of all other agents not decreasing. Thus, if the MNW value is non-zero, we get an allocation with strictly higher Nash Product, contradicting

[5] [15] considered an *issue* similarly.

the optimality of value of MNW. On the other hand, if MNW value is zero and the strict increase of value holds for one of the agents with non-zero value, then the Nash Product over these agents increases contradicting maximality of Nash Product of these agents. On the other hand, if the strict inequality holds for an agent who receives zero value, the number of agents with non-zero value increases, contradicting the maximality of number of agents who get non-zero value. In both cases, the optimality of MNW is contradicted. Thus any MNW allocation satisfies Pareto Optimality.

Next we prove that all MNW allocations satisfy $1/n$-RRS. Suppose there exists an MNW allocation $\mathbf{x}$ that is not $1/n$-RRS. This implies that for some agent $i \in \mathcal{A}$, $v_i(\mathbf{x}) < \frac{1}{n}\mathsf{RRS}_i$. Let us order the goods according to i's valuation: let $\mathcal{G} = \{g_1, g_2, \ldots, g_m\}$, such that $v_i(g_r) \geq v_i(g_s)$, for all $1 \leq r \leq s \leq m$. Let $p = \lfloor \frac{k}{n} \rfloor$. When $k < n$, $p = 0$, in that case $\mathsf{RRS}_i = 0$. Therefore, $k \geq n$. Observe that the round-robin share of i is given by $\mathsf{RRS}_i = v_i(\{g_1, \ldots, g_p\})$. We scale the valuations of the agents so that for every agent i, $v_i(\mathbf{x}) = 1$. In particular, this implies $\mathsf{RRS}_i > n$.

Let us order the goods in $\mathbf{x}$ according to i's valuation: let $\mathbf{x} = \{j_1, j_2, \ldots, j_k\}$, such that $v_i(j_r) \geq v_i(j_s)$, for all $1 \leq r \leq s \leq k$. Define for $r \in [p]$, $S_r = \{j_{rn-n+1}, \ldots, j_{rn}\}$, and $g'_r = \mathsf{argmin}_{j \in S_r} \sum_{h \in \mathcal{A} \setminus \{i\}} v_{hj}$.

We now construct another allocation $\mathbf{x}'$ as follows. We first check if $g_1 \in \mathbf{x}$. If not, we begin constructing $\mathbf{x}'$ by removing g'_1 from $\mathbf{x}$ and adding g_1. If $g_1 \in \mathbf{x}$, then we proceed to check whether $g_2 \in \mathbf{x}$ or not. For every $r \in [p]$, we remove g'_r and add g_r if g_r is not in $\mathbf{x}$. If g_r is already in $\mathbf{x}$ then for such an r no operation is done. Since we are removing g'_r and $v_i(g'_r) < v_i(g_r) \leq v_i(g_s)$ for all $s < r$, this ensures that $\{g_1, \ldots, g_p\} \subseteq \mathbf{x}'$, which shows $v_i(\mathbf{x}') \geq \mathsf{RRS}_i > n$. Observe that:

$$\begin{aligned}
\sum_{r=1}^{p} \sum_{h \in \mathcal{A} \setminus \{i\}} v_h(g'_r) &\leq \sum_{r=1}^{p} \frac{1}{n} \sum_{h \in \mathcal{A} \setminus \{i\}} \sum_{j \in S_r} v_{hj} && \text{(def. of } g_r\text{)} \\
&\leq \frac{1}{n} \sum_{r=1}^{p} \sum_{j \in S_r} \sum_{h \in \mathcal{A} \setminus \{i\}} v_{hj} && \text{(rearranging)} \\
&\leq \frac{1}{n} \sum_{j \in \mathbf{x}} \sum_{h \in \mathcal{A} \setminus \{i\}} v_{hj} && \text{(def. of } S_r\text{)} \\
&\leq \frac{1}{n} \sum_{h \in \mathcal{A} \setminus \{i\}} v_h(\mathbf{x}^*) && \text{(rearranging)} \\
&= \frac{n-1}{n}.
\end{aligned}$$

Then we have:

$$\begin{aligned}
\mathsf{NW}(\mathbf{x}')^n = \prod_{h \in \mathcal{A}} v_h(\mathbf{x}') \geq v_i(\mathbf{x}') &\prod_{h \in \mathcal{A} \setminus \{i\}} v_h(\mathbf{x}'), \\
\geq v_i(\mathbf{x}') &\prod_{h \in \mathcal{A} \setminus \{i\}} \left(1 - \sum_{r=1}^{p} v_h(g'_r)\right), \\
\geq v_i(\mathbf{x}') &\left(1 - \sum_{r=1}^{p} \sum_{h \in \mathcal{A} \setminus \{i\}} v_h(g_r)\right) > n\left(1 - \frac{n-1}{n}\right) = \mathsf{NW}(\mathbf{x})^n,
\end{aligned}$$

which contradicts the fact that $\mathbf{x}$ is Nash optimal.

Combining this with Lemma (12) and Lemma (15), we get the proof of the theorem. ◀

Similar fairness and efficiency properties for the leximin-optimal allocation. In particular, one can prove the following theorem (proof is in full version of the paper).

▶ **Theorem 17.** *All leximin-optimal allocations are PO, satisfy* RRS *and* Prop1*. Further, when $k \geq n$, a leximin-optimal allocation is also $(n/(2n-1))$-*Prop.

5 Complexity of MNW and Leximin

In this section, we show that PublicMNW and PublicLex are NP-hard. Our hardness results also hold for instances with binary values, which is in stark contrast to the private goods setting, where MNW and leximin-optimal allocations can be computed in polynomial-time. All proofs for this section can be found in the full version of paper. Since the cases of $k \geq n$ and $k < n$ are interesting in their own right, we consider them separately.

▶ **Theorem 18.** *Given a* PublicGoods *allocation instance where $k < n$, computing an α-approximation to MNW is* NP*-hard for any $\alpha > 0$, even when all valuations are binary.*

▶ **Theorem 19.** PublicMNW *is* NP*-hard, even when all valuations are binary.*

Proof. (Sketch) We reduce from the exact regular set packing (ERSP) problem. In the input to ERSP, there are n elements $X = \{x_1, \ldots, x_n\}$, a family of subsets $\mathcal{F} = \{F_1, \ldots, F_m\}$ where each $F_j \subseteq X$ and $|F_j| = d$. The problem is to compute a subfamily $\mathcal{F}' \subseteq \mathcal{F}$, $|\mathcal{F}'| = r$, s.t. for all $F_i \neq F_j \in \mathcal{F}', F_i \cap F_j = \emptyset$. Let $\mathcal{I} = (X, \mathcal{F}, d, r)$ be an instance of ERSP. We construct a PublicGoods instance $\mathcal{I}' = \{\mathcal{A}, \mathcal{G}, k, \{v_i\}_{i \in \mathcal{A}}, T\}$ as follows. We create a set $\mathcal{A} = [n]$ of n agents, a set $\mathcal{G} = \{g_1, \ldots, g_m\} \cup \{d_1, \ldots, d_n\}$ of $m+n$ public goods. For any agent $i \in \mathcal{A}$ and good $g_j \in \mathcal{G}$, $v_i(g_j) = 1$ if $x_i \in F_j$ else 0. For any agent $i \in \mathcal{A}$ and good $d_j \in \mathcal{G}$, $v_i(d_j) = 1$. We set $k = r + n$ and $T = ((n+1)^{dr} n^{n-dr})^{1/n}$. We show that $\mathcal{I}$ is a yes-instance for ERSP iff the MNW for $\mathcal{I}'$ is at least T. ◀

▶ **Theorem 20.** PublicMNW *is* NP*-hard, even for two agents.*

We next show a similar hardness results for computing leximin-optimal allocations, which as we show, apply even for instances with binary values.

▶ **Theorem 21.** PublicLex *is* NP*-hard, even when the valuations are binary.*

▶ **Theorem 22.** PublicLex *is* NP*-hard, even for two agents.*

Proof. (Sketch) We prove this by reducing from the NP-complete problem Monotone c-SAT. In an instance of Monotone c-SAT we have $X = \{x_1, \ldots, x_n\}$ variables, formula, $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ in CNF form with additional constraint that all literals in it are positive. We want to determine if we can satisfy F by setting at most c variables to true. To create $\mathcal{I} = (\mathcal{A}, \mathcal{G}, k, \{v_i\}_{i \in \mathcal{A}})$, corresponding to each clause C_i, we create an agent, i and corresponding to each variable, x_j we create a good, j. Each agent likes the goods corresponding to the variables that show up in her corresponding clause. To ensure that $k \geq |\mathcal{A}|$ in the public goods instance, we create one dummy agent and $m-c+1$ dummy goods. Finally, set $k = m+1$. We show that F has a satisfying assignment with c true variables iff in the Leximin-optimal the minimum utility is $m-c+1$, and the second minimum utility is at least $m-c+2$. ◀

Using the reductions of Theorems 4 and 9 and the NP-hardness results of this section, we obtain NP-hardness results for computing MNW and leximin allocations in the public decision making model. In fact, Observation 8 implies that this NP-hardness remains for the MNW problem even with the valuations are binary.

▶ **Corollary 23.** DecisionMNW *is* NP*-hard, even when all values are binary.*

Using our reductions (Theorems 4 and 9) together with the NP-hardness of PublicMNW and PublicLex (Theorems 19 and 21) implies that:

▶ **Corollary 24.** *The problems* DecisionMNW *and* DecisionLex *are* NP*-hard.*

6 Algorithms for MNW and Leximin

In light of the above computational hardness, we turn to approximation algorithms and exact algorithms for special cases. The proofs of results in this section and the algorithms for special cases can be found in the full version of paper. We first present an algorithm that provides an $O(n)$ factor approximation to MNW and satisfies fairness properties of RRS, Prop1 when valuations $\{v_i\}_{i\in\mathcal{A}}$ are *monotone* ($v_i(S) \le v_i(S \cup g)$ for all $S \subseteq \mathcal{G}$ and $g \in \mathcal{G} \setminus S$) and *subadditive* (for all $S_1 \subseteq \mathcal{G}, S_2 \subseteq \mathcal{G}$, $v_i(S_1) + v_i(S_2) \ge v_i(S_1 \cup S_2)$). The class of subadditive valuations captures complement-free goods, and subsumes additive valuations. Our algorithm assumes access to demand oracles[6] for the subadditive valuations. We use the following subroutine, Maximize, from [5] which takes:

- Input: Set of goods, $\mathcal{G}$, the valuation function v_i of the agent i, and an integer r; and returns:
- Output: $\mathbf{x} \subseteq \mathcal{G}$, s.t. $v_i(\mathbf{x}) \ge \frac{1}{2}\max_{S\subseteq\mathcal{G},|S|\le r} v_i(S)$

Our algorithm, AlgGreedy, has two steps:

- For all $i \in \mathcal{A}$, $\mathbf{x}_i \leftarrow$ Maximize$(\mathcal{G}, v_i, \lfloor\frac{k}{n}\rfloor)$
- Return $\mathbf{x} \leftarrow \cup_{i\in\mathcal{A}}\mathbf{x}_i$

For additive valuations, we assume that Maximize returns a set of $\lfloor k/n\rfloor$ most-preferred goods for each agent. This algorithm enables us to show that:

▶ **Theorem 25.** *There exists a polynomial-time algorithm for the problem of* PublicGoods *allocation (where $k \ge n$ and agents have monotone, subadditive valuations) that returns an allocation which satisfies* RRS*,* $\frac{1}{2}$-Prop*, and approximates the MNW to a factor of $O(n)$. Further, when the valuations are additive, the allocation satisfies* Prop1*.*

We now present pseudo-polynomial time algorithms for two special cases, namely constantly many types of agents, and constantly many types of goods. Our results apply to the more general model of *budget constraints.* We denote an instance of this model by $\mathcal{I} = (\mathcal{A}, \mathcal{G}, B, \{c_j\}_{j\in\mathcal{G}}, \{v_i\}_{i\in\mathcal{A}})$. Each good $j \in \mathcal{G}$ has an associated integral cost c_j, and in a feasible allocation the sum of costs of the picked goods must not exceed the budget B. The MNW and leximin-objectives are defined as before, but over feasible allocations that satisfy the budget constraints. Since cardinality constraints are a special case of budget constraints with uniform cost, our hardness results apply for the budget model also.

[6] Subadditive valuations are set functions and cannot in general represented efficiently. We thus assume access to the functions through some oracles. Given a set of prices p_j for each good $j \in \mathcal{G}$, a demand oracle returns any set S that maximizes $v_i(S) - \sum_{j\in S} p_j$.

Constantly many types of agents

We consider instances where the number of *agent types* is constant. We say agents i and h have the same *type* if $\forall j \in \mathcal{G}$, $v_{ij} = v_{hj}$.

▶ **Theorem 26.** *For a* PublicGoods *allocation instance,* $\mathcal{I} = (\mathcal{A}, \mathcal{G}, B, \{c_j\}_{j\in\mathcal{G}}, \{v_i\}_{i\in\mathcal{A}})$ *with* t *distinct types of agents, (i) an MNW allocation can be computed in time* $O(m \cdot (mV)^t)$*, (ii) a leximin-optimal allocation can be computed in time* $O(m \cdot n \log n \cdot (mV)^t)$*, where* $V = \max_{i\in\mathcal{A}, j\in\mathcal{G}} v_{ij}$*.*

We prove this by presenting a dynamic-programming based algorithm which computes such allocations. We also get:

▶ **Corollary 27.** *For binary valuations, with constantly many types of agents* PublicMNW *and* PublicLex *are polynomial-time solvable.*

Constantly many types of goods

We now consider instances where the number of *types of goods* is constant. We say two goods $j_1, j_2 \in \mathcal{G}$ have same type if for all agents $i \in \mathcal{A}$, $v_{ij_1} = v_{ij_2}$ and $c_{j_1} = c_{j_2}$. In this case, we can enumerate all feasible allocations efficiently, implying that an MNW or leximin-optimal allocation can be computed in polynomial-time.

▶ **Theorem 28.** *For a* PublicGoods *allocation instance* $\mathcal{I} = (\mathcal{A}, \mathcal{G}, B, \{c_j\}_{j\in\mathcal{G}}, \{v_i\}_{i\in[n]})$ *with* t *different types of goods, (i) an MNW, can be computed in time* $O(m^t)$ *(ii) a leximin-optimal allocation can be computed in time* $O(n \log n \cdot m^t)$*.*

7 Discussion

In this paper, we considered the problem of allocating indivisible public goods to agents subject to a cardinality constraint. We showed fundamental connections between the models of private goods, public goods, and public decision making, by presenting polynomial-time reductions for the popular solution concepts of maximum Nash welfare (MNW) and leximin. We also showed that MNW and leximin-optimal allocations satisfy desirable fairness properties like Prop1 and RRS, and the efficiency property of PO. Further we showed that these objectives are computationally NP-hard, including for several special cases like constantly many agents and binary valuations. Lastly, we designed an approximation algorithm for MNW and pseudo-polynomial time algorithms for the case of constantly many agents.

Our work opens up several interesting research directions. Firstly, extending our reductions to the budget model presents a challenging problem. A second question is devising an algorithm to compute a Prop1+PO or RRS+PO allocations in polynomial time, bypassing the hardness of computing MNW or leximin-optimal allocations. Appropriately defining properties like Prop1 in the budget model and investigating whether MNW and leximin satisfy them would be a third interesting research direction. Finally, designing constant-factor approximation algorithms, even for restricted cases like binary valuations, which captures a large class of voting-like scenarios, is another important open problem.

References

1 S Airiau, H Aziz, I Caragiannis, J Kruger, and J Lang. Positional social decision schemes: Fair and efficient portioning. In *Proceedings of the 7th International Workshop on Computational Social Choice (COMSOC)*, 2018.

2 Haris Aziz, Markus Brill, Vincent Conitzer, Edith Elkind, Rupert Freeman, and Toby Walsh. Justified representation in approval-based committee voting. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 784–790, 2015.

3 Haris Aziz, Barton E. Lee, and Nimrod Talmon. Proportionally representative participatory budgeting: Axioms and algorithms. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 23–31, 2018.

4 Haris Aziz and Nisarg Shah. Participatory budgeting: Models and approaches. *arXiv preprint arXiv:2003.00606*, 2020.

5 Ashwinkumar Badanidiyuru, Shahar Dobzinski, and Sigal Oren. Optimization with demand oracles. *Algorithmica*, 81(6):2244–2269, 2019.

6 Siddharth Barman and Sanath Krishnamurthy. On the proximity of markets with integral equilibria. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pages 1748–1755, 2019.

7 Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Finding fair and efficient allocations. In *Proceedings of the 2018 ACM Conference on Economics and Computation (EC)*, pages 557–574, 2018.

8 Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Greedy algorithms for maximizing Nash social welfare. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 7–13, 2018.

9 Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *SIGecom Exch.*, 5(3):11–18, April 2005.

10 S.J. Brams and A.D. Taylor. *Fair Division: From Cake-Cutting to Dispute Resolution.* Cambridge University Press, 1996.

11 Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum Nash welfare. In *Proceedings of the 2016 ACM Conference on Economics and Computation (EC)*, pages 305–322, 2016.

12 Bhaskar Ray Chaudhury, Yun Kuen Cheung, Jugal Garg, Naveen Garg, Martin Hoefer, and Kurt Mehlhorn. On fair division for indivisible items. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 25:1–25:17, 2018.

13 Yu Cheng, Zhihao Jiang, Kamesh Munagala, and Kangning Wang. Group fairness in committee selection. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, pages 263–279, 2019.

14 Richard Cole and Vasilis Gkatzelis. Approximating the Nash social welfare with indivisible items. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing (STOC)*, pages 371–380, 2015.

15 Vincent Conitzer, Rupert Freeman, and Nisarg Shah. Fair public decision making. In *Proceedings of the 2017 ACM Conference on Economics and Computation (EC)*, pages 629–646, 2017.

16 Andreas Darmann and Joachim Schauer. Maximizing Nash product social welfare in allocating indivisible goods. *SSRN Electronic Journal*, 247, January 2014.

17 Brandon Fain, Ashish Goel, and Kamesh Munagala. The core of the participatory budgeting problem. In *Web and Internet Economics (WINE)*, pages 384–399, 2016.

18 Brandon Fain, Kamesh Munagala, and Nisarg Shah. Fair allocation of indivisible public goods. In *Proceedings of the 2018 ACM Conference on Economics and Computation (EC)*, pages 575–592, 2018.

19 Till Fluschnik, Piotr Skowron, Mervin Triphaus, and Kai Wilker. Fair knapsack. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1941–1948, 2019.

20 Rupert Freeman, Sujoy Sikdar, Rohit Vaish, and Lirong Xia. Equitable allocations of indivisible goods. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 280–286, 2019.

21 Jugal Garg, Martin Hoefer, and Kurt Mehlhorn. Satiation in Fisher markets and approximation of Nash social welfare. *CoRR*, abs/1707.04428, 2017.

22 Jugal Garg, Edin Husic, and László A. Végh. Approximating Nash social welfare under Rado valuations. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1412–1425, 2021.

23 Jugal Garg, Pooja Kulkarni, and Aniket Murhekar. On fair and efficient allocations of indivisible public goods. *CoRR*, abs/2107.09871, 2021. arXiv:2107.09871.

24 M. S. Klamkin and D. J. Newman. Extensions of the Weierstrass product inequalities. *Mathematics Magazine*, 43(3):137–141, 1970. URL: http://www.jstor.org/stable/2688388.

25 Mayuresh Kunjir, Brandon Fain, Kamesh Munagala, and Shivnath Babu. ROBUS: Fair cache allocation for data-parallel workloads. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 219–234, 2017.

26 Euiwoong Lee. APX-hardness of maximizing Nash social welfare with indivisible items. *Information Processing Letters*, 122:17–20, 2017.

27 Peter McGlaughlin and Jugal Garg. Improving Nash social welfare approximations. *J. Artif. Intell. Res.*, 68:225–245, 2020.

28 H. Moulin. *Fair Division and Collective Welfare.* Mit Press. MIT Press, 2004.

29 Benjamin Plaut and Tim Roughgarden. Almost envy-freeness with general valuations. *SIAM Journal on Discrete Mathematics*, 34(2):1039–1068, 2020.

30 John Rawls. *A theory of justice.* Harvard university press, 2009.

31 Herbert E Scarf. The core of an N person game. *Econometrica: Journal of the Econometric Society*, pages 50–69, 1967.

32 Hugo Steinhaus. The problem of fair division. *Econometrica*, 16:101–104, 1948.

A Missing Proofs from Section 3

▷ Claim 5. $G_1^T > W_m \cdot G_2^T$.

Proof. Recall that W_ℓ denotes the Nash product of any MNW allocation for the PublicGoods instance $\mathcal{I}_\ell = (\mathcal{A}, \mathcal{G}, \ell, \{v_i\}_{i\in\mathcal{A}})$, for $0 \le \ell \le m$. We have $0 = W_0 \le W_1 \le \dots W_m \le (mV)^n$, and we assume $W_k \ge 1$. Recall that function $g : [m] \to \mathbb{Z}$, was defined as $g(a) = a^k(m-a)^{m-k}$. Let G_1 and G_2 denote the largest and second-largest values that g attains over its domain. We observe that g increases in $[0, k]$, and decreases in $[k, m]$. Hence:

$$G_1 = g(k) = k^k(m-k)^{m-k}.$$
$$G_2 = \max(g(k-1), g(k+1)).$$

Now observe that for $k \in [m] \setminus \{0, 1, m\}$:

$$\begin{aligned}\log g(k) - \log g(k-1) &= k(\log k - \log(k-1)) + (m-k)(\log(m-k) - \log(m-k+1)),\\ &> k \cdot \frac{1}{k-\frac{1}{2}} + (m-k)\cdot\frac{-1}{m-k} \ge \frac{1}{2k-1} \ge \frac{1}{2m},\end{aligned}$$

and for $k \in [m] \setminus \{0, m-1, m\}$:

$$\begin{aligned}\log g(k) - \log g(k+1) &= k(\log k - \log(k+1)) + (m-k)(\log(m-k) - \log(m-k-1)),\\ &> k \cdot \frac{-1}{k} + (m-k)\cdot\frac{1}{m-k-\frac{1}{2}},\\ &\ge \frac{1}{2(m-k)-1} \ge \frac{1}{2m},\end{aligned}$$

using standard properties of logarithms. Thus:

$$\log G_1 - \log G_2 > \frac{1}{2m}.$$

Then we have by recalling that $T = 2mn \log mV$,

$$T(\log G_1 - \log G_2) > 2mn \log mV \cdot \frac{1}{2m} \geq \log W_m,$$

which gives:

$$G_1^T > W_m \cdot G_2^T,$$

as required. Lastly, we consider the cases of $k = 1$ and $k = m - 1$. In both cases, $T(\log G_1 - \log G_2) = T[(m-1)\log(m-1) - \log 2 - (m-1)\log(m-2)] > 2mn \log mV \frac{1}{2m} \geq \log W_m$, which gives $G_1^T > W_m G_2^T$, as claimed. ◁

Proof of Claim 7. Consider first $m \geq n$. Suppose $\exists j \in [m]$ for which two goods $j_i, j_{i'} \in \mathbf{x}', i \neq i'$. Since exactly m goods are picked in $\mathbf{x}'$, there is some $j' \in [m]$, for which no good j'_i is picked in $\mathbf{x}'$ for any $i \in [n]$. This implies that the agents $2j' + n - 1, 2j' + n$ get zero value in $\mathbf{x}'$, making $\mathsf{NW}(\mathbf{x}') = 0$. However, choosing a good from each $j \in [m]$ gives non-zero value to all dummy agents. At the same time, since $m \geq n$, these goods can be chosen so that they give non-zero value to distinct agents in $[n]$. This makes $\mathsf{NW}(\mathbf{x}') \neq 0$ contradicting Nash optimality of $\mathbf{x}'$.

Now, if $m < n$ Nash welfare of all allocations in $\mathcal{I}$ is 0. Thus, the MNW allocation is the one that maximizes the number of agents who get non zero value and then maximizes the product of values for these agents. Consider any allocation $\bar{\mathbf{x}}$, suppose $\exists j \in [m]$ for which two goods $j_i, j_{i'} \in \bar{\mathbf{x}}, i \neq i'$. then again for some j', agents $n + 2j' - 1$ and $n + 2j'$ get value 0 making $\mathsf{NW}(\bar{\mathbf{x}}) = 0$. At the same time, even if $\bar{\mathbf{x}}$ has goods from all different S_j, since $m < n$, and each one item from S_j gives value only to one agent $i \in [n]$, the $\mathsf{NW}(\bar{\mathbf{x}}) = 0$ even in this case. Thus, if $m < n$, all allocations have Nash welfare 0 in $\mathcal{I}'$ also. Suppose the MNW allocation, $\mathbf{x}'$ had two goods from same S_j for some $j \in [m]$. Then, there exists a $j' \in [m]$ such that no good is selected from $S_{j'}$. The two goods from S_j give value to exactly four agents - the two dummy agents $2j + n - 1, 2j + n$ and two agents who receive their copy of good j. Instead, if we exchange one of these goods to a good from $S_{j'}$, we give non-zero value to at least five agents - dummy agents $2j + n - 1, 2j + n, 2j' + n - 1, 2j' + n$ and at least one of the agents in $[n]$. We did not change the value of any other agents in this process. Thus, we increase the number of agents who get non-zero value, contradicting the maximality of $\mathbf{x}'$. Thus, in both cases, all m goods are picked from different $S_j, j \in [m]$. ◁

Time Space Optimal Algorithm for Computing Separators in Bounded Genus Graphs

Chetan Gupta
Aalto University, Finland

Rahul Jain
Fernuniversität in Hagen, Germany

Raghunath Tewari
Indian Institute of Technology Kanpur, India

Abstract

A graph separator is a subset of vertices of a graph whose removal divides the graph into small components. Computing small graph separators for various classes of graphs is an important computational task. In this paper, we present a polynomial-time algorithm that uses $O(g^{1/2}n^{1/2}\log n)$-space to find an $O(g^{1/2}n^{1/2})$-sized separator of a graph having n vertices and embedded on an orientable surface of genus g.

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

Keywords and phrases Graph algorithms, space-bounded algorithms, surface embedded graphs, reachability, Euler genus, algorithmic graph theory, computational complexity theory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.23

Related Version *Extended Version*: `https://arxiv.org/abs/2005.06419`

Funding *Chetan Gupta*: Academy of Finland Grant 321901 and Visvesvaraya PhD Grant.
Raghunath Tewari: DST Inspire Faculty Grant and Visvesvaraya Young Faculty Fellowship.

1 Introduction

Graph separator is a valuable tool in designing divide and conquer based algorithms for various graph problems. In a graph, a separator is a small set of vertices of the graph whose removal divides the graph into pieces such that the size of each piece is at most a fraction of the original graph. Lipton and Tarjan's pioneering result showed that there exists a separator of size $O(n^{1/2})$ in planar graphs [12]. Subsequently, this separator was used to design many algorithms to solve various problems in planar graphs.

Recently, researchers have been interested in designing memory-constrained algorithms for various graph problems. They aim to optimize the space required by the algorithm while maintaining the polynomial time-bound. Graph separators have been used in designing memory-constrained algorithms for the *reachability* problem. Imai et al. and Ashida et al. presented polynomial-time algorithms that use $O(n^{1/2}\log n)$ space to find a separator of size $O(n^{1/2})$ in a planar graph [9, 3]. Imai et al. also gave a memory-constrained algorithm to solve the reachability problem using this separator [9]. A natural extension of planar graphs is the set of graphs that we can embed on a surface of constant *genus*. For such graphs, we know that a separator of size $O(n^{1/2})$ exists. Chakraborty et al. gave a polynomial-time algorithm which uses $O(n^{2/3}\log n)$ space to construct a separator of size $O(n^{2/3})$ in constant genus graphs [4].

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 23; pp. 23:1–23:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Jain and Tewari formalized the connection between separators in a class of undirected graphs and the reachability problem in the class of directed versions of those graphs [10]. They mainly show that if there exists a polynomial-time algorithm that uses $O(w \log n)$-space to find a separator of size $O(w)$, then there exists a polynomial-time algorithm that uses $O(w \log n)$ space to solve reachability as well.

In this paper, we continue along the above line of work and present a polynomial-time algorithm that uses $O(g^{1/2}n^{1/2} \log n)$ space to construct a separator of size $O(g^{1/2}n^{1/2})$ in a g-genus graph. Therefore, combining this construction with [10], we get a polynomial-time algorithm that uses $O(g^{1/2}n^{1/2} \log n)$ space to solve the reachability problem in g-genus graphs. Thus, for constant genus graphs, our approach gives a polynomial-time algorithm that uses $O(n^{1/2} \log n)$ space.

Our separator construction follows the standard paradigm used in previous constructions of separators for planar graphs and surface-embedded graphs. Hence some familiarity with earlier results such as those shown by Gazit and Miller [7], Koutis and Miller [11], Imai et al. [9], Ashida et al. [3], and Chakraborty et al. [4] is beneficial in understanding our construction. In particular, since our result is a generalization of Ashida et al. [3], we heavily borrow their framework.

Our Result

In this paper, we prove the following theorem.

▶ **Theorem 1.** *There exists a polynomial-time algorithm that takes as an input a graph G on n vertices along with its combinatorial embedding of genus g and outputs its separator of size $O(g^{1/2}n^{1/2})$. This algorithm uses $O(g^{1/2}n^{1/2} \log n)$ space.*

The running time of our algorithm is a polynomial in *both* the number of vertices n and the value of the genus g. To achieve the desired space-time bounds, given a graph G, we first find a maximal set of vertices in G whose *k-neighbourhoods* do not intersect each other. We call the vertices in this set *Boss vertices.* We associate each vertex of the graph to one of the Boss vertices. We call the set of vertices associated with the same Boss vertex a *Voronoi region.* If a non-contractible cycle of length $O(k)$ in the graph spans at most two of these Voronoi regions, we find that cycle and remove it from the graph. Removal of such a non-contractible cycle from the graph reduces its genus by at least one. Otherwise, if there exists no such cycle, we proceed by dividing the original graph further into a total of at most $O(n/k + g)$ regions so that each of them is bounded by a simple cycle of length $O(\sqrt{k})$ and the number of vertices present inside each region is at most $n/3$. We then use these regions to construct a *Frame Graph*, which is the graph induced by the vertices on these cycles. We assign weights to each face of the frame graph such that the weight of a face is equal to the number of vertices of the original graph *inside* the corresponding cycle. While constructing the frame graph, we might encounter some properties of the original graph, which allows us to output either a separator or a non-contractible cycle of size $O(k)$. Thus, in the end, we either have a non-contractible cycle of size $O(k)$, a small separator or the frame graph. If the result is a separator, then we output that separator. If the result is a non-contractible cycle C, we store it and restart the algorithm with the graph $G \setminus C$ as the input. The final output will be the union of C with the separator of $G \setminus C$. If the result is the frame graph, we use the algorithm of Gilbert et al. [8] to find a separator. For an appropriate value of k, our algorithm achieves the desired time and space bound.

Comparison with the previous result

The construction of separator by Chakraborty et al. [4] proceeds (roughly) as follows: Given a graph G they first find a subgraph H of the graph G, such that removal of vertices of H from G makes the resulting graph $G \setminus H$ planar. Subsequently, they obtain a separator of $G \setminus H$ using the algorithm of Imai et al. [9] and add it to the vertices of H to get a separator of the graph G. The subgraph H obtained in Chakraborty et al. [4] might not be a connected subgraph of G. We find a smaller separator by first finding a 2-connected weighted subgraph of G such that a weight-separator of this weighted subgraph acts as a separator of the original graph. We find this 2-connected subgraph by using *Ridge edges* which was previously used by Gazit and Miller [7], and Ashida et al. [3] for the case of planar graphs. We show that by first efficiently removing non-contractible cycles from Voronoi regions, ridge edges can be used in graphs of a higher genus. We call this 2-connected subgraph a *Frame graph*. We then show that a weight-separator for this Frame graph can be found efficiently and hence get our result.

Organization of the Paper

The rest of the paper is organized as follows. In section 2, we give some preliminary notations and definitions. We first divide the graph into Voronoi regions. We explain this procedure in section 3. We further divide Voronoi regions by using pre-frame-loops in section 3.1. In section 4, we show how to process the pre-frame-loops and construct a *frame graph* and then make *floor* modifications and *ceiling* modifications in this frame graph. We thus get the required subgraph. Finally, in section 5, we put it all together to construct the separator.

2 Preliminaries

A graph is an ordered triple $G = (V(G), E(G), \partial)$ where $V(G)$ is the set of *vertices*, $E(G)$ is the set of *edges* and ∂ is a function that assigns to each edge a pair of vertices. Let p be a path. We use $\mathsf{first}(p)$ to denote the first edge of p and $\mathsf{last}(p)$ to denote the last edge of p. In an undirected graph G, it is helpful to regard each edge in E as a pair of directed edges, or *darts*. Each dart goes from one vertex, called its *tail*, to another vertex, called its *head*. For a dart e, we use $\mathsf{tail}(e)$ to denote the tail of the dart, and similarly, we use $\mathsf{head}(e)$ to denote the head of the dart. The two darts that results from a single undirected edge are said to be *reverse* of each other. If two darts e_1 and e_2 are reverse of each other, we denote e_2 by $\mathsf{rev}(e_1)$ and e_1 by $\mathsf{rev}(e_2)$.

The genus of a surface Σ is the maximum number of non-intersecting simple closed curves in Σ such that the surface remains connected after cutting along these curves. The genus of a graph G is the smallest g such that G can be embedded on a surface of genus g. A surface is called orientable if it has two distinct sides; else, it is called non-orientable. In this paper, we only consider graphs that can be embedded on an *orientable* surface. Let G be a graph embedded on a surface S of genus g. The *faces* of the embedding of G are the connected components of $S \setminus G$. If a face is homeomorphic to an open disk, it is called a 2-cell. If every face is homeomorphic to an open disk, the embedding is called a 2-cell embedding. A combinatorial embedding of G is defined as $\pi = \{\pi_v \mid v \in V(G)\}$ where for each vertex v, π_v is a cyclic permutation of darts whose tail is v. This permutation of darts goes clockwise as per the embedding on the surface. For a dart e, we use $\mathsf{left}(e)$ to denote the face which is on the left of e and $\mathsf{right}(e)$ to denote the face on the right of e. A *triangulated graph* is a graph that is embedded on a surface such that every face is a 2-cell and has three boundary edges.

We define a dual graph $\tilde{G}$ of G for an embedding in the following way: $\tilde{G}$ contains a vertex $\tilde{v}$ corresponding to every face of G and two vertices of $\tilde{G}$ have an edge between them if their corresponding faces share an edge in G. We say that an edge $\tilde{e}$ of $\tilde{G}$ crosses an edge e of G if the faces at the endpoints of $\tilde{e}$ shares the edge e of G.

Let G be a graph embedded on a surface of genus g. Let U be a subset of vertices of G, F be a subset of edges of G, and R be a subset of faces of G. Then $G[U]$ denotes the subgraph of G, induced by the vertices in the set U. Similarly, $G[F]$ denotes the subgraph of G, containing all the edges of F together with their endpoints. By $G[R]$, we denote the graph containing all the vertices and edges in the boundary of a face in R.

Let G be a graph of genus g. A set R of its faces is a *region* if $\tilde{G}[R]$ is connected. The set of edges of G whose only one side has a face in R is called the *boundary* of R.

If G is a graph embedded on a surface and c is a cycle in G, then we define the *left graph* and *right graph* of c as follows: If e is a dart of c followed by $e' = \pi^k_{\mathsf{head}(e)}(\mathsf{rev}(e))$, then all edges $\pi_{\mathsf{head}(e)}(\mathsf{rev}(e)), \pi^2_{\mathsf{head}(e)}(\mathsf{rev}(e)), \ldots, \pi^{k-1}_{\mathsf{head}(e)}(\mathsf{rev}(e))$ are said to be on the *left* side of c. An edge e'' which is not incident with c and which is connected by a path in $G \setminus c$ to an end of an edge of the left side of c is also said to be on the left side. Now the left graph of c is defined as the edges on the left side of c together with all their ends. The right graph G is defined analogously. We will often use the term *inside of* c to denote the left graph of c and *outside of* c to denote the right graph of c. We do not include the cycle c itself in either of these sides.

▶ **Definition 2.** *A cycle c of a surface embedded graph G is called a* contractible cycle *if and only if one of the sides of c is planar. A cycle that is not contractible is called a* non-contractible cycle*.*

We say that a set C of cycles satisfies the 3-path-condition if the following property holds: If u and v are vertices of G and P_1, P_2 and P_3 are internally vertex disjoint paths from u to v. If two of the cycles $C_{i,j} = P_i \cup P_j (1 \leq i < j \leq 3)$ are not in C then the third one is also not in C. It is a well known fact that the set of non-contractible cycles satisfies the 3-path-condition [15].

We define $\mathsf{dist}(u, v)$ to be the length of the shortest path between two vertices u and v. We introduce a total order (denoted by $<_v$) in the vertex set V of the graph based on the distance from v. For any vertices u and w, we say that u is nearer to v than w (written as $u <_v w$) if we have either

- $\mathsf{dist}(u, v) < \mathsf{dist}(w, v)$ or
- $\mathsf{dist}(u, v) = \mathsf{dist}(w, v)$ and u has a smaller index than w

For any set W of vertices of G, $\mathsf{nrst}_v(W)$ denotes a vertex u in W such that $u <_v w$ of all $w \in W \backslash \{u\}$. For any sets W and W' of vertices, we write $W <_v W'$ if $\mathsf{nrst}_v(W) <_v \mathsf{nrst}_v(W')$.

Let G be a graph of genus g. A closed loop c is a sequence of *distinct* darts $e_1, e_2, \ldots, e_m$ of G such that $\mathsf{head}(e_i) = \mathsf{tail}(e_{(i+1) \bmod m})$.

▶ **Definition 3.** *Let G be a weighted-graph with positive integral weights on each vertex that sums to n and $\alpha \in (0, 1)$. An α-separator of G is a set S of vertices of G such that the removal of S creates disconnected subgraphs, each of which has at most αn weight, where the weight of a subgraph is the sum of the weights of the vertices in it.*

We use a multitape Turing machine model to discuss the space-bounded polynomial-time algorithms. A multi-tape Turing machine consists of a read-only input tape, a write-only output tape, and a constant number of work tapes. We measure the space complexity of a multitape Turing machine by the total number of bits used in the work tapes.

We note that Allender and Mahajan [2] showed that the problem of testing whether a graph is planar or not is in SL. They also gave the SL algorithm to construct the planar embedding. Subsequently, Reingold [14] showed that SL = L hence there exists a logspace algorithm to test if a graph is planar and also produce its embedding. We summarize this fact in the following Lemma.

▶ **Lemma 4.** *There exists a logspace algorithm that tests whether the input graph is planar and if so, it outputs an embedding of the input graph.*

Gilbert, Hutchinson and Tarjan proved the existence of an $O(n^{1/2}g^{1/2})$ size separator for the graphs of genus g [8]. They also presented an $O(n+g)$ time algorithm to find the separator. Therefore we can conclude that their algorithm runs in $O((n+g)\log n)$ space. We can thus use the following Lemma for our result.

▶ **Lemma 5.** *There exists a polynomial-time algorithm that takes, as an input, an n-vertex graph of genus g along with its combinatorial embedding and finds its separator of size $O(n^{1/2}g^{1/2})$ using $O((n+g)\log n)$ space.*

We will need the notion of fundamental cycles in our separator construction; therefore, we define it formally.

▶ **Definition 6.** *Let G be a graph and T be a spanning tree of G. Let e be an edge that does not belong to T. A simple cycle c, which consists of e and the path in T joining the endpoints of e, is called a fundamental cycle.*

3 Voronoi Region

As we discussed in section 1, we start by dividing the input graph into something that we call *Voronoi regions*. In this section, we define the notion of Voronoi Regions and explain how they could be constructed in a space-efficient manner. This notion has been previously used in designing a separator for planar graphs by Imai et al., Ashida et al., Gazit and Miller, and Koutis and Miller [9, 3, 7, 11].

We first define the k-neighbourhood of a vertex. This is a key tool that will help us define and construct a Voronoi region.

▶ **Definition 7.** *Let G be a graph and v be a vertex of G. Let $L(v,i)$ be the set of vertices at distance i from v. The k-neighbourhood $N_k(v)$ of a vertex v is defined as:*

$$N_k(v) = \bigcup_{1\leq i\leq d} L(v,i)$$

where d is the smallest integer such that $|\bigcup_{1\leq i\leq d} L(v,i)| \geq k$.

Note that we have defined k-neighbourhood in a slightly different way when compared to the definition of Imai et al. [9] and Chakraborty et al. [5]. In their work, $N_k(v)$ is chosen to contain at most k vertices, while here, it contains at least k vertices. We believe this definition makes our proof simpler to follow.

▶ **Definition 8.** *Let G be a graph. A set I of vertices of G is called a k-maximal independent set if the following holds:*

- *For every $b_1, b_2 \in I$, $N_k(b_1) \cap N_k(b_2) = \emptyset$.*
- *For every v that is not in I, we have a vertex $b \in I$ such that $N_k(v) \cap N_k(b) \neq \emptyset$.*

▶ **Lemma 9.** *There exists an $O((k + n/k) \log n)$-space and polynomial-time algorithm that takes a graph G as input and outputs a k-maximal independent set I.*

The proof of the above Lemma is quite straightforward. We refer readers to [9, 4].
For a graph G, we will use the notation $\mathsf{ind}(G)$ to denote the set returned by the algorithm of Lemma 9.

▶ **Definition 10.** *Let G be a graph. For any vertex v, the* boss-vertex *of v is a vertex b of $\mathsf{ind}(G)$ such that $N_k(b) <_v N_k(b')$, for all $b' \in \mathsf{ind}(G) \setminus \{b\}$. We define $\mathsf{vor}(b)$ to be the set of all vertices whose boss-vertex is b. We use $\mathsf{boss}(v)$ to denote the boss-vertex of v.*

Note that the graph induced by the vertices in the set $\mathsf{vor}(b)$ form a connected component in G. Therefore, the faces corresponding to these vertices form a region in $\tilde{G}$. We will henceforth call $\mathsf{vor}(b)$ the *Voronoi region* of b.

We note that while the Voronoi region of a vertex b can be large, its diameter is $O(k)$. We will now show that the BFS-tree of this Voronoi region can still be constructed in $O((n/k + k) \log n)$ space and polynomial time. To construct a rooted tree using small space, it suffices to show an algorithm to determine the parent of a given vertex $v \in \mathsf{vor}(b)$ in the BFS tree. The algorithm is the following: To determine the parent of a vertex v in $\mathsf{vor}(b)$, we first construct the BFS-tree T of k-neighbourhood of b. If $v \in N_k(b)$ then the parent of v is the same as its parent in T. Otherwise, construct the BFS-tree of $N_k(v)$. Let v' be the vertex in $N_k(v) \cap N_k(b)$ such that $\mathsf{dist}(b, v')$ is minimum. Break ties by picking the one with a smaller index. Consider the path from v to v' in the BFS tree of $N_k(v)$. The parent of v is the vertex adjacent to v in this path. We summarize in the following Lemma.

▶ **Lemma 11.** *Let G be a graph and b be a vertex of $\mathsf{ind}(G)$. There exists a polynomial time algorithm that constructs the BFS-tree of $\mathsf{vor}(b)$ in $O((k + n/k) \log n)$ space.*

A similar lemma was observed for planar graphs by Imai et al. [9]

The input graph might contain small non-contractible cycles. We require that the union of any two Voronoi regions do not have a non-contractible cycle, similarly as Chakraborty et al. [4]. Thus, we remove such non-contractible cycles from the graph using the following Lemma in our main algorithm.

▶ **Lemma 12** ([4])**.** *There is an $O((k + n/k) \log n)$-space and polynomial time algorithm that takes a graph G, and two boss-vertices b_1 and b_2 as input and checks for a non-contractible cycle of size $O(k)$ in $\mathsf{vor}(b_1) \cup \mathsf{vor}(b_2)$. The algorithm outputs one such cycle if it exists.*

Proof. First, consider the case when $\mathsf{vor}(b_1) \cup \mathsf{vor}(b_2)$ forms a connected subgraph of G. We know that $\mathsf{vor}(b_1)$ and $\mathsf{vor}(b_2)$ can be computed in $O((k + n/k) \log n)$ space and polynomial time by Lemma 11. We combine the BFS-trees of $\mathsf{vor}(b_1)$ and $\mathsf{vor}(b_2)$ using an arbitrary edge to get a spanning tree of $\mathsf{vor}(b_1) \cup \mathsf{vor}(b_2)$ with diameter $O(k)$. We denote this spanning tree as T. Note that T can be computed in polynomial-time and $O((n/k + k) \log n)$ space. We know that the set of all non-contractible cycles of any graph G satisfy 3-path condition [15]. Since the diameter of T is $O(k)$, any fundamental cycle of this tree of size $O(k)$. The 3-path condition implies that if a non-contractible cycle exists, then one of the fundamental cycles is non-contractible (see Allender et al. 2005, Lemma 5.1 [1]). We can check whether a cycle is contractible by checking the planarity of the left and the right sides of the cycle. The left and the right side of a given cycle can be computed in logspace by using reachability queries [14]. Therefore, by Lemma 4, we can check if a cycle is contractable in $O(\log n)$ space. Thus, the lemma follows. In the other case where $\mathsf{vor}(b_1)$ and $\mathsf{vor}(b_1)$ are not connected, we can apply the same procedure on spanning trees of $\mathsf{vor}(b_1)$ and $\mathsf{vor}(b_2)$ separately. ◀

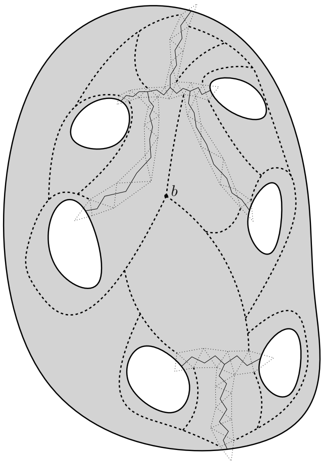


Figure 1 A diagram showing vor(b) for a boss vertex b. The part of the surface where the vertices of vor(b) are present is shown in grey colour. The boundary of the Voronoi region is shown using thick solid lines. The ridge edges are shown using normal solid lines. Dashed lines show some of the edges of the spanning tree of vor(b). Dotted lines show faces in G that corresponds to a vertex v, which is an endpoint of a ridge edge in $\tilde{G}$.

As mentioned in the introduction, we will use the Voronoi regions to construct our Frame graph. For this construction, we first divide the Voronoi Regions.

3.1 Dividing Voronoi Regions using Pre-Frame-Loops

In this subsection, we find a set of loops in the input graph G. Each of these loops contains vertices of at most two Voronoi regions *inside* them. We then further process these loops so that the number of vertices inside them is small.

Let G be a triangulated graph of genus g. Note that any connected component of G forms a region in $\tilde{G}$. Also, note that since the size of each face of G is three, all the vertices of the graph $\tilde{G}$ will have degree three. Thus, a region of faces in $\tilde{G}$ will have a boundary that is a set of vertex-disjoint simple cycles.

We require two kinds of edges in the dual graph to construct the desired loops. One is the set of the boundary edges of all the Voronoi regions, and the other is the set of *Ridge edges.* Ridge edges have been used previously by Gazit and Miller [7] and Ashida et al. [3]. We define them as follows.

▶ **Definition 13.** *Let G be a graph and b be a boss-vertex. Let T be the BFS tree of* vor(b)*. For an edge e of $G[\mathsf{vor}(b)]$ that does not belong to T, let c_e be the fundamental cycle induced by e on T. If each of the two sides of the cycle c_e contains at least one boundary cycle of* vor(b)*, then the edge $\tilde{e}$ of $\tilde{G}$ crossing e is called a ridge edge.*

Figure 1 shows ridge edges in the Voronoi region of a boss vertex b.

▶ **Definition 14.** *Let G be a graph of genus g. Let $\tilde{B}$ be the set of boundary edges of* vor(b) *for all boss vertices b. Similarly, let $\tilde{R}$ be the set of ridge-edges. A* branch vertex *is a degree three vertex in the graph $\tilde{G}[\tilde{B} \cup \tilde{R}]$. For each branch vertex $\tilde{v}$, the boundary of the face consisting of three vertices incident to $\tilde{v}$ is a* branch-triangle*. Two branch vertices are called* adjacent *to each other if a path connects them consists of darts corresponding to the edges in the set $\tilde{B} \cup \tilde{R}$ such that no other branch vertex exists on this path. The path connecting adjacent branch vertices is called a* connector*. We denote the set of connectors in $\tilde{G}$ by* con(G)*.*

Let $\tilde{p}$ be a connector. Note that the end points of $\tilde{p}$ are an adjacent pair of branch vertices. Also note that, $\mathsf{boss}(\mathsf{left}(\mathsf{first}(p)))$ is same as $\mathsf{boss}(\mathsf{left}(\mathsf{last}(p)))$ and $\mathsf{boss}(\mathsf{right}(\mathsf{first}(p)))$ is same as $\mathsf{boss}(\mathsf{right}(\mathsf{last}(p)))$. We define a pre-frame-loop with respect to $\tilde{p}$ as follow.

▶ **Definition 15.** *Let G be a graph embedded on a surface of genus g. For any connector $\tilde{p}$, a pre-frame-loop (denoted by $\mathsf{pfloop}(\tilde{p})$) is a closed loop that consists of*

1. *A path from $\mathsf{right}(\mathsf{first}(p))$ to $\mathsf{boss}(\mathsf{right}(\mathsf{first}(p)))$ in the BFS-tree of $\mathsf{vor}(\mathsf{boss}(\mathsf{right}(\mathsf{first}(p))))$*
2. *A path from $\mathsf{boss}(\mathsf{right}(\mathsf{first}(p)))$ to $\mathsf{right}(\mathsf{last}(p))$ in the BFS-tree of $\mathsf{vor}(\mathsf{boss}(\mathsf{right}(\mathsf{first}(p))))$*
3. *A branch-triangle dart e_{last} from $\mathsf{right}(\mathsf{last}(p))$ to $\mathsf{left}(\mathsf{last}(p))$*
4. *A path from $\mathsf{left}(\mathsf{last}(p))$ to $\mathsf{boss}(\mathsf{left}(\mathsf{last}(p)))$ in the BFS-tree of $\mathsf{vor}(\mathsf{boss}(\mathsf{left}(\mathsf{last}(p))))$.*
5. *A path from $\mathsf{boss}(\mathsf{left}(\mathsf{last}(p)))$ to $\mathsf{left}(\mathsf{first}(p))$ in the BFS-tree of $\mathsf{vor}(\mathsf{boss}(\mathsf{left}(\mathsf{last}(p))))$.*
6. *A branch-triangle dart e_{fst} from $\mathsf{left}(\mathsf{first}(p))$ to $\mathsf{right}(\mathsf{first}(p))$.*

We denote the set of all the pre-frame-loop in G as $\mathsf{pfloop}(G)$

The proof of the following lemma is straightforward.

▶ **Lemma 16.** *There exists an $O((k + n/k)\log n)$-space and polynomial-time algorithm that takes G as an input and outputs the list $\mathsf{pfloop}(G)$ of all pre-frame-loops in G.*

In the next section, we will use these pre-frame loops to create faces of our subgraph. Following Ashida et al. [3], we call this new graph Frame Graph.

4 Frame Graph

We wish to use pre-frame-loops to create faces of the frame graph. In order to do this, we first preprocess these loops so that the *inside* of each loop is small, i.e., has at most $n/3$ vertices in it. This preprocessing would ensure that the weight on any face of the frame graph is bounded. In the second step, we remove those edges of the loop for which both of its darts are traversed and thus break the loop into simple cycles. These cycles will act as boundaries of the faces in the frame graph.

Consider a connector $\tilde{p}$ of the input graph G and the pre-frame-loop c induced by $\tilde{p}$. Note that c is in the union of two Voronoi regions. Since we have eliminated all non-contractible cycles from the union of any two Voronoi regions, c cannot contain a non-contractible cycle. Thus, c divides the surface. A pre-frame loop is of type A if it consists of two boss vertices, and it is of type B if it consists of only one boss vertex (see Figure 2).

Let the part of a connector $\tilde{p}$ excluding its first and last vertex be called the *body* of $\tilde{p}$. Let P_0 denote the surface of $G \setminus c$ that has the body of the connector $\tilde{p}$. Call this *inside* of c. Let n_0 be the number of vertices in P_0 not including the vertices of c. We say that the inside of c is *large* if n_0 is greater than $2n/3$. Note that the inside of c is included in the union of atmost two Voronoi regions. Let b_1 and b_2 be the boss vertices of these two regions. We use the BFS-Trees of $\mathsf{vor}(b_1)$ and $\mathsf{vor}(b_2)$ to find a spanning tree of $\mathsf{vor}(b_1) \cup \mathsf{vor}(b_2)$. Since $\mathsf{vor}(b_1) \cup \mathsf{vor}(b_2)$ does not have a non-contractible cycle, it has a planar embedding. Consider a spanning tree T of $\mathsf{vor}(b_1) \cup \mathsf{vor}(b_2)$. There exists a fundamental cycle of this tree in the triangulated version of the graph $G[\mathsf{vor}(b_1) \cup \mathsf{vor}(b_2)]$ which acts as its separator [12]. Since by removing the boundary of the pre-frame loop from the graph, we can get components, the largest of which is formed by the vertices $\mathsf{vor}(b_1) \cup \mathsf{vor}(b_2)$, we can combine the separator of $G[\mathsf{vor}(b_1) \cup \mathsf{vor}(b_2)]$ with the boundary of the pre-frame-loop to get a separator of the whole graph G. Since the length of the boundary of pre-frame-loop is $O(k)$ and the diameter of Voronoi region of any boss vertex is $O(k)$, we get the following lemma:

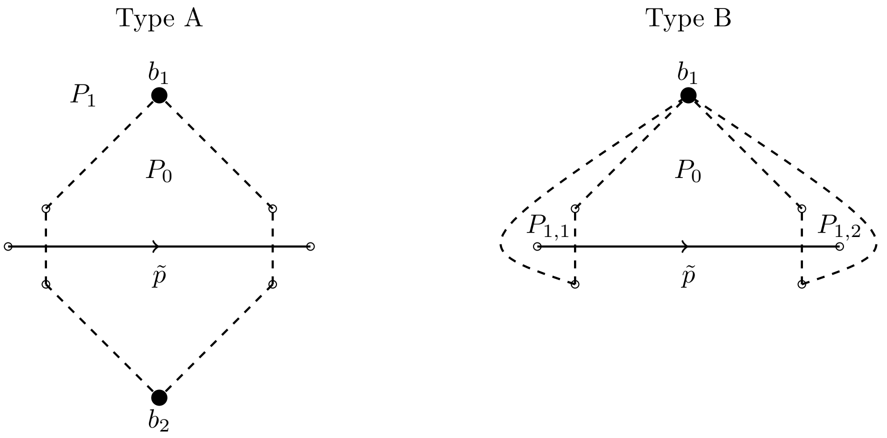


Figure 2 On the left, a pre-frame-loop of type A. The two boss vertices corresponding to the loop are b_1 and b_2. On the right, a pre-frame-loop of type B. The only boss vertex corresponding to this loop is b_1.

▶ **Lemma 17.** *Let G be a graph of genus g which contains a pre-frame loop whose inside is large. There exists a polynomial-time algorithm that takes as an input G and outputs a separator of G of size $O(k)$ in $O((k + n/k)\log n)$ space.*

Thus, if any of the pre-frame-loop acts as a separator or has a large inside, we can get a separator of the graph G. Otherwise, we construct a set C in the following way: We first add all the pre-frame-loop of type A into C. Note that if a pre-frame-loop c is of type B, it divides the surface into three parts. Call the two parts of the surface, which does not contain the body of the connector, $P_{1,1}$ and $P_{1,2}$ respectively. Let the number of vertices in P_0, $P_{1,1}$ and $P_{1,2}$ be n_0, $n_{1,1}$ and $n_{1,2}$ respectively. We see that either $n_{1,1} > 2n/3$ or $n_{1,2} > 2n/3$, for otherwise, our pre-frame-loop acts as a separator. Let us assume, without loss of generality, that $n_{1,2} > 2n/3$. We merge P_0 and $P_{1,1}$ into a single surface, and add the loop c_0 bounding this surface to the set C. The *inside* of c_0 is the side containing the surfaces P_0 and $P_{1,1}$.

Now, consider a loop c of C, that is not contained in the inside of any other loop c of C. Let E_c be the set of darts whose reverse does not appear in c. Let E be the union of E_c over all such c. We observe that E is a set of simple cycles, which we call frame-cycles and denote by $\mathsf{fcycle}(G)$.

4.1 Definition and construction of Frame Graph

▶ **Definition 18.** *Let G be a graph of genus g. Let E_1 be the set of all frame-cycles edges, and let E_2 be the set of all branch-triangle edges. A* frame-graph *of G is a subgraph $H = G[E_1 \cup E_2]$. For each face of a frame-graph H, its weight is the number of vertices of G located inside that face. We denote the frame graph of G by $\mathsf{frame}(G)$.*

▶ **Definition 19.** *Let G be a triangulated graph. Let $L(v, i)$ be the set of vertices at distance i from v. Let $d_{nb}(v)$ be the largest d such that $|\cup_{0\le i\le d} L(v,i)| < k$. For any boss-vertex $b \in \mathsf{ind}(G)$, let $d_{core}(b)$ denote the largest $d \le d_{nb}(b)$ such that $|L(b,d)| \le k^{1/2}$. The* core *of b (denoted by $\mathsf{core}(b)$) is defined by*

$$\mathsf{core}(b) = \bigcup_{0\le i\le d_{core}(b)} L(b,i)$$

Note that core(b) forms a region in $\tilde{G}$. The boundary of this region might not be a single cycle. In the next definition, we pick one of these cycles to be the core boundary-cycle and use it to construct the core cycle in the graph G.

▶ **Definition 20.** *Let G be a triangulated graph of genus g and b be a boss-vertex. The* core-boundary-cycle *of b is the boundary cycle of the region $\mathsf{core}(b)$ in $\tilde{G}$ that has the largest number of dual-vertices on its outside.*

The core-cycle *of $\mathsf{core}(b)$ is a directed cycle induced by the set of vertices in $\mathsf{core}(b)$ sharing an edge with the core boundary cycle. The inside of the core-cycle is the side with the boss-vertex b.*

For any $l \geq 1$, let $L_{nb}(l)$ denote a set of vertices v of G whose distance from its nearest k-neighborhood in $\{N_k(b)\}_{b \in \mathsf{ind}(G)}$ is l. More formally,

$$L_{nb}(l) = \{v \mid \mathsf{dist}(v, v_{\mathsf{nrst}}) = l, \text{where } v_{\mathsf{nrst}} = \mathsf{nrst}_v(N_k(\mathsf{boss}(v)))\}.$$

Let $\tilde{L}_{nb}(l)$ denotes the set of faces in $\tilde{G}$ corresponding to the vertices in the set $L_{nb}(l)$. Let C be a region of $\tilde{L}_{nb}(l)$. Each boundary edge of C is an edge between a pair of vertices of level either $l-1$ and l or l and $l+1$. Let us call the former one an *interior* edge and the latter one an *exterior* edge. We call a boundary cycle an *interior boundary cycle* if it consists of interior edges. Similarly, we call a boundary cycle an *exterior boundary cycle* if it consists of exterior edges.

▶ **Definition 21.** *Let $\tilde{c}$ be any interior boundary cycle corresponding to $L_{nb}(l)$. Let c be the loop in C formed by the set of vertices sharing a boundary edge with $\tilde{c}$, and let D_c be the set of cycles obtained from c by removing all the darts in the loop whose reverse also appears in the loop. An interior-cycle is a cycle in D_c.*

We define an *exterior-cycle* in a similar way. A cycle is said to be a *small* cycle if it consists of at most $k^{1/2}$ vertices. We denote the set of small interior cycles by $\mathsf{smint}(G)$ and the set of small exterior cycles by $\mathsf{smext}(G)$. A contractible cycle is said to be *light* if it has less than $n/3$ vertices in its inside.

▶ **Definition 22.** *A floor cycle is a light and small interior cycle if it is not inside any other light and small interior cycle. For any boss-vertex b which is not contained in any floor-cycle, we regard the core-cycle of $\mathsf{core}(b)$ also as a floor-cycle. A ceiling-cycle is a light and small exterior cycle that is not inside any other light and small exterior-cycle, and that has at least one dual-vertex of some branch-triangle on its inside.*

▶ **Definition 23.** *Let G be a graph of genus g. Let F and C be respectively a set of floor-cycles and ceiling-cycles having at least one vertex of $\mathsf{frame}(G)$ in their insides. Let E_1' be the set of edges of G that appear in some cycle in $F \cup C$ and E_2' be the set of edges of $\mathsf{frame}(G)$ that are not in the inside of any cycle of $F \cup C$. A graph with vertices U' and edges D' is a modified frame-graph denoted as $\mathsf{mframe}(G)$, where $D' = E_1' \cup E_2'$ and U' is the set of all vertices that are endpoints of edges of D'. For each face of a modified frame-graph $\mathsf{mframe}(G)$, its weight is the number of vertices of G located in the face.*

The following Lemma is a generalization of a result that was presented by Ashida et al. [3]. They presented a similar lemma for planar graphs. The proof of the following Lemma has been moved to Appendix.

▶ **Lemma 24.** *Let G be a graph of genus g such that voronoi region $\mathsf{vor}(b_1) \cup \mathsf{vor}(b_2)$ does not contain a non-contractible cycle for any two vertices $b_1, b_2 \in \mathsf{ind}(G)$, $\mathsf{pfloop}(\tilde{p})$ is not a separator of G for any connector $\tilde{p}$, the inside of any loop in $\mathsf{pfloop}(\tilde{p})$ is not large, $\mathsf{core}(b)$ is not a separator of G for any boss-vertex b, and no cycle in $\mathsf{smext}(G)$ or $\mathsf{smint}(G)$ is a non-contractible cycle. Following statements hold:*

1. *The weight of each face of* mframe(G) *is less than* $n/3$.
2. mframe(G) *is* 2*-connected.*
3. *Size of each face of* mframe(G) *is* $O(k^{1/2})$.
4. *The number of faces in* mframe(G) *is* $O(n/k + g)$

5 Construction of separator

Using the tools developed so far, we can obtain a space-efficient algorithm which, given a graph G as input, outputs either a separator, a non-contractible cycle or the modified frame graph mframe(G). We summarize this in the following Lemma.

▶ **Lemma 25.** *Let G be a g-genus triangulated graph of n vertices. For any positive integer k, there is a polynomial time, $O((n/k + k)\log n)$-space algorithm that takes G along with its combinatorial embedding as input and outputs one of the following:*

1. *A non-contractible cycle of size $O(k)$ of G.*
2. *A separator of size $O(k)$ of G.*
3. *A a weighted subgraph H' of G that satisfies the following conditions:*
   a. *The weight of each face f of H' is proportional to the number n_f of vertices of G located inside the face, and is less than $n/3$*
   b. *H' is 2-connected.*
   c. *H' contains $O(n/k + g)$ faces.*
   d. *The size of each face of H' is $O(k^{1/2})$.*

Proof. We first find a k-maximal independent set ind(G) of G in $O((n/k+k)\log n)$-space and polynomial time using lemma 9. We then check for a non-contractible cycle in vor$(b_i) \cup$ vor(b_j) for all pairs of boss-vertices b_i and b_j using lemma 12. If we manage to find such a cycle, we output it. Otherwise, we pick each pre-frame loops using lemma 16 see if it acts as a separator of the graph. If so, we output it.

If the algorithm has not produced an output so far, we see if the inside of any pre-frame-loop is large. If so, we use lemma 17 to find a separator of the graph. For every boss vertex b, we check if core(b) is a separator. If so, we output it. Next, we check if any cycle in smext(G) or smint(G) is a non-contractible cycle. If so, we output it. Otherwise, we output the modified frame graph mframe(G). ◀

With these ingredients, we are now ready to prove our main theorem.

Proof of Theorem 1. Elberfeld and Kawarabayashi presented an algorithm to construct a combinatorial embedding of a graph of the constant genus in logspace [6]. Hence, we do not require a combinatorial embedding as part of the input when dealing with a constant-genus graph. Otherwise, we require the combinatorial embedding of the graph as an input. We assume that the genus of the input graph g is at most $O(n)$. Let π be the combinatorial embedding of G. We first triangulate the input graph in logspace. To do this, for each face f of the input graph, we connect each vertex of f with the lowest index vertex in it. This triangulation is done implicitly, whenever required, as storing the triangulated graph will require a large amount of space. We call the resultant triangulated graph G. Note that triangulating the graph only introduces more edges; therefore, a separator for G will also be a separator for the input graph. Our objective now is to construct a separator of G. We do this by iteratively applying Lemma 25. We will describe the algorithm by describing an iteration of it. Before the ith iteration, we will have a set S of vertices which is empty before the first iteration. Let $G_1, G_2, \ldots, G_m$ be the set of connected components in $G \setminus S$. We will describe the ith iteration as follows. The algorithm takes the component G_j whose size n_j is greater than $2n/3$. If no such component exists, then the set S would be a separator of G, and the algorithm outputs S and halts. Otherwise, consider the embedding induced by π on G_j as its embedding. If the genus g_j of this component is zero, the algorithm uses Imai et al. planar separator algorithm to get its separator S_1 and outputs $S \cup S_1$. If its genus is non-zero we apply the algorithm from Lemma 25 on G_j with k set as $n_j^{1/2}/g_j^{1/2}$. If the result of the application of the algorithm from Lemma 25 on G_j is a non-contractible cycle, say S_2, then we add the vertices of S_2 to the set S and continue with the next iteration. If the result is a separator, say S_3, we output the set $S \cup S_3$ as the separator for the entire graph. Otherwise, if the result is a subgraph H' of G_j, we take its dual $\tilde{H}'$ and find its separator $\tilde{S}'$ using Lemma 5. $\tilde{S}'$ is a set of faces of H'. Consider the set S_4 of vertices on the boundary of these faces. We return the set $S \cup S_4$. To see that the size of separator returned by the above algorithm is $O(g^{1/2}n^{1/2})$, note that Lemma 25 returns a non-contractible cycle, the genus of the graph is reduced by at least one. It is sufficient for us to use the induced embeddings (see Mohar and Thomassen [13], Proposition 4.2.1 and Lemma 4.2.4). Hence, our algorithm can return at most g such cycles; each has length $O(k)$. For our value of k, the total number of vertices in all such cycles can be at most $O(g^{1/2}n^{1/2})$. If it does not returns a non-contractible cycle, then it returns either a separator of size $O(k) \leq O(g^{1/2}n^{1/2})$ or it returns the subgraph H'. The number of faces in H' is $O(n/k + g)$. Hence the size of the separator returned by using the algorithm of Gilbert et al. [8] on the dual of H' will be $O(g^{1/2}(n/k + g)^{1/2})$. Size of each face of H' is at most $k^{1/2}$, hence, size of the set S_4 is $O(k^{1/2}g^{1/2}(n/k + g)^{1/2})$. For our value of k, this is at most $O(g^{1/2}n^{1/2})$. ◀

We can use the following Lemma, which was formalized by Jain and Tewari [10] to get a space-efficient polynomial-time algorithm for reachability in constant-genus graphs. Reachability is determining if there is a directed path from one vertex to another in a directed graph.

▶ **Lemma 26.** *Let $\mathcal{G}$ be a class of graphs and $w : \mathcal{N} \mapsto \mathcal{N}$ be a function. If there exist a polynomial-time algorithm that uses $O(w(n) \log n)$ space to find a separator of size $w(n)$ then there exists a polynomial time algorithm to decide reachability in G that uses $O(w(n) \log n)$ space.*

▶ **Corollary 27.** *There exists a polynomial-time algorithm that uses $O(n^{1/2} \log n)$ space to solve reachability in a constant-genus graph.*

Previously, a polynomial-time algorithm that uses $O(n^{1/2} \log n)$ space for reachability was known for planar graphs [9]. While for constant-genus graphs, a polynomial-time algorithm that uses $O(n^{2/3} \log n)$ space was known [4]. Corollary 27 improves the space-bound to $O(n^{1/2} \log n)$. Our result can thus be seen as both a generalization of Imai et al. [9] and as an improvement to a previous result by Chakraborty et al. [5].

References

1 Eric Allender, Samir Datta, and Sambuddha Roy. The directed planar reachability problem. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, pages 238–249, 2005. doi:10.1007/11590156_19.

2 Eric Allender and Meena Mahajan. The complexity of planarity testing. *Information and Computation*, 189(1):117–134, 2004. doi:10.1016/j.ic.2003.09.002.

3 Ryo Ashida, Tomoaki Imai, Kotaro Nakagawa, A. Pavan, N. V. Vinodchandran, and Osamu Watanabe. A sublinear-space and polynomial-time separator algorithm for planar graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:91, 2019.

4 Diptarka Chakraborty, Aduri Pavan, Raghunath Tewari, N. V. Vinodchandran, and Lin F. Yang. New time-space upperbounds for directed reachability in high-genus and h-minor-free graphs. In *Proceedings of the 34th Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*, pages 585–595, 2014.

5 Diptarka Chakraborty and Raghunath Tewari. An $O(n^\epsilon)$ space and polynomial time algorithm for reachability in directed layered planar graphs. *ACM Transactions on Computation Theory (TOCT)*, 9(4):19:1–19:11, 2017.

6 Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 383–392. ACM, 2014. doi:10.1145/2591796.2591865.

7 H. Gazit and G. L. Miller. A parallel algorithm for finding a separator in planar graphs. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science (FOCS 1987)*, pages 238–248, October 1987. doi:10.1109/SFCS.1987.3.

8 John R Gilbert, Joan P Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs of bounded genus. *Journal of Algorithms*, 5(3):391–407, 1984. doi:10.1016/0196-6774(84)90019-1.

9 Tatsuya Imai, Kotaro Nakagawa, Aduri Pavan, N. V. Vinodchandran, and Osamu Watanabe. An $O(n^{\frac{1}{2}+\epsilon})$-space and polynomial-time algorithm for directed planar reachability. In *Proceedings of the 28th Conference on Computational Complexity (CCC 2013)*, pages 277–286, 2013.

10 Rahul Jain and Raghunath Tewari. Reachability in High Treewidth Graphs. In *Proceedings of the 30th International Symposium on Algorithms and Computation (ISAAC 2019)*, 2019.

11 Ioannis Koutis and Gary L. Miller. A linear work, $O(n^{1/6})$ time, parallel algorithm for solving planar laplacians. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007)*, pages 1002–1011, 2007.

12 Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. doi:10.1137/0136016.

13 B. Mohar and C. Thomassen. *Graphs on Surfaces*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2001. URL: https://books.google.com.sg/books?id=_VFKscYKSicC.

14 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008.

15 Carsten Thomassen. Embeddings of graphs with no short noncontractible cycles. *Journal of Combinatorial Theory, Series B*, 48(2):155–177, 1990. doi:10.1016/0095-8956(90)90115-G.

A Appendix

A.1 Proof of Lemma 24

Proof. We will prove each of the four statements of the proof in order.

1. Consider a face of the frame-graph $\mathsf{frame}(G)$. The boundary of this face is either a frame-cycle or a branch triangle. The weight of a branch-triangle is zero, while the number of vertices inside a frame-cycle is less than $n/3$ by construction. Hence the weight of any face of $\mathsf{frame}(G)$ is less than $n/3$. The number of vertices inside a floor or a ceiling cycle is less than $n/3$ by definition. Hence the weight of any face of $\mathsf{mframe}(G)$ is also less than $n/3$.
2. We first prove that $\mathsf{frame}(G)$ is 2-connected. Let u and v be two distinct vertices of $\mathsf{frame}(G)$. We have the following cases:

Case 1 (Both u and v are on same frame cycle in pfloop(G)) : Since u and v are on a cycle, there exist two vertex-disjoint paths from u to v.

Case 2 (u and v are on two different cycles in pfloop(G)) : Let c_u and c_v be the cycles of $\mathsf{pfloop}(G)$ which contain vertices u and v respectively. Let $\tilde{p}_u$ and $\tilde{p}_v$ be the connectors whose bodies are contained in c_u and c_v respectively. We first note that there is a sequence of connecters $\tilde{p}_u = \tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_k = \tilde{p}_v$ such that $\tilde{p}_i$ and $\tilde{p}_{i+1}$ has a common end point for each $i \in [1, k-1]$. Also note that the body of $\tilde{p}_i$ is contained in a cycle c_i of $\mathsf{pfloop}(G)$. Orient the darts of $\tilde{p}_i$ to form a path from $\mathsf{first}(p_1)$ to $\mathsf{last}(p)_k$. Let c_i^{left} be the path from $\mathsf{left}(\mathsf{first}(\tilde{p}_i))$ to $\mathsf{left}(\mathsf{last}(\tilde{p}_i))$. Similarly, let c_i^{right} be the path from $\mathsf{right}(\mathsf{first}(\tilde{p}_i))$ to $\mathsf{right}(\mathsf{last}(\tilde{p}_i))$. We see that c_i^{left} and c_i^{right} do not share any vertex. Thus, we can see that there exist two vertex-disjoint paths q_{left} and q_{right} from u to v such that q_{left} contains vertices of c_i^{left} and q_{right} contains vertices of c_i^{right} for all $i \in [2, k-1]$.

The analysis of other cases is similar.

We will show that there are two vertex-disjoint paths between any two vertices u and v of the graph $\mathsf{mframe}(G)$.

Case 1 (u and v are both in frame(G)): Since we have proved that $\mathsf{frame}(G)$ is two connected, we know that there exist two vertex disjoint paths q_{left} and q_{right} between u and v in $\mathsf{frame}(G)$. Note that several floor-cycles and ceiling-cycles were added to $\mathsf{frame}(G)$ and the vertices inside them were removed in order to construct $\mathsf{mframe}(G)$. Let c be one such cycle.

- If c intersects both q_{left} and q_{right}. Let u_{left} and v_{left} denote the first and the last vertices of q_{left} which intersects c. Similarly, let u_{right} and v_{right} denote the first and the last vertices of q_{right} which intersects c. These four vertices divide c into four paths c_1, c_2, c_3 and c_4. Let the set of these four paths be C. Then one of the following statements is true:
  - There exist paths from u_{left} to v_{left} and from u_{right} to v_{right} in C.
  - There exist paths from u_{left} to v_{right} and from u_{right} to v_{left} in C.

  For both the above cases, we see that there exist two disjoint paths from u to v.
- If c intersects only one of the path q_{left} and q_{right} then we can modify that path to contain part of the cycle.

Case 2 (u and v are on different floor-cycles or ceiling-cycles c_u and c_v): Let w_u and w_v be vertices of $\mathsf{frame}(G)$ inside c_u and c_v respectively. We know such vertices exits because of the way these cycles are defined. Since the graph $\mathsf{frame}(G)$ is 2-connected, there exist two disjoint paths q_{left} and q_{right} between w_u and w_v in it. Let u_{left} be the last intersection of q_{left} and c_u. Similarly let u_{right} be the last intersection of q_{right} and c_u. Note that, since the paths q_{left} and q_{right} are disjoint, $u_{left} \neq u_{right}$. We similarly define v_{left} and v_{right}.

Since the three vertices u, u_{left} and u_{right} lie on the cycle c_u, there exists two disjoint paths: first from u to u_{left} and second from u to u_{right}. Similarly, there exists two disjoint paths from v_{right} to v and from v_{left} to v. We can thus get two vertex-disjoint paths from u to v, using these. Note that there may be other floor and ceiling cycles intersecting these disjoint paths. In that case, we can use an argument similar to above to show the existence of two disjoint paths from u to v.

3. We now prove that the size of each face of $\mathsf{mframe}(G)$ is $O(k^{1/2})$. Note that the boundary of a face of the graph $\mathsf{mframe}(G)$ is one of the following:

a. A floor-cycle of G.

b. A ceiling-cycle of G.
c. A branch-triangle of G.
d. A frame-cycle of $\mathsf{frame}(G)$ modified by floor-cycles and ceiling-cycles.

In the first three cases, the size bound of the face follows by definition. We thus consider the fourth case.

Consider any face defined by a modified frame-cycle, and let c denote the pre-frame-loop from which we have defined it. Consider any path p of c connecting a boss-vertex of c and a vertex of a branch-triangle used in c such that the path does not contain any other vertex of a branch-triangle. By our modification, we can use a part p' of p that is in the outside of the corresponding floor-cycle and ceiling-cycle (if it exists) as a component of the modified frame-cycle, and its length is bounded by $4k^{1/2}$. Note that the floor-cycle may not be used in $\mathsf{mframe}(G)$ if it only intersects with the darts that we have removed for defining the face. In this case, however, only a part of p' is used for the modified frame-cycle, which is even shorter. Thus, the modified frame-cycle consists of at most four such reduced paths, a part of two floor-cycles, a part of four ceiling-cycles, and two edges from two branch-triangles, and their total length is $O(k^{1/2})$.

4. We first prove that the number of connectors is $O(n/k+g)$, and the number of branch vertices is $O(n/k+g)$. Since there is a one-to-one correspondence respectively between branch-triangles and branch vertices, and between frame-cycles and connectors, it will follow that the number of faces in $\mathsf{frame}(G)$ is $O(n/k+g)$.

We first define a new graph G'. The vertex set of G' is the set of branch vertices in G. We add an edge between two vertices of G' if they are adjacent pair of branch vertices. Since every edge of G' corresponds to a connector of G, the graph G' can also be embedded on the surface of genus g where the embedding corresponds to the embedding of G. Let n', e', f' be the number of vertices, edges and faces in G' respectively. Thus, we have $n'-e'+f'=2-2g$ by Euler's formula. Note that every branch vertex have a degree 3, therefore we have $2e'=3n'$. This implies $e'=6g+3f'-6=O(f'+g)$. Since, there is one-to-one correspondence between voronoi regions and the faces of G', we have $f'=O(n/k)$. Hence, we can conclude that $e'=O(n/k+g)$ and $n'=O(n/k+g)$.

Now, to prove that the number of faces in $\mathsf{mframe}(G)$ is $O(n/k+g)$, we see that the new faces introduced by our modification are those defined by floor cycles or ceiling cycles. By definition, the number of these cycles is at most the number of boss-vertices or that of branch-triangles, which is bounded by $O(n/k+g)$. Note that we can divide a face defined by a frame-cycle of $\mathsf{frame}(G)$ by ceiling-cycles, but it is easy to see that each face is divided into at most some constant number of faces because the number of floor-cycles and ceiling-cycles overlapping each frame-cycle is constant, say, at most six. From these observations, we can bound the number of faces of $\mathsf{mframe}(G)$ by $O(n/k+g)$. ◀

Near-Optimal Cayley Expanders for Abelian Groups

Akhil Jalan[1]
Department of Computer Science, University of Texas at Austin, TX, USA

Dana Moshkovitz
Department of Computer Science, University of Texas at Austin, TX, USA

Abstract

We give an efficient deterministic algorithm that outputs an expanding generating set for any finite abelian group. The size of the generating set is close to the randomized construction of Alon and Roichman [9], improving upon various deterministic constructions in both the dependence on the dimension and the spectral gap. By obtaining optimal dependence on the dimension we resolve a conjecture of Azar, Motwani, and Naor [14] in the affirmative. Our technique is an extension of the bias amplification technique of Ta-Shma [40], who used random walks on expanders to obtain expanding generating sets over the additive group of $\mathbb{F}_2^n$. As a consequence, we obtain (i) randomness-efficient constructions of almost k-wise independent variables, (ii) a faster deterministic algorithm for the Remote Point Problem, (iii) randomness-efficient low-degree tests, and (iv) randomness-efficient verification of matrix multiplication.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization; Theory of computation → Error-correcting codes

Keywords and phrases Cayley graphs, Expander walks, Epsilon-biased sets, Derandomization

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.24

Related Version *Full Version*: https://arxiv.org/abs/2105.01149 [29]

Funding This material is based upon work supported by the National Science Foundation under grants number 1218547 and 1678712.

1 Our Contributions

1.1 Main Result

A graph is an expander if there exists a constant $\alpha > 0$ such that the spectral gap of its adjacency matrix (namely, the difference between its top eigenvalue and its second eigenvalue) is at least α. Such graphs are very well-connected in the sense that they lack sparse cuts. Expanders that are additionally sparse are immensely important in computer science and mathematics (see, e.g. the survey [28]).

Cayley graphs are an important class of graphs built from groups. Given a group G and a generating set $S \subset G$, the graph $\mathrm{Cay}(G, S)$ has vertex set G and edges $(g, g \cdot s)$ for all $g \in G$, $s \in S$. In addition to describing various well-known graphs such as the hypercube and the torus, Cayley graphs of (non-abelian) groups gave the first explicit constructions of near-optimal expander graphs [34]. Moreover, their algebraic structure makes Cayley graphs easier to analyze. In particular, the eigenvectors and eigenvalues of a Cayley graph are well-understood through the Fourier transform on the group.

When is a Cayley graph an expander? Alon and Roichman showed that given a group G, integer $n \geq 1$, and $\epsilon > 0$, taking a uniformly random subset $S \subset G^n$ of size $O(\frac{n \log(|G|)}{\epsilon^2})$ gives an expander with spectral gap $1 - \epsilon$, with high probability [9]. They also proved a

[1] Corresponding author.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 24; pp. 24:1–24:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

nearly matching lower bound of $|S| = \Omega((\frac{n\log(|G|)}{\epsilon^2})^{1-o(1)})$ when G is abelian. When $G = \mathbb{F}_2$ the lower bound is $\Omega(\frac{n}{\epsilon^2\log(1/\epsilon)})$ [6] [2].

An explicit construction with parameters matching the Alon-Roichman bound has remained elusive, despite being widely studied in the pseudorandomness literature [32, 35, 6, 36, 1, 7, 26, 14, 23, 12, 17, 11].

The best known results achieve $O((\log(|G|) + \frac{n^2}{\epsilon^2})^5)$ for arbitrary abelian G [12], $O(\frac{n^2}{\epsilon^2})$ for abelian G where $|G| \leq \log(\frac{n^2}{\epsilon^2})^{O(1)}$, and $O(\frac{n\log(|G|)^{O(1)}}{\epsilon^{11}})$ for general G [23]. For solvable subgroups of permutation groups one can improve this to $O(\frac{n^2}{\epsilon^8})$ [11].

In this paper we give an explicit construction of expanding generating sets for abelian groups whose size is near the Alon-Roichman bound.

▶ **Theorem 1.** *There is a deterministic, polynomial-time algorithm which, given a generating set of an abelian group G, integer $n \geq 1$, and $\epsilon > 0$, outputs a generating set $S \subset G^n$ of size $O(\frac{n\log(|G|)^{O(1)}}{\epsilon^{2+o(1)}})$ such that $\mathrm{Cay}(G^n, S)$ has spectral gap $1-\epsilon$.*

Our construction immediately improves parameters in several applications – see Section 1.3 for details. We remark that in most settings, one fixes a group G while $n \to \infty$ and $\epsilon \to 0$. In this regime, since $|G|$ is a constant, the size of the generating set in Theorem 1 is optimal up to an $\epsilon^{-o(1)}$ factor. The $o(1)$ term in the exponent approaches 0 as $\epsilon \to 0$.

Expanding Cayley graphs are equivalent to pseudorandom objects called ϵ-biased sets. These were originally defined over $\mathbb{F}_2^n$ by Naor and Naor [35]. A set $S \subseteq \mathbb{F}_2^n$ is said to be ϵ-biased if for every non-empty $T \subseteq [n]$, we have $\mathop{\mathbb{E}}_{x\in S}[\bigoplus_{i\in T} x_i] = 1/2 \pm \epsilon$.

Naor and Naor initiated a long line of work culminating in a recent breakthrough result by Ta-Shma, that achieves $|S| = O(\frac{n}{\epsilon^{2+o(1)}})$ [40]. This construction approaches the Alon-Roichman bound as $\epsilon \to 0$.

Ta-Shma's construction follows previous work in using a 2-step "bias amplification" approach. First, identify an explicit set $S_0 \subset \mathbb{F}_2^n$ with constant bias, usually through algebraic methods. Second, amplify the bias of S_0 to any $\epsilon > 0$ by performing a random walk on an expander graph. While this general method was already known, it could only achieve $|S| = O(\frac{n}{\epsilon^{4+o(1)}})$. To break this barrier, Ta-Shma identified a graph structure obtained from a "wide replacement product", which was more effective for the bias amplification step and resulted in $|S| = O(\frac{n}{\epsilon^{2+o(1)}})$.

Our main contribution is to show that the wide replacement walk is a near-optimal "character sampler," and therefore also amplifies bias well for abelian Cayley graphs.

1.2 Wide Replacement Walks are Near-Optimal Character Samplers

Random walks on expander graphs are useful for a variety of algorithmic purposes. A classical fact is that expander walks are good approximate samplers, in the sense that a sufficiently long random walk on an expander will visit sets of density δ for approximately a δ fraction of the steps. This is called the "expander Chernoff bound" and one can characterize this as the property that expander walks fool a suitable test function.

Ta-Shma observed that expander walks fool the much more sensitive class of parity functions on $\{0,1\}^n$ as well. Parity functions are sensitive to input perturbations – flipping a single bit in the input can change the output. The classical expander Chernoff bound is

[2] It is possible that this lower bound is tight. A candidate construction based on algebraic-geometric codes could achieve this lower bound [17].

not fine-grained enough to prove that t-step expander walks fool parity functions. The fact that they nevertheless do fool parity functions is therefore surprising, and Ta-Shma referred to this fact as "expanders are good parity samplers" [40].

Since parity functions are just the characters of $\mathbb{F}_2^n$, we can ask: do expander walks also fool the characters of more general classes of groups? We show that this is indeed true, and therefore "expander walks are good character samplers." Moreover, just as in the $\mathbb{F}_2$ case, a random walk on a wide replacement product of expander graphs is a near-optimal type of character sampler.

Character sampling explained. Let us precisely explain what we mean by "character sampling." A character of an abelian group is a homomorphism $\chi : G \to \mathbb{C}^*$, where $\mathbb{C}^*$ is the multiplicative group of complex numbers. The eigenvalues of an abelian Cayley graph $\mathrm{Cay}(G, S)$ are given by $|\mathbb{E}_{x\sim S}\,\chi(x)|$ for all characters χ. Note that the constant function that maps all values to 1 is a character, and the eigenvalue associated with it is the top eigenvalue. Therefore, we are interested in generating sets S such that $|\mathbb{E}_{x\sim S}\,\chi(x)| \leq \epsilon$ for all non-constant χ.

For simplicity, consider the case $G = \mathbb{Z}_d$ for some $d \geq 2$. Let $\omega_d := \exp(\frac{2\pi i}{d})$. In this case the characters are just the maps $x \mapsto \omega_d^{x\cdot j}$ for $j = 0, 1, \ldots, d-1$.

Now, suppose we have some ϵ_0-biased set $G_0 \subset G$, where $\epsilon_0 < 1$ is a constant. First, observe that taking t *independent* samples from G_0 and outputting their sum obtains a distribution with bias $(\epsilon_0)^t$. However, since independent sampling also results in a distribution with support size $|G_0|^t$, there is no improvement in size as a function of bias.

The idea of the random walk approach is to derandomize independent sampling by taking *correlated* samples. Specifically, identify G_0 with the vertices of some degree-regular expander graph Γ. We need to show that taking a random walk of length t on Γ and then summing the elements in the path gives a distribution with lower bias than G_0.

A t-step walk on Γ gives a sequence of group elements $(x_0, \ldots, x_t) \in G_0^{t+1}$. We are interested in the bias of the random group element $\sum_i x_i$. In general, we cannot hope that $(\sum_i x_i)$ is close to the uniform distribution in *statistical distance*. However, if Γ is an expander with second eigenvalue λ, then for every non-constant character χ the quantity $|\mathbb{E}[\chi(\sum_i x_i)]|$ is at most $(\epsilon_0 + \lambda)^{\lfloor t/2 \rfloor}$, where the expectation is over paths $(x_0, \ldots, x_t)$ in the graph. Notice that $\mathbb{E}_{x\in G}[\chi(x)] = 0$, so the random element $(\sum_i x_i)$ is close to uniform in the weaker sense of fooling characters. Therefore, the expander walk is a good "character sampler."

Why expanders are character samplers. We express the bias of the random walk distribution algebraically in terms of matrix norms corresponding to the random walk.

Abusing notation, let Γ denote the random walk matrix of the graph Γ. Let the character $\chi^* : \mathbb{Z}_d \to \mathbb{C}$ be the worst-case character for the random-walk distribution. Partition G_0 into $S_0, \ldots, S_{d-1}$ depending on their values with respect to χ^*, so that $x \in S_k \iff \chi^*(x) = \omega_d^k$.

We need to track how often the walk enters $S_0, S_1, \ldots, S_{d-1} \subset V(\Gamma)$. Identify each S_i with an $|S_i|$-dimensional subspace of $\mathbb{C}^{V(\Gamma)}$. For $i \in \mathbb{Z}_d$ let $\Pi_i : \mathbb{C}^{V(\Gamma)} \to \mathbb{C}^{V(\Gamma)}$ be the projection onto this subspace. Finally, let $\Pi = \sum_{y\in\mathbb{Z}_d} \omega_d^y \Pi_y$ be the weighted projection matrix.

Given some initial distribution $\vec{u}$ on the vertices, the vector $\Gamma^t \vec{u}$ tracks the distribution after taking a t-step walk on the graph. The matrix Π tracks how often the walk enters the sets $S_0, \ldots, S_{d-1}$, and so the bias of the random walk distribution can be bounded by the norm of $(\Pi\Gamma)^t$.

Let $V^{\|}$ denote the subspace spanned by the all-ones vector $\vec{1}$, and $V^{\perp} = (V^{\|})^{\perp}$. For a vector $v \in V^{\|} \oplus V^{\perp}$, let $v^{\|}$ and $v^{\perp}$ denote the projections onto $V^{\|}, V^{\perp}$ respectively.

While $\|\Pi\Gamma\| = 1$ since $\|\Pi\Gamma\vec{1}\| = \|\Pi\vec{1}\| = 1$, it turns out that $\|(\Pi\Gamma)^2\| \leq bias(G_0) + 2\lambda(\Gamma)$, where $\lambda(\Gamma)$ is the second eigenvalue of Γ in absolute value.

To see this, notice that if $\vec{v} \in V^{\perp}$ is a unit vector, then $\|\Pi\Gamma\Pi\Gamma\vec{v}\| \leq \|\Pi\Gamma\Pi\|\lambda(\Gamma)\|\vec{v}\| \leq \lambda(\Gamma)$. Therefore, the "bad" case is when $\vec{v} \in V^{\|}$. Let $u = \frac{1}{\sqrt{|V(\Gamma)|}}\vec{1}$. Using the fact that $\|\Pi\| = 1$,

$$\begin{aligned}\|\Pi\Gamma\Pi\Gamma u\| &= \|\Pi\Gamma\Pi u\| \\ &\leq \|\Pi\Gamma(\Pi u)^{\|}\| + \|\Pi\Gamma(\Pi u)^{\perp}\| \\ &\leq \|\Pi(\Pi u)^{\|}\| + \lambda(\Gamma)\|\Pi(\Pi u)^{\perp}\| \\ &\leq \|\Pi(\Pi u)^{\|}\| + \lambda(\Gamma)\end{aligned}$$

It remains to show that $\|\Pi(\Pi u)^{\|}\| \leq bias(G_0)$. To see this, notice that Π is a diagonal matrix and u is just $\vec{1}$ scaled by a constant. Further, Π is a block-diagonal matrix of the form

$$\Pi = \begin{bmatrix} I_{|S_0|} & & & \\ & \omega_d I_{|S_1|} & & \\ & & \ddots & \\ & & & \omega_d^{d-1} I_{|S_{d-1}|} \end{bmatrix}$$

Note that we have reordered the vertices of the graph in order of S_0, S_1 and so on.

If the blocks are exactly the same size, then $\Pi u \in V^{\perp}$, because $\sum_{y \in \mathbb{Z}_d} \omega_d^y = 0$. In general the blocks have different dimensions, but they are the same size up to the bias of G_0. Therefore $\|(\Pi u)^{\|}\| \leq bias(G_0)$.

It follows that a random walk on Γ is a good character sampler. However, this approach can never amplify bias fast enough to achieve a generating set smaller than $O(\frac{|G_0|}{\epsilon^{4+o(1)}})$. The reason is because while we can bound $\|(\Pi\Gamma)^2\|$, we cannot bound $\|\Pi\Gamma\|$ below 1. Therefore, we effectively only gain from one in every two steps.

Wide Replacement Walks are Near-Optimal Character Samplers. To circumvent the "2-step barrier" of expander walks outlined above, Ta-Shma used the *wide replacement walk* on a product of two expander graphs [40]. The idea of the wide replacement walk is to take the product of a D_1-regular graph Γ as before with an "inner graph" H on D_1^s vertices, for some $s \geq 2$. The product graph replaces every vertex of Γ with a copy of H (called a "cloud") and then connects clouds to other clouds according to the edge structure of Γ.

Analyzing the bias of the walk involves bounding the matrix norm of $\dot{\Pi}\dot{\Gamma}\dot{H}$, where $\dot{\Gamma}$ and $\dot{H}$ are random walk matrices on the product corresponding to Γ, H.

Let $V^{\|}$ denote the subspace of vectors which are constant on the H-component of the product, and let $V^{\perp} = (V^{\|})^{\perp}$.

Similar to the above case, one can show that $\dot{\Pi}\dot{\Gamma}\dot{H}$ shrinks the norm of any $v \in V^{\perp}$ by a factor of $\lambda(H)$. The difficult case is when $v \in V^{\|}$. Here we arrive at the core idea of the replacement product: if the inner graph H is *pseudorandom* with respect to Γ, then when the walk is in $V^{\|}$, the next s steps approximate the ordinary random walk on Γ.

This is enough to circumvent the "2-step barrier" since in even the "bad case" where the walk is stuck in $V^{\|}$, we can shrink the bias as though it were taking an ordinary walk on Γ. As we showed above, this shrinks the bias from some ϵ_0 to $(\epsilon_0 + 2\lambda(\Gamma))^{\lfloor s/2 \rfloor}$ every s steps. If we select Γ, H such that $\epsilon_0 + 2\lambda(\Gamma) \leq \lambda(H)^2$, then we conclude that we shrink the bias by a factor of $\lambda(H)^{s-O_s(1)}$ every s steps. So we gain from $s - O(1)$ out of every s steps.

Going from the $\mathbb{F}_2$-case to the case of general abelian groups simply requires a more careful analysis of characters. We defer the full proof to Appendix 2.2.

Morally speaking, the only difference in the analysis is that the projection matrix Π which tracks how often the walk enters each S_i is different. This does not change the overall argument much; in particular, we can use almost identical graphs Γ, H as in [40].

We conclude that a wide replacement walk allows us to amplify bias of a constant-biased subset $G_0 \subset G^n$ of size $O(n \log(|G|)^{O(1)})$ (e.g. the construction of [11]) to an ϵ-biased set of size $O(\frac{n \log(|G|)^{O(1)}}{\epsilon^{2+o(1)}})$, nearly matching the Alon-Roichman bound. For explicit parameters of the construction, see Appendix C.

1.3 Applications

Explicit constructions of expander graphs are an essential component of algorithms, especially for derandomization. Here we are interested in the setting of constructing an expanding Cayley graph from a given abelian group G. Our construction achieves a near-optimal degree, which improves parameters in various applications. We defer precise statements of these results and the full proofs to the full version.

Almost k-wise independence. A distribution $D \sim G^n$ is (ϵ, k)-wise independent if for every index set $I \subset [n]$ of size k, the restriction of D to I is ϵ-close to uniform in statistical distance. Almost k-wise independent distributions are a fundamental object in and of themselves. They also have a variety of applications in derandomization, including load balancing [24], derandomization of Monte-Carlo simulations [24], derandomization of CSP approximation algorithms [21], and pseudorandom generators [22]. We note that certain applications (e.g. quantum t-designs [10]) really require almost k-wise independent distributions over *arbitrary* alphabet size rather than just the binary alphabet, which motivates our study of ϵ-biased sets over arbitrary abelian groups.

Vazirani's XOR Lemma asserts that an ϵ-biased distribution D is also $(\epsilon\sqrt{|G|^k}, k)$-wise indepdent for all $k \leq n$. Therefore, by constructing an ϵ'-biased distribution where $\epsilon' = \frac{\epsilon}{\sqrt{|G|^k}}$, we also obtain explicit constructions of (ϵ, k)-wise independent random variables on G^n.

▶ **Proposition 2** (Almost k-wise independent sets over abelian groups). *Let G be a finite abelian group given by some generating set. For any $\epsilon > 0$ and $n \geq k \geq 1$ there exists a deterministic, polynomial-time algorithm whose output is an (ϵ, k)-wise independent distribution over G^n. The support size is $O(\frac{n \cdot |G|^{k+o(1)}}{\epsilon^{2+o(1)}})$.*

Remote Point Problem. A matrix $A \in \mathbb{F}_2^{m \times n}$ is (k, d)-rigid iff for all rank-k matrices $R \in \mathbb{F}_2^{m \times n}$, the matrix $A - R$ has a row with at least d nonzero entries. Valiant initiated the study of rigid matrices in circuit complexity, proving that an explicit construction of an $(\Omega(n), n^{\Omega(1)})$-rigid matrix for $m = O(n)$ would imply superlinear circuit lower bounds [43]. After more than four decades of research, state of the art constructions have yet to meet this goal [19].

The Remote Point Problem was introduced by Alon, Panigrahy, and Yekhanin as an intermediate problem in the overall program of rigid matrix constructions [8]. Arvind and Srinivasan generalized the problem to any group [12].

Let G be a group, $n \geq 1$, and $H \leq G^n$ a subgroup given by some generating set. For a given G, H and integer $r > 0$, the Remote Point Problem is to find a point $x \in G^n$ such that x has Hamming distance greater than r from all $h \in H$, or else reject. In the case of

$G^n = \mathbb{F}_2^n$, this is a relaxation of the matrix rigidity problem, since rather than finding m vectors $x_1, \ldots, x_m \in \mathbb{F}_2^n$ whose linear span is far from all low-dimensional subspaces, we are given a single subspace and must find just a single point far from it.

To find a remote point, existing algorithms first construct a collection of subgroups $H_1, \ldots, H_m \leq G^m$ whose union covers all points of distance at most r from H. In the $\mathbb{F}_2$ case, [8] find a point $x \notin \bigcup_i H_i$ by the method of pessimistic estimators. In the general case, [12] instead prove that any generating set $S \subset G^n$ such that $\mathrm{Cay}(G^n, S)$ has sufficiently good expansion must contain a point outside of $\bigcup_i H_i$. They find this remote point by first constructing an expanding generating set S, and then exhaustively searching it. Their argument implicitly uses the fact that small-bias sets correspond to rigid matrices, albeit with weak parameters - this connection was developed further in [5].

The construction of [12] for small-bias sets over abelian groups has size $O((\log(|G|) + \frac{n^2}{\epsilon^2})^5)$ in general, and for $\log(|G|) \leq \log(\frac{n^2}{\epsilon^2})^{O(1)}$ this is improved to $O(\frac{n^2}{\epsilon^2})$. Our algorithm improves the dependence on n from n^2 to n.

Randomness-Efficient Low-Degree Testing. Let $\mathbb{F}_q$ be the finite field on q elements. Low-degree testing is a property testing problem in which, when given query access to a function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ and $d \geq 1$, one must decide whether f is a degree d polynomial or far (in Hamming distance) from all degree d polynomials. These tests are a key ingredient in constructions of Locally Testable Codes (LTCs) and Probabilistically Checkable Proofs (PCPs) [18].

To test whether f is a degree-d polynomial, a natural test is to sample $x, y \sim \mathbb{F}^n$ and check whether $f(x)$ agrees with the unique (degree-d, univariate) polynomial obtained by Lagrange interpolation along $d+1$ points on the line $\{x + ty : t \in \mathbb{F}_q\}$.

Rubinfeld and Sudan introduced a low-degree test using this idea [38]. It is given query access to the function f, along with a *line oracle* function g. Let $\mathbb{L}$ denote all lines $\{\vec{a} + t\vec{b} : t \in \mathbb{F}_q\} \subset \mathbb{F}_q^n$, where $\vec{a}, \vec{b} \in \mathbb{F}^n$. Given a description of a line, the line oracle g returns a univariate polynomial of degree d defined on that line. Hence we write $g : \mathbb{L} \to \mathbb{F}_q[t]$, where the image of g is understood to only contain degree-d polynomials.

If f is indeed a degree-d polynomial, then one can set $g(\ell) = f|_\ell$ for all $\ell \in \mathbb{L}$, and the following two-query test clearly accepts.

(i) Select $x, y \in \mathbb{F}^n$ independently, uniformly at random.
(ii) Let ℓ be the line determined by $\{x + ty : t \in \mathbb{F}\}$. Accept iff $f(x)$ agrees with $g(\ell)(x)$.

They also showed this test is sound: when f is far from degree-d polynomials, the test rejects with high probability.

Ben-Sasson et al derandomized this test by replacing the second uniform sample y with a sample from an ϵ-biased set [18]. This modification improves the randomness efficiency of the tests, and therefore the length of the resulting LTC and PCP constructions. Moreover, they showed that the soundness guarantees of low-degree tests are almost unchanged due to the expansion properties of the Cayley graph on $\mathbb{F}_q^n$.

Our constructions of small-bias sets immediately imply improved randomness-efficiency of this low-degree test.

▶ **Proposition 3** (Improved [18] Theorem 4.1). *Let $\mathbb{F}_q$ be the finite field of q elements, $n \geq 1$, $f : \mathbb{F}_q^n \to \mathbb{F}_q$ a function, and $g : \mathbb{L} \to \mathbb{F}_q[t]$ a line oracle. There exists a degree-d test which has sample space size $O(q^n \cdot \frac{n \log(q)^{O(1)}}{\epsilon^{2+o(1)}})$. For $d \leq q/3$ and sufficiently small $\delta > 0$, if the test accepts with probability $\geq 1 - \delta$ then f has Hamming distance at most 4δ from a degree d polynomial.*

Randomness-Efficient Verification of Matrix Multiplication. Let R denote some finite field $\mathbb{F}_q$ or cyclic group $\mathbb{Z}_q$ for $q \geq 2$. Given $A, B, C \in R^{n \times n}$, the matrix multiplication verification problem asks whether $AB = C$.

Naively, one could multiply A, B and then check whether $AB = C$ entry-wise in $O(n^\omega)$ time, where $\omega \approx 2.373$ [2]. A classical result of Freivalds suggests the following much simpler quadratic-time randomized algorithm: Sample $x \in R^n$ and check whether $ABx = Cx$ [27].

Observe that the entries of ABx and Cx are linear functions of x. Therefore, sampling x from a small-bias set gives a randomness-efficient version of Freivalds' algorithm, at the cost of slightly higher error. Our construction therefore gives the following randomness efficient algorithm for verification of matrix multiplication.

▶ **Proposition 4.** *Let R denote a finite field $\mathbb{F}_q$ or cyclic group $\mathbb{Z}/q\mathbb{Z}$. Given matrices $A, B, C \in R^{n \times n}$ and ϵ-biased set $S \subset R^n$, there exists randomized algorithm to decide whether $AB = C$ with one-sided error $(\frac{1}{q} + \epsilon)$. Its runtime is $O(n^2)$ and it uses $\log(\frac{n \log(q)^{O(1)}}{\epsilon^{2+o(1)}})$ random bits.*

We note that if $R = \mathbb{Z}$, there exists a deterministic $O(n^2)$ time algorithm to verify matrix multiplication [33]. However, this result relies on the fact that $\mathbb{Z}$ has characteristic zero. For the analysis to hold in the case of $\mathbb{Z}_q$, we would need a very strong bound on the entries of A, B, C – namely, that $\max_{i,j}\{|A_{i,j}|, |B_{i,j}|, |C_{i,j}|\} \leq q^{\frac{1}{n-1}}$.

1.4 Related Work

Explicit Constructions. Explicit constructions of expanding generating sets for Cayley graphs have been mostly studied in the pseudorandomness literature in the context of small-bias sets for derandomization. Naor and Naor gave a combinatorial construction over $\mathbb{F}_2^n$ of size $O(\frac{n}{\epsilon^3})$ [35]. Alon, Goldreich, Hastad, and Peralta used algebraic arguments to give constructions over finite fields $\mathbb{F}^n$ of size $O(\frac{n^2}{\epsilon^2})$, assuming the field size is bounded as $\log(|\mathbb{F}|) < \frac{n}{\log(n)+\log(1/\epsilon)}$ [6].

Resarchers in various communities have obtained constructions that achieve size $O(\text{poly}(\frac{n \log(|G|)}{\epsilon}))$, but suboptimal exponents. In number theory and additive combinatorics researchers studying the case of $n = 1$ gave constructions over $\mathbb{Z}_d$ of size $O((\frac{\log(d)}{\epsilon})^{O(1)})$ [36], $O(\frac{\log(d)^{O(1)}}{\epsilon^2})$ [32], and $O(\frac{d}{\epsilon^{O(\log^*(d))}})$ [1].

Other constructions equivalent to small-bias sets include $O(\frac{(n-1)^2}{\epsilon^2})$-sized ϵ-discrepancy sets over finite fields of prime order p when $n \leq p$ [7], and ϵ-balanced codes over finite fields, corresponding to small-bias sets over $\mathbb{F}_q^n$ of size $O(n \cdot q)$ with constant bias [31].

Ta-Shma's tour de force gave the first explicit construction of expanding generating sets of size $O(\frac{n \log(|G|)}{\epsilon^{2+o(1)}})$, nearly attaining the Alon-Roichman bound, but only for the special case of $G = \mathbb{F}_2$ [40]. Our work is an extension of Ta-Shma's bias amplification technique to the more general setting of arbitrary abelian groups.

Azar, Motwani, and Naor generalized the study of small-bias sets to finite abelian groups [14]. Over $\mathbb{Z}_d^n$ they used character sum estimates to give a construction of size $O((d + \frac{n^2}{\epsilon^2})^C)$, where $C \leq 5$ is Linnik's constant [45]. Assuming the Extended Riemann Hypothesis, $C \leq 2 + o(1)$ [15]. When $\log(d) \leq \log(\frac{n^2}{\epsilon^2})^{O(C)}$ they improve the size to $O((1 + o(1))\frac{n^2}{\epsilon^2})$.

Arvind and Srinivasan proved that one can project small-bias sets over $\mathbb{Z}_d^n$ to any abelian group G^n when d is the largest invariant factor of G. Therefore, using the construction from [14] they obtain small-bias sets over G^n with the same bias and size as [14], with $d = O(\log(|G|))$ [12].

The most general setting is to consider Cayley graphs over non-abelian groups. Wigderson and Xiao derandomized the Alon-Roichman construction using the method of pessimistic estimators [44]. Arvind, Mukhopadhyay, and Nimbhorkhar later gave a derandomization for both directed and undirected Cayley graphs using Erdos-Renyi sequences [13]. However, both algorithms require the entire group table of G^n as input, rather than just a generating set. Since generating sets are of size $O(n\log(|G|))$, these algorithms are exponentially slower, running in time $O(\text{poly}(|G|^n))$ rather than $O(\text{poly}(n\log(|G|))$. Nevertheless, they have applications to settings such as homomorphism testing [39], which Wigderson and Xiao derandomized using their construction of expanding generating sets [44].

Chen, Moore, and Russell obtained generating sets of size $O(\frac{n\log(|G|)^{O(1)}}{\epsilon^{11}})$ over arbitrary groups G^n when $|G|$ is a constant [23] . Like Ta-Shma, their technique is to use bias amplification via expander graphs; specifically, they amplify bias via an iterated application of a 1-step random walk on an expander graph. Alon in 1993, and later Rozenman and Wigderson in 2004, had already noted that this technique amplifies bias for $G = \mathbb{F}_2$ [25]. Chen, Moore, and Russell generalized this analysis to all groups, using techniques from harmonic analysis and random matrix theory [23].

Existing work seems far from obtanining constructions for non-abelian groups near the Alon-Roichman bound. Known work tends to concentrate on special classes of non-abelian groups with some useful algebraic structure. Chen, Moore, and Russell constructed generating sets of size $O(\frac{(n\log(|G|))^{1+o(1)}}{\epsilon^{O(1)}})$ for smoothly solvable groups with constant-exponent abelian quotients [23]. Their analysis exploits the structure of solvable groups via Clifford theory. It also hinges on the assumption that the quotients in the derived series have constant exponent.

Arvind et al later gave a construction of size $\tilde{O}(\frac{\log(|G|)^{2-o(1)}}{\epsilon^8})$ for solvable subgroups G of permutation groups [11]. Their construction recursively generates expanding generating sets for quotients in the derived series of the group, and uses the thin sets construction of [1] as a base set. Unlike [23] they do not require successive quotients of the derived series to be small; however, their argument does rely on an $O(\log(n))$ upper bound on the length of the derived series for any solvable $G \leq S_n$, which is not true for solvable groups in general.

Lower Bounds. Alon and Roichman gave a randomized upper bound of $O(\frac{n\log(|G|)}{\epsilon^2})$ on the size of a generating set for any finite G^n with spectral gap $(1-\epsilon)$ [9]. In the same paper, they gave a nearly matching lower bound when G is abelian, of $\Omega((\frac{n\log(|G|)}{\epsilon^2})^{1-o(1)})$. This is a sharper version of the folklore result that an abelian group G^n requires $O(n\log(|G|))$ generators for its Cayley graph to be connected.

For non-abelian groups, the existence of sparse expanders means the best lower bound in general is the Alon-Boppana bound. This removes the dependence on $|G|$ and n, only requiring a generating set of size $\Omega(\frac{1}{\epsilon^2})$ [3] to achieve spectral gap of $1-\epsilon$. Indeed, explicit constructions of Ramanujan graphs can be built from Cayley graphs of non-abelian groups [34], and therefore attain this bound.

Expander Walks. Random walks on expander graphs are an essential tool in computer science. Rather than surveying the vast literature, we refer the reader to the surveys [28, 42]. Two remarks are in order.

First, our use of wide replacement walks is essentially a way of building expander graphs from other expander graphs. This is thematic of several previous works, such as the zig-zag product [37]. Note that the zig-zag product is just a modification of the replacement product; indeed, the (wide) replacement product itself can be used to give explicit, combinatorial constructions of Ramanujan graphs [16]. Ta-Shma used wide replacement walks to amplify spectral gaps of Cayley graphs on $\mathbb{F}_2^n$ [40]; this construction relied on previous constructions of expander graphs, although the expander graphs were not required to be Cayley graphs themselves.

Second, the fact that "expanders are good character samplers" is surprising given that characters are sensitive to input perturbations. A recent work of Cohen, Peri, and Ta-Shma uses Fourier-analytic techniques to classify a large class of Boolean functions which can be fooled by expander walks, including all symmetric Boolean functions [25].

1.5 Open Problems

In this work, we gave an efficient deterministic algorithm to compute an expanding generating set of an abelian group. Our construction achieves optimal dependence on dimension and near-optimal dependence on error, resulting in improvements in various applications. Here, we discuss some natural open questions raised by our work.

Expanding generating sets of optimal size. The Alon-Roichman theorem proves that every group G^n has an expanding generating set $S \subset G^n$ of size $|S| = O(\frac{\log(|G|)}{\epsilon^2})$ [9]. This construction has not been fully derandomized for any group; even in the case of $G^n = \mathbb{F}_2^n$, Ta-Shma's construction only asympotically approaches a size of $O(\frac{n}{\epsilon^2})$ as $\epsilon \to 0$. The actual size of the generating set is $O(\frac{n}{\epsilon^{2+o(1)}})$, and this $o(1)$ term is seemingly unavoidable when using expander walks [40].

Similarly, our algorithm gives an expanding generating $S \subset G^n$ of size $O(\frac{n \log(|G|)^{O(1)}}{\epsilon^{2+o(1)}})$, for finite abelian G. The additional poly $\log(|G|)$ factor comes from the bounds on constant-bias subsets of abelian groups; any construction of a constant-bias set $S \subset G^n$ of size $O(n \log(|G|))$ would immediately give expanding generating sets of size $O(\frac{n \log(|G|)}{\epsilon^{2+o(1)}})$. To our knowledge, not even a candidate construction exists which would give constant-bias subsets of size $O(n \log(|G|))$ for abelian groups; this is an interesting and potentially easier open problem, since it requires none of the expander walks machinery that we need to get arbitrarily small ϵ.

There is a candidate construction that could beat the Alon-Roichman bound for $G = \mathbb{F}_2$, based on algebraic-geometric codes [17]. The code construction would give an ϵ-biased set $S \subset \mathbb{F}_2^n$ of size $|S| = O(\frac{n}{\epsilon^2 \log(1/\epsilon)})$, assuming a conjecture in algebraic geometry. The authors themselves note that they have "no idea" whether this conjecture is valid [17].

Expanding generating sets of non-abelian groups. While wide replacement walks amplify bias quite naturally for abelian groups, it is unclear whether they can do so for general groups. Dealing with matrix-valued irreducible representations, rather than scalar-valued characters, makes the analysis of bias amplification considerably more involved; hence even the analysis of the 1-step walk is nontrivial [23]. It would be very interesting to see whether one can place algebraic conditions on a group that are weaker than commutativity, but still ensure that the wide replacement walk amplifies bias.

Existing works on expanding generating sets for non-abelian groups have studied solvable groups, which generalize abelian groups [23, 11]. However, if we restrict the algorithm to input instances which are all non-abelian groups, then existence results suggest that one should be able to *beat* the Alon-Roichman bound.

For example, it is known that for every finite *simple* non-abelian group G^n, there exists a generating set $S \subset G^n$ such that $\mathrm{Cay}(G^n, S)$ has spectral gap $1 - \epsilon$, and $|S|$ is independent of n [20]. Therefore, restricting input instances to simple groups seems too easy, while an algorithm for all groups seems too hard. Is there some natural natural class of non-abelian, non-simple groups for which algorithms can efficiently find expanding generating sets near (or even below) the Alon-Roichman bound?

Decoding over any finite field. A recent work of Jeronimo et al gives a decoding algorithm for a modified version of Ta-Shma's codes [30]. Since our work gives ϵ-balanced codes over any finite field, it would be interesting to extend both the modification of the codes and the decoding algorithm of [30] to this general setting.

Classifying the power of expander walks on groups. So far we have discussed how random walks on expanders are good samplers in various ways, such as the expander Chernoff bound, parity sampling, and character sampling. Cohen, Peri, and Ta-Shma study the class of all Boolean functions that expander walks fool [25]. It would be very interesting to extend their results to functions on groups, perhaps using similar tools from harmonic analysis and representation theory. For example, for which groups G besides $\mathbb{F}_2$ do expander walks fool all symmetric functions on G^n?

1.6 Organization

The rest of this paper is organized as follows. In Section 2 we prove that our wide replacement walk construction gives an expanding generating set over any finite abelian group with near-optimal degree. Due to space constraints we defer some proofs to the full version of the paper.

Appendix C contains the precise parameters of the construction. Appendices A and B contain technical preliminaries on Cayley graphs and wide replacement walks, respectively.

2 Expanding Generating Sets for Abelian Groups

Throughout this section, let G be a finite abelian group and $n \geq 1$. In this section, we will describe an efficient deterministic algorithm to construct a generating set $S \subset G^n$ such that the Cayley graph $Cay(G^n, S)$ has second eigenvalue at most ϵ. The degree is $|S| = O(\frac{n \log(|G|)^{O(1)}}{\epsilon^{2+o(1)}})$.

The inputs to our algorithm are a generating set $G' \subset G$, integer $n \geq 1$, and desired expansion $\epsilon > 0$. The algorithm proceeds as follows:

(i) Construct an ϵ_0-biased set $S_0 \subset G^n$ with support size $O(n \log(|G|)^{O(1)})$ for a constant $\epsilon_0 < 1$.

(ii) Perform a wide replacement walk to amplify the bias of S_0 to ϵ. Specifically, we identify S_0 with the vertices of an outer graph Γ, and then choose an inner graph H in a manner described later. We emphasize that while Γ is an expander graph whose vertex set is S_0, it is not required to be a Cayley graph on S_0. For the purposes of this step, the group structure of G is irrelevant.

Let $t \geq 1$ be the walk length, to be chosen later. The output ϵ-biased set $S \subset G^n$ corresponds to length-t walks on the wide replacement product of Γ and H. Given a sequence of vertices $(x_0, ..., x_t) \in V(\Gamma) \times V(H)$, we add up the components corresponding to $V(\Gamma)$, which are just elements of S_0, to obtain some element of G^n. This gives the elements of S.

Next, let us informally describe parameter choices (precise choices are in section C). Let D_2 be the degree of H. At every step in the wide replacement walk we need to specify some $i \in [D_2]$ to take a step. It follows that $S \subset G^n$ has a size of $O(n \log(|G|)^{O(1)} \cdot D_2^t)$. We must choose t large enough to shrink the bias to ϵ. The choice t (walk length) and D_2 (degree of the inner graph) will determine the overall size of the output generating set.

These choices hinge on the bias amplification bound of the wide replacement walk. We show that the s-wide replacement walk shrinks the bias by a factor of $O(s^2 \cdot \lambda(H)^{s-3})$ every s steps. However, the size of the walk distribution grows by a factor of $O(D_2^s)$ every s steps. This imperfect bias amplification is why we cannot get optimal dependence on ϵ, as that would require that the bias shrinks by exactly $O(\lambda(H)^s)$ every s steps.

Therefore we cannot choose H to be an optimal spectral expander with $\lambda(H) = \Theta(\frac{1}{\sqrt{D_2}})$. Instead, optimizing for the size of the output distribution, we set $s = \Theta(\frac{\log(1/\epsilon)^{1/3}}{\log\log(1/\epsilon)^{1/3}})$, second eigenvalue $\lambda(H) = \Theta(\frac{s \cdot \log(D_2)}{\sqrt{D_2}})$, and the walk length $t = \Theta(\frac{\log(1/\epsilon)}{\log(1/\lambda(H))} \cdot \frac{s^2}{s^2-5s+1}) = \Theta((\frac{\log(1/\epsilon)}{\log(1/\lambda(H))})^{1+o(1)})$. This is exactly the reason our output set has a dependence of $O(\frac{1}{\epsilon^{2+o(1)}})$ rather than exactly $O(\frac{1}{\epsilon^2})$, and the same is true for [41].

This section is organized as follows. In section 2.1, we describe how one can identify the elements S_0 with the vertices of an expander graph, and then perform the ordinary random walk on the graph to amplify the bias of S_0, albeit suboptimally. In section 2.2 we show how to express the bias of a wide replacement walk algebraically. In section 2.3 we prove an upper bound on this algebraic expression, therefore proving the bias amplification bound of the wide replacement walk. Finally, in section C we describe the details and exact parameters for the wide replacement walk, as well as the ϵ_0-biased subset of G^n.

2.1 The ordinary expander walk

Let G be a finite abelian group. For ease of notation, we will refer to G rather than G^n until section C, when we need to discuss parameters. Since H^n is a finite abelian group for all abelian H, there is no loss of generality.

In this section we will show how to amplify the bias of a small-bias set in G by performing a random walk on an expander. This will be a lemma in the analysis of our actual construction, which involves a *wide replacement walk.*

To state the bias amplification theorem, we need some notation.

Let $G = \mathbb{Z}_{d_1} \oplus \cdots \oplus \mathbb{Z}_{d_k}$ be the invariant factor decomposition of G. Notice that $d_i | d_j$ for any $i < j$. In particular, all d_i divide d_k. For $x \in G$ write $x = (x_1, ..., x_k)$, so that $x_i \in \mathbb{Z}_{d_i}$ for each i.

Fix a nontrivial character $\chi : G \to \mathbb{C}^*$ corresponding to a group element $a \in G$. Let $a = (a_1, ..., a_k)$. Then for a given $x \in G$, $\chi(g) = \omega_{d_1}^{a_1 \cdot x} \cdots \omega_{d_k}^{a_k \cdot x}$. Since all d_i divide d_k, we can write this as

$$\chi(g) = \omega_{d_k}^{\sum_{i=1}^k (\frac{d_k}{d_i} a_i \cdot x_i) \mod d_k}$$

Now, let $S_{init} \subset G$ have bias ϵ_0. Identify S_{init} with the vertices of some degree-regular expander graph Γ. We write $V := V(\Gamma) = S_{init}$. In order to understand the bias of a random walk on Γ with respect to χ, we have to track how often the walk enters vertices which map to $\omega_{d_k}, \omega_{d_k}^2$, and so on.

We will partition S_{init} as follows. For $y \in \mathbb{Z}_{d_k}$, let S_y be the elements of S_{init} which are mapped to $\omega_{d_k}^y$ by χ. Formally, $S_y = \{x \in S_{init} : y = (\sum_{i=1}^k \frac{d_k}{d_i} x_i \cdot a_i) \mod d_k\}$. Observe that $\{S_y : y \in \mathbb{Z}_{d_k}\}$ is a partition of S_{init}.

Next, let $t > 0$ be the walk length. We will partition all length-$(t+1)$ sequences in S_{init} according to their sum. For $y \in \mathbb{Z}_{d_k}$, let $T_y = \{b \in \mathbb{Z}_{d_k}^{t+1} : (\sum_i b_i) \mod d_k = y\}$. Again, notice that $\{T_y : y \in \mathbb{Z}_{d_k}\}$ is a partition of $\mathbb{Z}_{d_k}^{t+1}$.

Finally, fix $y \in \mathbb{Z}_{d_k}$. The set S_y corresponds to some subset of the vertices of Γ. Therefore we can identify S_y with an $|S_y|$-dimensional subspace of $\mathbb{C}^V$. Let $\Pi_y : \mathbb{C}^V \to \mathbb{C}^V$ be the projection matrix onto this subspace. Let $\Pi = \sum_{y \in \mathbb{Z}_{d_k}} \omega_{d_k}^y \Pi_y$. We write $\Pi = \Pi(\chi)$ to indicate the dependence on choice of χ.

We can now state the bias amplification theorem for ordinary expander walks.

▶ **Theorem 5** (Ordinary t-step expander walk)**.** *Let $S_{init} \subset G$ have bias ϵ_0 and let $\Gamma = (S_{init}, E)$ be a d-regular expander graph with $\lambda(\Gamma) = \lambda < 1$. Suppose $D \sim G$ is the distribution induced by beginning at a uniform vertex and taking a t-step random walk $(x^{(0)}, ..., x^{(t)})$ and then adding the results of the walk to get an element $(\sum_i x^{(i)}) \in G$.*

Let $\chi^* : G \to \mathbb{C}^*$ be the nontrivial character which maximizes the bias of D. Let $\Pi = \Pi(\chi^*)$, and $\|\cdot\|$ be the matrix operator norm. Finally, abusing notation, let Γ be the random walk matrix of Γ. Then,

$$bias(D) = bias(\chi^*) \leq \|(\Pi\Gamma)^t\Pi\|$$

Proof. Let $u = \frac{1}{\sqrt{|V(\Gamma)|}}\vec{1}$ be the normalized all-ones vector. Let $a^* \in G$ be the element corresponding to χ^*. Let $(a_1^*, ..., a_k^*) \in \mathbb{Z}_{d_1} \oplus \cdots \oplus \mathbb{Z}_{d_k}$ denote a^* written in the invariant factor decomposition.

Let $W \sim V^{t+1}$ denote the distribution of all t-step walks on Γ. Let $(x^{(0)}, ..., x^{(t)}) \sim W$ be some sequence of random walk steps. So $x^{(0)} \sim S_{init}$ (since the walk begins at a uniformly random vertex) $x^{(i+1)}$ is a uniformly random neighbor of $x^{(i)}$. If $\vec{v}^{(i)} \in \mathbb{C}^V$ is the distribution at step i, then $\vec{v}^{(i+1)} = \Gamma\vec{v}^{(i)}$.

Recall that we use subscripts to denote invariant factors, so $x = (x_1, ..., x_k) \in \bigoplus_{i=1}^{k} \mathbb{Z}_{d_i}$.

$$\begin{aligned}
Bias(D) &= Bias_D(\chi^*) \\
&= \left| \underset{(x^{(0)},...,x^{(t)})\sim W}{\mathbb{E}} \prod_{i=1}^{k} \omega_{d_i}^{x_i \cdot a_i^*} \right| \\
&= \left| \underset{(x^{(0)},...,x^{(t)})\sim W}{\mathbb{E}} \omega_{d_k}^{\sum_{i=1}^{k} \frac{d_k}{d_i} x_i \cdot a_i^*} \right| \\
&= \left| \sum_{y \in \mathbb{Z}_{d_k}} \omega_{d_k}^{y} \underset{(x^{(0)},...,x^{(t)})\sim W}{\mathbb{P}} [y = (\sum_{j=0}^{t} \sum_{i=1}^{k} \frac{d_k}{d_i} x_i^{(j)} \cdot a_i^*) \mod d_k] \right| \\
&= \left| \sum_{y \in \mathbb{Z}_{d_k}} \sum_{b \in T_y} \omega_{d_k}^{y} \underset{(x^{(0)},...,x^{(t)})\sim W}{\mathbb{P}} [\bigwedge_{j=0}^{t} (x^{(j)} \in S_{b_j})] \right| \\
&= \left| \sum_{y \in \mathbb{Z}_{d_k}} \omega_{d_k}^{y} (u^T \sum_{b \in T_y} \Pi_{b_t}\Gamma \cdots \Pi_{b_1}\Gamma\Pi_{b_0} u) \right| \\
&= \left| u^T (\sum_{b \in \mathbb{Z}_{d_k}^{t+1}} \omega_{d_k}^{\sum_j b_j} \Pi_{b_t}\Gamma \cdots \Pi_{b_1}\Gamma\Pi_{b_0}) u \right| \\
&= \left| u^T (\sum_{b_t \in \mathbb{Z}_{d_k}} \omega_{d_k}^{b_t} \Pi_{b_t})\Gamma \cdots (\sum_{b_1 \in \mathbb{Z}_{d_k}} \omega_{d_k}^{b_1} \Pi_{b_1})\Gamma(\sum_{b_0 \in \mathbb{Z}_{d_k}} \omega_{d_k}^{b_0} \Pi_{b_0}) u \right| \\
&= \left| u^T (\Pi\Gamma)^t \Pi u \right| \\
&\leq \|(\Pi\Gamma)^t\Pi\|
\end{aligned}$$

◀

We have thus obtained an algebraic expression for the bias of the walk distribution, which we will now upper-bound. We defer the proof to the full version.

▶ **Theorem 6** (Matrix norm bounds). *Let Π, Γ be as before.*

(i) $\|\Pi\| = 1$.

(ii) $\|(\Pi\Gamma)^2\| \le \epsilon_0 + 2\lambda$

It follows that $\|(\Pi\Gamma)^t\Pi\| \le (\epsilon_0 + 2\lambda)^{\lfloor t/2 \rfloor}$.

Combining the two results in this section, it follows that a t-step walk amplifies the bias to $(\epsilon_0 + 2\lambda)^{\lfloor t/2 \rfloor}$.

2.2 The wide replacement walk

In this section and the subsequent one, we will show how the wide replacement walk amplifies bias more efficiently than an ordinary expander walk. We will proceed in a similar manner to the last section, by first obtaining an algebraic expression for the bias of the random walk distribution, and then upper-bounding the algebraic expression in section 2.3.

2.2.1 Setup

Let $\Gamma = (S_{init}, E)$ be a graph whose vertices are some constant-bias set $S_{init} \subset G$ as before. Suppose Γ is D_1-regular. Let $\phi_\Gamma : [D_1] \to [D_1]$ be the local inversion function of Γ.

Let $s > 0$ be an integer, and let H be a D_2-regular expander graph on $[D_1]^s$ vertices. We will abuse notation and use Γ, H to denote the random walk matrices of Γ, H respectively.

Let $V^1 = \mathbb{C}^{S_{init}} = \mathbb{C}^{V(\Gamma)}$ and $V^2 = \mathbb{C}^{D_1^s} = \mathbb{C}^{V(H)}$. We define three operators on $V^1 \otimes V^2$ that we need to describe the bias of the wide replacement walk. Let $v^1 \otimes v^2 \in V^1 \otimes V^2$.

For $i \in [s]$ define the projection matrix $P_i : V^2 \to \mathbb{C}^{D_1}$ as follows. Notice $V^2 = \mathbb{C}^{V(H)} \cong \mathbb{C}^{D_1^s}$. Identifying $V(H)$ with $\mathbb{Z}_{D_1}^s$, let $Z_i \subset V(H)$ correspond to $\{(0, ..., 0, a_i, 0, ..., 0) \in \mathbb{Z}_{D_1}^s : a_i \in \mathbb{Z}_{D_1}\}$. So we can identify $Z_i \subset V(H)$ with a D_1-dimensional subspace of $\mathbb{C}^{V(H)}$. Then let $P_i : V^2 \to \mathbb{C}^{D_1}$ be the projection onto this subspace.

Given some $v^1 \in V^1$ and $j \in [D_1]$, the vector $v^1[j] \in V^1$ is a permutation of the coordinates of v^1 based on the mapping of each vertex to its j^{th} neighbor in Γ [3]. This corresponds to taking a step in Γ, by moving along the edge numbered j incident to the current vertex. For $w \in \mathbb{C}^{D_1}$, let $v^1[w] = \sum_{j=1}^{D_1} w_j \cdot v_1[j]$.

Finally, given the local inversion function $\phi_\Gamma : [D_1] \to [D_1]$ of Γ and $i \in [s]$, define $\psi_\Gamma^{(i)} : [D_1]^s \to [D_1]^s$ as the function which applies ϕ_Γ to the i^{th} coordinate and leaves other coordinates unchanged. Since ϕ_Γ is a permutation on $[D_1]$, $\psi_\Gamma^{(i)}$ is a permutation on $[D_1]^s$. Abusing notation, let $\psi_\Gamma^{(i)} : \mathbb{C}^{D_1^s} \to \mathbb{C}^{D_1^s}$ denote the permutation matrix which permutes coordinates according to $\psi_\Gamma^{(i)}$.

We are ready to define the three operators which describe the bias of the wide replacement walk.

$$\dot{H}(v^1 \otimes v^2) = v^1 \otimes H(v^2)$$

$$\forall \chi \in \hat{G}, y \in \mathbb{Z}_d : \dot{\Pi}_y(\chi)(v^1 \otimes v^2) = \Pi_y(\chi)(v^1) \otimes v^2$$

$$\forall \ell \in \{0, 1, ..., s-1\} : \dot{\Gamma}_\ell(v^1 \otimes v^2) = v^1[P_\ell(v^2)] \otimes \psi_\Gamma^{(\ell)}(v^2)$$

Note that each of these operators is a tensor product of operators on V^1, V^2, and hence preserves tensor products.

[3] This is well-defined as long as the graph Γ is d-regular, since its adjacency matrix is then just a sum of d permutation matrices.

Moreover, notice $\dot{H}, \dot{\Gamma}_{t \bmod s}$ are precisely the transition matrices of the H-step and Γ-step in the wide replacement walk at time t.

For a character $\chi : G \to \mathbb{C}^*$ let $\dot{\Pi}(\chi) = \sum_{y \in \mathbb{Z}_{d_k}} \omega_{d_k}^{y} \dot{\Pi}_y(\chi)$. $\dot{\Pi}$ plays the role of Π from the analysis of the ordinary expander walk.

For notational convenience,

$$\dot{L}_j(\chi) := \dot{\Pi}(\chi)\dot{\Gamma}_j\dot{H}$$

2.2.2 Algebraic Expression for the Bias

In this section we will express the bias of the wide replacement walk distribution in terms of the matrix norms of $\dot{L}_0, ..., \dot{L}_{s-1}$.

▶ **Proposition 7** (t-step s-wide replacement product walk)**.** *Let G be a finite abelian group. Let $S_{init} \subset G$ have bias ϵ_0 and let $\Gamma = (S_{init}, E)$ be a D_1-regular expander graph. Let H be a D_2 regular expander on $[D_1]^s$ vertices for some integer $s \geq 1$.*

Let $D_{walk} \sim G$ be the t-step s-wide replacement product walk distribution. It is defined by beginning at a uniform vertex and performing an t-step wide replacement wide on $V(\Gamma) \times V(H)$. Given a sequence of vertices $((a_0, b_0), ..., (a_t, b_t)) \in V(\Gamma) \times V(H)$ obtained from a walk, we output $(\sum_i a_i) \in G$. Then $D_{walk} \sim G$ is the distribution induced by taking all such t-step walks.

We claim that if $\chi^* : G \to \mathbb{C}^*$ is the nontrivial character which maximizes the bias of D_{walk}, and $\dot{\Pi} = \dot{\Pi}(\chi^*)$, then using the notation from above,

$$bias(D_{walk}) = bias(D_{walk}, \chi^*) \leq \|\dot{L}_{s-1}(\chi^*) \cdots \dot{L}_0(\chi^*)\|^{\lfloor t/s \rfloor}$$

The proof is similar to that of Theorem 5. See the full version.

It remains to be shown that this matrix norm is indeed bounded. To show that the wide replacement walk gains from $s - O(1)$ out of every s steps, we need to show that $\|\dot{L}_{s-1} \cdots \dot{L}_0\| \leq \lambda(H)^{s-O(1)}$.

2.3 Bounding the matrix norm

In the previous section we showed that the bound the bias of the wide-replacement walk distribution, it suffices to bound the operator norm of the following matrix, defined with respect to the worst-case character χ^* of the walk distribution:

$$\dot{L}_{s-1} \cdots \dot{L}_0$$

This is almost exactly the same matrix as the one analyzed in [41]. The difference is that the operator $\dot{\Pi}$, instead of tracking how often the walk enters the sets in a bipartition of S_{init}, now tracks how often the walk enters the sets in a d_k-way partition of S_{init}. Here $d_k = \Omega(\log(|G|))$ is the largest invariant factor of G.

As a consequence, the diagonal entries of $\dot{\Pi}$ now come from the ${d_k}^{th}$ roots of unity, rather than $\{\pm 1\}$. The analysis of the matrix bound from [41] mostly carries through, although working over $\mathbb{C}^{V_1} \otimes \mathbb{C}^{V^2}$ rather than the reals will require some care.

As in [41], our argument will proceed by considering arbitrary vectors v, w and analyazing $\langle v, \dot{L}_{s_1} \cdots \dot{L}_0 w\rangle$. We will repeatedly decompose the vectors into their parallel and perpendicular components. Let $V^{\parallel} = V^1 \otimes \vec{1}$ denote vectors whose H-component is a scalar multiple of $\vec{1}$ ("parallel vectors"), and $V^{\perp} = (V^{\parallel})^{\perp}$ ("perpendicular vectors").

Because of the spectral expansion of H, every time a vector is in $V^{\perp}$ we can show it shrinks by a factor of $\lambda(H)$. The hard case is when vectors are in $V^{\|}$. Here, we will prove a technical lemma which is a straightforward generalization of the core lemma in [41]. The lemma shows if the walk distribution is in $V^{\|}$, then any *sequence* of s steps imitates a random walk of s steps on the outer graph Γ. This allows us to argue that the bias is amplified as though taking the ordinary random walk on Γ. If the bias so far is α, then this scales the bias by $\alpha \mapsto (\alpha + 2\lambda(\Gamma))^{s/2}$ after s steps.

This turns out to be enough. Let $\epsilon_0 = bias(S_{init})$ be the bias of the initial set $S_{init} \subset G$. Since ϵ_0 is a constant, we can select graphs Γ, H such that $\epsilon_0 + 2\lambda(\Gamma) \leq \lambda(H)^2$. Therefore, while we do not gain a factor of $(\lambda(\Gamma))^s$ every s steps, we will gain according to a factor of $(\lambda(H))^{s-O(1)}$.

Therefore, whether in the $V^{\perp}$ or $V^{\|}$ case, we shrink the bias by a factor of $\lambda(H)^{s-O(1)}$ for every s steps.

We begin by proving the technical lemma about parallel vectors. We will frequently use the following fact.

▶ **Proposition 8** (Operator-Averaging, [41] Claim 14). *Let Ω be a finite set and P, Q probability distributions on Ω. Let $\|P - Q\|_1$ denote the difference of the distributions in the 1-norm. Further, let $\{T_x\}_{x\in\Omega}$ be a family of linear operators on $\mathbb{C}^n$ indexed by Ω, such that for all $x \in \Omega$, $\|T_x\| \leq 1$. Let $A = \mathbb{E}_{x\sim P}[T_x]$ and $B = \mathbb{E}_{x\sim Q}[T_x]$. We claim that for all $v, w \in \mathbb{C}^n$ that*

$$|\langle Av, w\rangle - \langle Bv, w\rangle| \leq \|P - Q\|_1 \|v\| \|w\|$$

Next, we need to formalize the notion of the wide replacement walk "imitating" the ordinary random walk on the outer graph, which we do via the notion of a pseudorandom inner graph.

▶ **Definition 9** (Pseudorandom inner graph). *Let Γ be a D_1-regular graph with local inversion function $\phi_\Gamma : [D_1] \to [D_1]$. Let H be a D_2-regular graph on D_1^s vertices. Let $\zeta \geq 0$. We say H is ζ-pseudorandom with respect to Γ if for all s-step sequences in the s-wide replacement walk, the corresponding V^1-instructions are ζ-close to $Unif([D_1]^s)$ in ℓ_1-norm.*

Formally, let the adjacency matrix of H be $H = \frac{1}{D_2}\sum_{i=1}^{D_2} \Xi_i$, where each Ξ_i is a permutation matrix [4]. *Let $\xi_i : V(H) \to V(H)$ be the permutation map corresponding to Ξ_i. For $0 \leq k < s$, let $\psi_k : [D_1]^s \to [D_1]^s$ be $\psi_k(a_0, ..., a_{s-1}) = (a_0, ..., a_{k-1}, \phi_\Gamma(a_k), a_{k+1}, ..., a_{s-1})$.*

Fix $(j_0, ..., j_{s-1}) \in [D_2]^s$. For some $(u^1, u^2) \in V(\Gamma) \times V(H)$ let $\sigma_{j_0}(u^2) = \gamma_{j_0}(u^2)$. For $\ell > 0$, let

$$\sigma_{j_\ell,...,j_0}(u^2) = \gamma_{j_\ell}(\psi_{\ell-1}(\sigma_{j_{\ell-1},...,j_0}(u^2)))$$

We say $(j_0, ..., j_{s-1}) \in [D_2]^s$ is ζ-pseudorandom with respect to Γ if

$$\|(\pi_0(\sigma_{j_0}(Unif([D_1]))), ..., \pi_{s-1}(\sigma_{j_{s-1},...,j_0}(Unif([D_1])))) - Unif([D_1]^s)\|_1 \leq \zeta$$

We say the inner graph H is ζ-pseudorandom with respect to the outer graph Γ if for all $(j_0, ..., j_{s-1}) \in [D_2]^s$, $(j_0, ..., j_{s-1})$ is ζ-pseudorandom with respect to Γ.

[4] By the Birkhoff-von Neumann Theorem, the adjacency matrix of a d-regular graph is a sum of d permutation matrices.

If we unravel the definition, this is simply requiring that H is compatible with the edge labeling of Γ in precisely the way that we want. Pseudorandomness is a strong condition on H which, by definition, guarantees the wide-replacement walk imitates the ordinary walk on Γ in a suitable sense.

With this definition we can return to proving the lemma. We will begin by proving the pseudorandomness claim for the case where $D_2 = 1$; the general case where $D_2 \neq 1$ follows from another application of operator averaging, viewing the matrix H as an average of D_2 permutation matrices. We defer the proofs to the full version.

▶ **Proposition 10** (Action on parallel vectors). *Let $\ell \leq s$. Suppose that the sequence $(j_0, ..., j_{\ell-1}) \in [D_2]^s$ is ζ-pseudoranom with respect to the local inversion function $\phi : [D_1] \to [D_1]$. Let $\tilde{\Xi}_{j_0}, ..., \tilde{\Xi}_{j_{\ell-1}}$ denote the operators on $V^1 \otimes V^2$ corresponding to the permutations $\xi_{j_0}, ..., \xi_{j_{\ell-1}}$ on $V(H)$. Let $1_{V(H)}$ denote the normalized all-ones vector of length $|V(H)|$.*

For any $\tau = \tau^1 \otimes 1_{V(H)}$ and $\upsilon = \upsilon^1 \otimes 1_{V(H)}$,

$$\left|\langle \dot{\Pi}\dot{\Gamma}_{\ell-1}\tilde{\Xi}_{j_{\ell-1}} \cdots \dot{\Pi}\dot{\Gamma}_0\tilde{\Xi}_{j_0}\tau, \upsilon\rangle - \langle(\pi\Gamma)^{\ell}\tau^1, \upsilon^1\rangle\right| \leq \zeta\|\tau\|\|\upsilon\|$$

▶ **Corollary 11** (Generalized action on parallel vectors ([41] Theorem 27)). *Suppose that H is ζ-pseudorandom with respect to the local inversion function ϕ_Γ of Γ. For every $i_1, i_2 \in \{0, 1, ..., s-1\}$, and every $\tau, \upsilon \in V^{\parallel}$,*

$$\left|\langle \dot{L}_{i_2} \cdots \dot{L}_{i_1}\tau, \upsilon\rangle - \langle(\Pi\Gamma)^{i_2-i_1+1}\tau^1, \upsilon^1\rangle\right| \leq \zeta\|\tau\|\|\upsilon\|$$

Now we are ready to prove bound the matrix norm of $\dot{L}_{s_1} \cdots \dot{L}_0$, which expresses the bias of the wide replacement walk. Our argument will proceed by considering the quadratic form $\langle v, \dot{L}_{s_1} \cdots \dot{L}_0 w\rangle$ for arbitrary v, w and then repeatedly decomposing v, w into their $V^{\parallel}$ and $V^{\perp}$ components. Because of the spectral expansion of H, every time a vector is in $V^{\perp}$ we can show it shrinks by a factor of $\lambda_2 = \lambda(H)$.

The hard case is when vectors are in $V^{\parallel}$. Here, we will use Corollary 11 to argue that any *sequence* of s steps imitates a random walk on the outer graph Γ. This allows us to argue that the bias is amplified as though taking the ordinary random walk on Γ. This scales the bias by $(\epsilon_0 + 2\lambda_1)^{s/2}$ at every s steps.

This is enough, as we can assume that $\epsilon_0 + 2\lambda_1 \leq \lambda_2^2$. Therefore, while we do not gain a factor of $(\lambda_1)^s$ every s steps, we will gain according to a factor of $(\lambda_2)^s$. Since $\lambda_2 < 1$, the difference between gaining according to λ_2 or λ_1 does not matter asymptotically.

▶ **Theorem 12** (Bounding algebraic expression for bias). *Suppose that:*

(i) *H is ζ-pseudorandom with respect to ϕ_Γ*

(ii) $\epsilon_0 + 2\lambda(\Gamma) \leq \lambda(H)^2$

Then we obtain the following bound for the bias of the walk after s steps.

$$\|\dot{L}_{s-1} \cdots \dot{L}_0\| \leq \lambda(H)^s + s\lambda(H)^{s-1} + s^2(\lambda(H)^{s-2} + \zeta)$$

We defer the proof to the full version.

References

1 Miklós Ajtai, Henryk Iwaniec, János Komlós, János Pintz, and Endre Szemerédi. Construction of a thin set with small Fourier coefficients. *Bull. London Math. Soc.*, 22(6):583–590, 1990. doi:10.1112/blms/22.6.583.

2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

3 Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

4 Noga Alon. Explicit expanders of every degree and size. *Combinatorica*, pages 1–17, 2021.

5 Noga Alon and Gil Cohen. On rigid matrices and u-polynomials. In *2013 IEEE Conference on Computational Complexity*, pages 197–206. IEEE, 2013.

6 Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

7 Noga Alon and Yishay Mansour. ϵ-discrepancy sets and their application for interpolation of sparse polynomials. *Inform. Process. Lett.*, 54(6):337–342, 1995. doi:10.1016/0020-0190(95)00032-8.

8 Noga Alon, Rina Panigrahy, and Sergey Yekhanin. Deterministic approximation algorithms for the nearest codeword problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 339–351. Springer, 2009.

9 Noga Alon and Yuval Roichman. Random Cayley graphs and expanders. *Random Structures Algorithms*, 5(2):271–284, 1994. doi:10.1002/rsa.3240050203.

10 Andris Ambainis and Joseph Emerson. Quantum t-designs: t-wise independence in the quantum world. In *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*, pages 129–140. IEEE, 2007.

11 V. Arvind, Partha Mukhopadhyay, Prajakta Nimbhorkar, and Yadu Vasudev. Expanding generating sets for solvable permutation groups. *SIAM J. Discrete Math.*, 32(3):1721–1740, 2018. doi:10.1137/17M1148979.

12 V. Arvind and Srikanth Srinivasan. The remote point problem, small bias spaces, and expanding generator sets. In *STACS 2010: 27th International Symposium on Theoretical Aspects of Computer Science*, volume 5 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 59–70. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2010.

13 Vikraman Arvind, Partha Mukhopadhyay, and Prajakta Nimbhorkar. Erdős-rényi sequences and deterministic construction of expanding cayley graphs. In *Latin American Symposium on Theoretical Informatics*, pages 37–48. Springer, 2012.

14 Yossi Azar, Rajeev Motwani, and Joseph Naor. Approximating probability distributions using small sample spaces. *Combinatorica*, 18(2):151–171, 1998. doi:10.1007/PL00009813.

15 Eric Bach and Jonathan Sorenson. Explicit bounds for primes in residue classes. *Mathematics of Computation*, 65(216):1717–1735, 1996.

16 Avraham Ben-Aroya and Amnon Ta-Shma. A combinatorial construction of almost-ramanujan graphs using the zig-zag product. *SIAM Journal on Computing*, 40(2):267–290, 2011.

17 Avraham Ben-Aroya and Amnon Ta-Shma. Constructing small-bias sets from algebraic-geometric codes. *Theory Comput.*, 9:253–272, 2013. doi:10.4086/toc.2013.v009a005.

18 Eli Ben-Sasson, Madhu Sudan, Salil Vadhan, and Avi Wigderson. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 612–621. ACM, New York, 2003. doi:10.1145/780542.780631.

19 Amey Bhangale, Prahladh Harsha, Orr Paradise, and Avishay Tal. Rigid matrices from rectangular pcps or: Hard claims have complex proofs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 858–869. IEEE, 2020.

20 Emmanuel Breuillard and Alexander Lubotzky. Expansion in simple groups. *arXiv preprint*, 2018. arXiv:1807.03879.

21 Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Near-optimal algorithms for maximum constraint satisfaction problems. *ACM Trans. Algorithms*, 5(3):Art. 32, 14, 2009. doi:10.1145/1541885.1541893.

22 Eshan Chattopadhyay, Pooya Hatami, Kaave Hosseini, and Shachar Lovett. Pseudorandom generators from polarizing random walks. *Theory Comput.*, 15:Paper No. 10, 26, 2019. doi:10.4086/toc.2019.v015a010.

23 Sixia Chen, Cristopher Moore, and Alexander Russell. Small-bias sets for nonabelian groups: derandomizations of the Alon-Roichman theorem. In *Approximation, randomization, and combinatorial optimization*, volume 8096 of *Lecture Notes in Comput. Sci.*, pages 436–451. Springer, Heidelberg, 2013. doi:10.1007/978-3-642-40328-6_31.

24 Tobias Christiani and Rasmus Pagh. Generating k-independent variables in constant time. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 196–205. IEEE, 2014.

25 Gil Cohen, Noam Peri, and Amnon Ta-Shma. Expander random walks: A fourier-analytic approach. In *Electron. Colloquium Comput. Complex*, volume 27, page 6, 2020.

26 Guy Even, Oded Goldreich, Michael Luby, Noam Nisan, and Boban Veličković. Efficient approximation of product distributions. *Random Structures Algorithms*, 13(1):1–16, 1998. doi:10.1002/(SICI)1098-2418(199808)13:1<1::AID-RSA1>3.0.CO;2-W.

27 Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP congress*, volume 839, page 842, 1977.

28 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

29 Akhil Jalan and Dana Moshkovitz. Near-optimal cayley expanders for abelian groups. *arXiv preprint*, 2021. arXiv:2105.01149.

30 Fernando Granha Jeronimo, Dylan Quintana, Shashank Srivastava, and Madhur Tulsiani. Unique decoding of explicit epsilon-balanced codes near the gilbert-varshamov bound. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 434–445. IEEE, 2020.

31 Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18(5):652–656, 1972.

32 Nicholas M. Katz. An estimate for character sums. *J. Amer. Math. Soc.*, 2(2):197–200, 1989. doi:10.2307/1990974.

33 Ivan Korec and Jiří Wiedermann. Deterministic verification of integer matrix multiplication in quadratic time. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 375–382. Springer, 2014.

34 Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

35 Joseph Naor and Moni Naor. Small-bias probability spaces: efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993. doi:10.1137/0222053.

36 A. Razborov, E. Szemerédi, and A. Wigderson. Constructing small sets that are uniform in arithmetic progressions. *Combin. Probab. Comput.*, 2(4):513–518, 1993. doi:10.1017/S0963548300000870.

37 Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 3–13. IEEE, 2000.

38 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

39 Amir Shpilka and Avi Wigderson. Derandomizing homomorphism testing in general groups. *SIAM Journal on Computing*, 36(4):1215–1230, 2006.

40 Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *STOC'17—Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 238–251. ACM, New York, 2017. doi:10.1145/3055399.3055408.

41 Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *TR 17-041*. Electronic Colloqium on Computational Complexity, 2017.

42 Salil Vadhan. *Pseudorandomness*, volume 7. Now Delft, 2012.

43 Leslie G Valiant. Graph-theoretic arguments in low-level complexity. In *International Symposium on Mathematical Foundations of Computer Science*, pages 162–176. Springer, 1977.

44 Avi Wigderson and David Xiao. Derandomizing the ahlswede-winter matrix-valued chernoff bound using pessimistic estimators, and applications. *Theory of Computing*, 4(1):53–76, 2008.

45 Triantafyllos Xylouris. *Über die Nullstellen der Dirichletschen L-Funktionen und die kleinste Primzahl in einer arithmetischen Progression*, volume 404 of *Bonner Mathematische Schriften [Bonn Mathematical Publications]*. Universität Bonn, Mathematisches Institut, Bonn, 2011. Dissertation for the degree of Doctor of Mathematics and Natural Sciences at the University of Bonn, Bonn, 2011.

A Cayley Graphs and Expanders

We begin with some preliminaries on graphs and group theory.

▶ **Definition 13** (Spectral expander graph). *Let $G = ([n], E, w)$ be a weighted, d-regular undirected graph. By d-regular we mean that for all $u \in V$, $\sum_{v \in V} w(\{u, v\}) = d$.*

Let $A \in \mathbb{C}^{n \times n}$ be the (weighted) adjacency operator of G, and let $M = \frac{1}{d}A$ be the normalized adjacency operator, also known as the random walk matrix. Let the eigenvalues of M be denoted $\lambda_n \leq \ldots \leq \lambda_2 \leq \lambda_1 = 1$, counting multiplicity. Then G is a one-sided spectral expander if $\lambda_2 < 1 - \Omega(1)$, and G is a two-sided spectral expander if $\max\{|\lambda_n|, |\lambda_2|\} < 1 - \Omega(1)$.

Let $\lambda(G) := \max\{|\lambda_n|, |\lambda_2|\}$. The two-sided spectral gap of G is $1 - \lambda(G)$.

Next, we define Cayley graphs.

▶ **Definition 14** (Symmetric generating set). *Let G be a group and $S \subset G$. We say that S is symmetric if for all $s \in S$, $s^{-1} \in S$. Further, S is a generating set if for all $g \in G$ there exist $s_1, \ldots, s_k \in S$ (possibly repeated) such that $s_k \cdots s_1 = g$.*

We write $\langle S \rangle = G$.

▶ **Definition 15** (Cayley Graph). *Let G be a group and $S \subset G$ be a symmetric generatring set, and $w : S \to \mathbb{R}_{\geq 0}$ a weight function. The Cayley graph $Cay(G, S, w)$ is the graph with vertex set G and edge set $\{\{g, g \cdot s\} : g \in G, s \in S\}$. The weight of an edge $\{g, g \cdot s\}$ is $w(s)$.*

We will require the total weight of S to be normalized to $|S|$ by convention. Notice that since S is symmetric, we can consider the graph $\text{Cay}(G, S)$ to be an undirected and weighted $|S|$-regular multigraph.

The eigenvectors of abelian Cayley graphs are described by their group characters.

▶ **Definition 16** (Characters of abelian group). *Let $\mathbb{C}^*$ be the multiplicative group of nonzero complex numbers. For any finite abelian group G, the characters of G, denoted $\hat{G}$, are the set of all homomorphisms $\chi : G \to \mathbb{C}^*$.*

▶ **Proposition 17.** *Let G be a finite abelian group and $S \subset G$ a symmetric generating set. Then the eigenvalues of $Cay(G, S)$ are given by*

$$\{|\mathop{\mathbb{E}}_{x \sim S}[\chi(x)]| : \chi \in \hat{G}\}$$

Notice that any group has a *trivial character* $\chi : G \to \mathbb{C}^*$ such that $\chi(g) = 1$ for all g. The eigenvalue corresponding to the trivial character is always 1. Therefore, for a Cayley graph to be an expander we need bounds on all of its nontrivial characters.

▶ **Definition 18** (Small-bias distributions for abelian groups). *Let G be a finite abelian group and $D \sim G$ a random variable. For any character χ of G, the bias of D with respect to χ is*

$$Bias_\chi(D) := |\mathop{\mathbb{E}}_{x \sim D}[\chi(x)]|$$

Let χ_0 denote the trivial character. The bias of D is its maximum bias with respect to nontrivial characters.

$$Bias(D) := \max_{\chi \neq \chi_0} Bias_\chi(D)$$

If $S \subset G$, then $bias(S)$ is the bias of the uniform distribution on S. If S is a symmetric generating set, $\lambda(Cay(G, S)) = Bias(S)$.

Notice that if S is non-negatively weighted, we can normalize weights to sum to 1 and obtain a (not necessarily uniform) distribution on S. Then the bias of S is just the bias of this distribution.

Finally, we will need a few more facts about characters of abelian groups.

▶ **Proposition 19** (Characters of cyclic groups). *Let $\mathbb{Z}_d$ be the cyclic group on $d \geq 2$ elements. Let $\omega_d := exp(\frac{2\pi i}{d})$. The characters of $\mathbb{Z}_d$ are the maps $\chi_j(x) = \omega_d^{j \cdot x}$ for $j = 0, 1, ..., d-1$.*

▶ **Definition 20** (Direct sum of groups). *Let A, B be abelian groups. The direct sum $A \oplus B$ is the abelian group whose elements belong to the Cartesian product $A \times B$. For $(a_1, b_1), (a_2, b_2) \in A \times B$, the group operation is $(a_1, b_1) + (a_2, b_2) = (a_1 + a_2, b_1 + b_2)$.*

Notice that the direct sum is associative.

▶ **Proposition 21** (Fundamental theorem of finite abelian groups). *Let G be a finite abelian group. Then G is isomorphic to a direct sum of cyclic groups. That is, there exist $d_1, ..., d_k \geq 2$ such that*

$$G \cong \mathbb{Z}_{d_1} \oplus \cdots \oplus \mathbb{Z}_{d_k}$$

Moreover, $d_i | d_j$ for all $i < j$.

We refer to $\mathbb{Z}_{d_1} \oplus \cdots \oplus \mathbb{Z}_{d_k}$ as the invariant factor decomposition of G. The integers $d_1, ..., d_k$ are the invariant factors.

From the above propositions one can show that the characters of a finite abelian group are products of maps of the form $x \mapsto \omega_{d_i}^{j \cdot x}$. This structure is crucial to our overall argument.

B Wide Replacement Walks

In this section we define what it means to take a wide replacement walk.

Let G be a D_1-regular graph on N_1 vertices and H be a D_2-regular graph on D_1 vertices. The *replacement product* $G Ⓡ H$ is a $(D_2 + 1)$-regular graph on $N_1 \cdot D_1$ vertices. Each vertex of G (the "outer graph") is replaced by a copy of H (the "inner graph"). We call these copies *clouds*.

The intra-cloud edges in each cloud of $G Ⓡ H$ are just the edges from H. However, $G Ⓡ H$ also has *inter*-cloud edges which arise by identifying the D_1 vertices of H with the D_1 incident edges of a vertex $v \in V(G)$. This identification requires that we number the edges of every vertex in G. We formalize this with the concept of a rotation map.

▶ **Definition 22** (Rotation map). *Let G be a D-reguluar graph such that the edges incident to every $v \in V(G)$ are numbered $1, ..., D$. Formally there is a function $N : V \times [D] \to V$ such that $N(v, i) = w$ iff w is the i^{th} neighbor of v.*

Then a rotation map is a function $Rot : V \times [D] \to V \times [D]$ such that for all $v, w \in V$ and $i, j \in [D]$, $Rot(v, i) = (w, j)$ iff the i^{th} neighbor of v is w and the j^{th} neighbor of w is v.

For technical reasons, we need a special kind of rotation map called a local inversion function. This is a rotation map where if (v, i) maps to (w, j) then j only depends on i.

▶ **Definition 23** (Local inversion function). *Let G be a D-regular graph with a rotation map $Rot : V \times [D] \to V \times [D]$. A local inversion function $\phi_G : [D] \to [D]$ is a permutation on $[D]$ such that for all $v \in V, i \in [D]$,*

$$Rot(v, i) = (N(v, i), \phi_G(i))$$

We are ready to define the wide replacement product walk. Instead of the usual inner graph H we use a "wide" inner graph on D_1^s vertices for some integer $s \geq 1$. The vertices of H correspond to s-tuples that define s local inversion functions. The walk cycles through them.

To take a step in the usual replacement product walk, we start at some vertex $v \in G \textcircled{r} H$ then compose two steps: an intra-cloud step which changes the H-component, and an inter-cloud step which changes the G-component. Every vertex in $G \textcircled{r} H$ is incident to a unique inter-cloud edge; therefore, there is only one choice of neighboring cloud, and so the position after the intra-cloud step determines the entire step.

The s-wide replacement walk modifies the inter-cloud step so that there are s choices during inter-cloud step. If G is D_1-regular, then a vertex of H corresponds to some vector $(a_0, ..., a_{s-1}) \in [D_1]^s$. The wide replacement walk maintains a clock which tracks how many steps have been taken. At time step t, the clock is set to $\ell = t \mod s$, and the inter-cloud step moves to a neighboring cloud according to the value of $a_\ell \in [D_1]$.

After deciding which neighboring cloud to move to, the choice of which vertex in the cloud to land in is also determined by a_ℓ. The walk updates the H-component by feeding the ℓ^{th} coordinate to the local inversion function $\phi_G : [D_1] \to [D_1]$ of G, and leaving all other coordinates unchanged. So $(a_0, ..., a_{s-1}) \in [D_1]^s$ is mapped to $(a_0, ..., a_{\ell-1}, \phi_G(a_\ell), a_{\ell+1}, ..., a_{s-1})$. This completes the inter-cloud step.

The utility of the wide replacement walk is that the H-component of a vertex now stores $O(s \log(D_1))$ bits of information, rather than just $O(\log(D_1))$ bits. As we discussed in the introduction, the barrier to bias amplification is when the walk distribution is uniform within clouds.

Now, the values of the H-component are precisely the instructions for the inter-cloud steps of the walk; therefore, the fact that the H-component is uniform is no longer bad news, since it means that the inter-cloud steps of the replacement walk imitate the truly random walk on the outer graph for the next s steps.

▶ **Definition 24.** *Let G be a D_1-regular graph with local inversion function $\phi_G : [D_1] \to [D_1]$. Let H be a D_2-regular graph on D_1^s vertices, for integer $s \geq 1$. A random step in the wide replacement product is determined as follows.*

Let $(v^{(1)}, v^{(2)}) \in V(G) \times V(H)$ be the current state of the walk at time $t \in \mathbb{N}$. Sample random $i \in [D_2]$. Then the time-t step according to i, denoted $Step_{i,t}(v^{(1)}, v^{(2)})$ is given by the composition of two steps:

(i) *Intra-cloud step: Leave the G-component $v^{(1)}$ unchaged. Move the $v^{(2)}$ component to its i^{th} neighbor in H. Formally, set*

$$w^{(1)} = v^{(1)}$$
$$w^{(2)} = v^{(2)}[i].$$

(ii) *Inter-cloud step: Identifying $V(H)$ with $[D_1]^s$, let $\pi_j : [D_1]^s \to [D_1]$ be projection onto the j^{th} coordinate. Write $w^{(2)} \in V(H)$ as $w^{(2)} = (\pi_0(w^{(2)}), ..., \pi_{s-1}(w^{(2)})) \in [D_1]^s$. Let $\ell = t \mod s$. Move to the neighbor of $w^{(1)}$ in G that is numbered by $\pi_\ell(w^{(2)}) \in D_1$. Then, update the ℓ^{th} coordinate of H-component $w^{(2)}$ by the local inversion function $\phi_G : [D_1] \to [D_1]$ and leave other coordinates unchaged. Formally, let $\psi_\ell : [D_1]^s \to [D_1]^s$ be*

$$\psi_\ell(a_0, ..., a_{s-1}) = (a_0, ..., a_{\ell-1}, \phi_G(a_\ell), a_{\ell+1}, ..., a_{s-1})$$

Set

$$Step_{i,t}(v^{(1)}, v^{(2)}) = (w^{(1)}[\pi_\ell(w^{(2)})], \psi_\ell(w^{(2)})).$$

A few remarks are in order. First, notice that the number of random bits needed to specify a random step is only $O(\log(D_2))$, despite the fact that we are moving on a graph with $V(G) \times V(H)$ vertices. This will be crucial in the analysis of the tradeoff between bias amplification and size increase of the small-bias set.

Second, once a value of t is fixed, so the clock is set to $\ell = t \mod s$, the wide replacement walk can be regarded as taking a usual step in the usual replacement walk. The intra-cloud step is unchaged, and the inter-cloud step depends only on the ℓ^{th} coordinate of the H-component.

Since we have specified what it means to take a random step, this is sufficient to describe the walk. We simply initialize at a uniform vertex of $V(G) \times V(H)$ and then take some number of steps, to be chosen later.

C Parameters of the Construction

In this section we describe how to optimize parameters such that the wide replacement walk construction achieves our desired support size. Our construction and hence the parameters we choose are almost identical to those discussed in Section 5 of [41].

The algorithm is given integer $n \geq 1$, desired second eigenvalue $\epsilon > 0$, and an arbitrary generating set for a group G.

It first generates an ϵ_0-biased set $S_{init} \subset G^n$ of size $O(\frac{n\log(|G|)^{O(1)}}{poly(\epsilon_0)})$ for a constant ϵ_0. For concreteness we set $\epsilon_0 = 0.1$.

▶ **Proposition 25.** *There exists a deterministic, polynomial time algorithm which, given a generating set for an abelian group G and integer $n \geq 1$, outputs a generating set $S_{init} \subset G^n$ of size $O(n(\log(|G|))^{O(1)})$ such that the Cayley graph has second eigenvalue at most* 0.1.

Proof. First, by Theorem 4 of [23], we can construct a generating set $S \subset G$ with second eigenvalue $(1 - \frac{C}{\log\log(|G|)} + \beta)$ for a parameter β and universal constant C. Its size will be $|S| = O(\frac{n\log(|G|)}{\beta^{O(1)}}) = O(n\log(|G|)^2)$. Setting $\beta = \frac{C}{2\log\log(|G|)}$, we obtain second eigenvalue $(1 - \frac{C}{2\log\log(|G|)})$.

Next, we can amplify the bias of S to 0.1 by taking a t-step ordinary expander walk. By the results of section 3.1, if we take a walk on a D-regular expander graph with second eigenvalue λ and $D = O(1)$, then the t-step walk will amplify the bias to $((1 - \frac{C}{2\log\log(|G|)}) + 2\lambda)^{\lfloor t/2 \rfloor}$. For this quantity to be at most 0.1, it suffices to set $t > \frac{\log\log(|G|)}{C}(1+2\lambda) = \Theta(\log\log(|G|))$.

Therefore, after t steps we obtain a generating set $S_0 \subset G^n$ with bias 0.1, whose size is $|S_0| \cdot D^t = O(\frac{n\log(|G|)^2}{(0.1)^{O(1)}} \cdot 2^{\Theta(\log\log(|G|))}) = O(n(\log(|G|))^{O(1)})$. ◀

Next, the algorithm performs a wide replacement walk. We must specify the inner and outer graphs as well as the number of steps. Our parameters are almost identical to [41].

Let $\alpha = \Theta((\frac{\log\log(\frac{1}{\epsilon})}{\log(\frac{1}{\epsilon})})^{1/3})$. We will show that the wide replacement walk amplifies bias to ϵ and produces a generating set of size $O(\frac{n\log(|G|)^{O(1)}}{\epsilon^{2+O(\alpha)}}) = O(\frac{n\log(|G|)^{O(1)}}{\epsilon^{2+o(1)}})$.

Let the "width" $s = \frac{1}{\alpha}$.

Inner Graph: Let D_2 be the least power of two such that $D_2 \geq s^{4s}$. Let $b_2 = 4s\sqrt{2}\log(D_2)$. Let $D_1 = D_2^4$. Let $m = \log(D_1)$.

Let $H = Cay(\mathbb{Z}_2^{ms}, A)$ for a generating set of size $|A| = D_2$ (found, e.g via [41]) such that the second eigenvalue is $\lambda(H) = \frac{b_2}{\sqrt{D_2}}$.

Outer graph: Let $D_1 = D_2^4$. Find a D_1-regular expander graph Γ with $\lambda(\Gamma) = \Theta(\frac{1}{\sqrt{D_1}})$ (using, e.g. [4]). Identify its vertices with the ϵ_0-biased set S_{init}.

Walk length: Finally, set t to be the least integer such that $\lambda(H)^{(1-4\alpha)(1-\alpha)t} \leq \epsilon$ and $t \geq \frac{s}{\alpha}$.

▶ **Proposition 26.** *The t-step wide replacement walk distribution is ϵ-biased.*

Proof. The bias after t steps is given by $(\lambda(H)^s + s\lambda(H)^{s-1} + s^2\lambda(H)^{s-2})^{\lfloor t/s \rfloor}$. Therefore,

$$\begin{aligned}
(\lambda(H)^s + s\lambda(H)^{s-1} + s^2\lambda(H)^{s-2})^{\lfloor t/s \rfloor} &\leq (2s^2\lambda(H)^{s-3})^{\lfloor t/s \rfloor} \\
&\leq (2s^2\lambda(H)^{s-3})^{t/s-1} \\
&\leq (\lambda(H)^{s-4})^{t/s-1} \\
&= \lambda(H)^{\frac{s-4}{s}(t-s)} \\
&= \lambda(H)^{(1-\frac{4}{s})(1-\frac{s}{t})t} \\
&\leq \lambda(H)^{(1-4\alpha)(1-\alpha)t} \\
&\leq \epsilon
\end{aligned}$$

The last step follows by assumption on t. ◀

▶ **Proposition 27.** *The support size of the wide replacement walk distribution is $O(|S_{init}| \cdot \frac{1}{\epsilon^{2+O(\alpha)}})$, where S_{init} is the initial constant-bias set.*

Proof. Recall that we identify our initial 0.1-biased distribution with the vertices of the outer graph Γ. Therefore $N_1 = |V(\Gamma)| = O(\frac{n\log(|G|)^{O(1)}}{\epsilon_0^c})$ for constant $\epsilon_0, c > 0$. Since ϵ_0 is constant we can assume $D_2 \geq \epsilon_0^{-1}$. The walk begins at a uniform vertex of the replacement product, so the initial support size is N_1N_2. After t steps it increases by a factor of D_2^t. Therefore

$$\begin{aligned}
N_1N_2D_2^t &= O(\frac{n\log(|G|)^{O(1)}}{\epsilon_0^c}N_2D_2^t) \\
&= O(\frac{n\log(|G|)^{O(1)}}{\epsilon_0^c}D_2^{4s}D_2^t) \\
&= O(n\log(|G|)^{O(1)} \cdot D_2^{4s+t+c}) \\
&\leq O(n\log(|G|)^{O(1)} \cdot D_2^{4\alpha t+t+c}) \\
&\leq O(n\log(|G|)^{O(1)} \cdot D_2^{t(1+5\alpha)})
\end{aligned}$$

Next, notice $b_2 = 4\sqrt{2}s\log(D_2) = 4\sqrt{2}\cdot 4s^2\log(s) \leq s^4$ for sufficiently large s (equivalently, small enough ϵ). Therefore, $D_2 \geq (s^4)^s \geq b_2^s = b_2^{1/\alpha}$. Therefore $D_2^{1/2-\alpha} \leq \lambda(H)^{-1} = \frac{\sqrt{D_2}}{b_2}$.

It follows that for small enough α (equivalently, small enough ϵ), that

$$D_2^t \leq (\lambda(H)^{-1})^{\frac{t}{1/2-\alpha}} = (\lambda(H)^{-1})^{\frac{2t}{1-2\alpha}} = (\epsilon^{-1})^{\frac{1}{(1-4\alpha)(1-\alpha)t}\frac{2t}{1-2\alpha}} \leq (\epsilon^{-1})^{2(1+8\alpha)}$$

Finally, $D_2^{t(1+5\alpha)} \leq (\epsilon^{-1})^{2(1+8\alpha)(1+5\alpha)} \leq (\epsilon^{-1})^{2(1+14\alpha)}$.

Therefore, the overall size of the generating set is $O(\frac{n\log(|G|)^{O(1)}}{\epsilon^{2+O(\alpha)}})$. In particular, since $\alpha \to 0$ as $\epsilon \to 0$, the size is $O(\frac{n\log(|G|)^{O(1)}}{\epsilon^{2+o(1)}})$. ◀

Matchings, Critical Nodes, and Popular Solutions

Telikepalli Kavitha
Tata Institute of Fundamental Research, Mumbai, India

Abstract

We consider a matching problem in a marriage instance G. Every node has a strict preference order ranking its neighbors. There is a set C of prioritized or critical nodes and we are interested in only those matchings that match as many critical nodes as possible. Such matchings are useful in several applications and we call them *critical matchings*. A stable matching need not be critical. We consider a well-studied relaxation of stability called *popularity*. Our goal is to find a popular critical matching, i.e., a weak Condorcet winner within the set of critical matchings where nodes are voters. We show that popular critical matchings always exist in G and min-size/max-size such matchings can be efficiently computed.

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

Keywords and phrases Bipartite graphs, Stable matchings, LP-duality

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.25

Funding *Telikepalli Kavitha*: Supported by the Department of Atomic Energy, Government of India, under project no. RTI4001.

1 Introduction

We consider a matching problem in a bipartite graph $G = (A \cup B, E)$ on n nodes and m edges where every node ranks its neighbors in a strict order of preference. Such a graph is also called a *marriage instance*. We seek an optimal matching in G and the classical notion of optimality for matchings in such an instance is *stability* introduced by Gale and Shapley [9] in 1962. A matching M is stable if there is no edge that *blocks* M where an edge (a, b) is said to block M if a and b prefer each other to their respective assignments in M.

Stable matchings always exist in a marriage instance and the Gale-Shapley algorithm finds one in linear time. The Gale-Shapley algorithm and its many-to-one generalization have been used to match students to schools and colleges [1, 2, 17] and graduating medical students to hospitals [4, 21]. All stable matchings in G match the same set of nodes [10]. As discussed in [3], in the medical matching scheme in Scotland, a stable matching left several students unmatched. There was a matching that matched all the students, however this matching admitted some blocking edges. Thus there are real-world applications where the size of the matching is more important than the absence of blocking edges.

More generally, there are applications where certain nodes are prioritized or *critical* and the number of critical nodes that get matched is of primary importance. One such application is the assignment of sailors to billets in the US Navy [22, 26]. Here every sailor has to be matched to a billet and some critical billets cannot be left vacant. So such billets and all the sailors are the critical nodes here. Allocation problems in humanitarian organizations constitute more such applications, see e.g., [24, 25].

Motivated by such applications, we consider the following model where we are given a marriage instance $G = (A \cup B, E)$ along with a set $C \subseteq A \cup B$ of critical nodes. The number of critical nodes that get matched is the most important attribute of a matching. An admissible or critical matching is one that matches as many critical nodes as possible.

▶ **Definition 1.** *A matching M in G is critical if there is no matching in G that matches more critical nodes than M.*

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 25; pp. 25:1–25:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A stable matching need not be critical. When stable matchings are not critical, a natural alternative is to seek a critical matching that admits the least number of blocking edges. However this is an NP-hard problem [3]. It was shown there that finding a maximum matching (so every node is critical here) that admits the minimum number of blocking edges is NP-hard; moreover, this is NP-hard to approximate within $n^{1-\varepsilon}$, for any $\varepsilon > 0$. This motivates relaxing the problem of finding a critical matching with the least number of blocking edges to finding one that satisfies a more relaxed variant of stability. *Popularity* is a natural relaxation of stability that captures welfare in a collective sense.

We say node v prefers matching M to matching N if v prefers its partner in M to its partner in N and being left unmatched is the worst choice for any node. We can compare any pair of matchings M and N by holding an election between them where every node casts a vote for the matching in $\{M, N\}$ that it prefers and it abstains from voting if it is indifferent between M and N. Let $\phi(M, N)$ (resp., $\phi(N, M)$) be the number of votes for M (resp., N) in the M versus N election. Matching N is more popular than matching M if $\phi(N, M) > \phi(M, N)$.

▶ **Definition 2.** *A matching M is popular if $\Delta(N, M) \leq 0$ for all matchings N in G, where $\Delta(N, M) = \phi(N, M) - \phi(M, N)$.*

Thus a matching M is popular if there is no matching that is more popular than M. The notion of popularity was introduced in 1975 by Gärdenfors [11] where he observed that every stable matching is popular. It is easy to decide if there is a popular matching that is also critical – it is known that any node that is matched in some popular matching has to be matched in any max-size popular matching [12]. A max-size popular matching can be computed in linear time [14]. However as was the case with stable matchings, it can be the case that no popular matching is critical. Consider the following example where $A = \{a_0, a_1, a_2\}$ and $B = \{b_0, b_1, b_2\}$. Node preferences are described below.

$a_0\colon\ b_1$	$a_1\colon\ b_1 \succ b_2 \succ b_0$	$a_2\colon\ b_1 \succ b_2$
$b_0\colon\ a_1$	$b_1\colon\ a_1 \succ a_2 \succ a_0$	$b_2\colon\ a_1 \succ a_2$

The node a_0 has only one neighbor b_1. The node a_1 regards b_1 as its top choice, b_2 as its second choice, and b_0 as its third choice. The node a_2 regards b_1 as its top choice and b_2 as its second choice. The preferences of nodes in B are symmetric to those in A.

The above instance has only one stable matching $S = \{(a_1, b_1), (a_2, b_2)\}$. This instance has one more popular matching $P = \{(a_1, b_2), (a_2, b_1)\}$. Suppose $C = \{a_0, a_1\}$ is the set of critical nodes. Then neither S nor P is critical. Here $M_0 = \{(a_0, b_1), (a_1, b_2)\}, M_1 = \{(a_0, b_1), (a_1, b_0)\}$, and $M_2 = \{(a_0, b_1), (a_1, b_0), (a_2, b_2)\}$ are the critical matchings. Thus there need not exist any popular matching that is critical.

A natural alternative is to ask for a critical matching M such that there is no *critical* matching more popular than M. Given that the number of critical nodes that get matched is more important than node preferences, elections that involve non-critical matchings are not relevant since by the definition of our setting, any critical matching is better than any non-critical matching. So the desired matchings are the critical ones and any pair of critical matchings can be compared by holding an election between them. Thus we are only interested in elections between pairs of critical matchings.

▶ **Definition 3.** *A critical matching M is a popular critical matching in G if $\Delta(N, M) \leq 0$ for any critical matching N.*

A popular critical matching is a weak Condorcet winner [5, 18] in the voting instance where every critical matching is a candidate and nodes are voters. The relation "more popular than" is not transitive, i.e., there may be cycles with respect to this relation, so weak Condorcet winners need not exist in every voting instance. It might be the case that for any critical matching, there is a "more popular" critical matching. Interestingly, it was shown in [14] that popular *maximum* matchings (i.e., $C = A \cup B$) always exist in G. Does this positive result hold for every $C \subset A \cup B$? So the following questions are relevant:

- For any $C \subset A \cup B$, does a popular critical matching always exist in G?
- Is it easy to find one?
- Is it easy to find a max-size popular critical matching?

In this paper we show positive answers to all the above questions. Recall that $|E| = m$.

▶ **Theorem 4.** *For any $C \subset A \cup B$, popular critical matchings always exist in $G = (A \cup B, E)$ and a max-size such matching can be computed in $O(|C|m + m)$ time.*

We first show the following result. Then we extend this algorithm to show Theorem 4.

▶ **Theorem 5.** *Given a marriage instance $G = (A \cup B, E)$ along with a subset C of critical nodes, a min-size popular critical matching in G can be computed in $O(|C|m + m)$ time.*

1.1 Background and related results

Algorithmic questions in popular matchings have been well-studied during the last decade and we refer to [6] for a survey. Popular matchings always exist in a marriage instance and efficient algorithms are known to find min-size/max-size popular matchings in a marriage instance [9, 13, 14]. A size-popularity trade-off was shown in [14] to efficiently find matchings whose *unpopularity* is bounded from above and size is bounded from below. As shown there, this implies that a maximum matching that is popular within the set of maximum matchings always exists and can be efficiently computed. So $C = A \cup B$ in [14] while $C = \emptyset$ in the Gale-Shapley algorithm. Thus for the two extreme cases of C, it was known that popular critical matchings always exist and can be efficiently computed.

A related problem is the hospital-residents problem with lower quotas. This is a many-to-one matching problem where every node has a strict preference order over its neighbors and every hospital has a capacity; moreover certain hospitals have lower quotas which denotes the minimum number of residents that have to be matched to this hospital in any feasible matching. It was shown in [19] that whenever feasible matchings exist, a matching that is popular among feasible matchings always exists and a max-size such matching can be computed in polynomial time. Very recently and independent of our work, the above result was generalized in [20] to the setting where certain residents are marked and every marked resident has to be matched in any feasible matching.

Hardness results for "almost stable" critical matchings. Several hardness results for finding *almost* stable maximum matchings (so every vertex is critical) in a marriage instance were shown in [3]. It was shown there that even if all preference lists were restricted to be of length at most 3, finding a maximum matching that admits the minimum number of blocking edges is NP-hard. An alternative approach is to count the number of nodes that are involved in blocking edges [8, 23]. The problem of finding a maximum matching that minimizes this number is also NP-hard to compute/approximate, as shown in [3].

1.2 Techniques

We use the machinery of stable matchings and LP-duality to show our results. We construct a new marriage instance $G' = (A' \cup B', E')$ on $O(|C|n + n)$ nodes and $O(|C|m + m)$ edges such that any stable matching in G' corresponds to a popular critical matching in G. The instance G' resembles instances used in [7, 14, 16] to compute max-size popular matchings and popular maximum matchings.

We now give a quick overview of the popular maximum matching algorithm from [14]. This algorithm partitions the node set $A \cup B$ into *levels* so that any stable matching in this "graph with levels" corresponds to a popular maximum matching in G. To begin with, all nodes are in some level ℓ and the Gale-Shapley algorithm is run on this instance. If the stable matching leaves some nodes in A unmatched then all unmatched nodes in A are *promoted* to level $\ell + 1$. Once promoted to level $\ell + 1$, each such node starts proposing all over again – it will be the case that every node in B prefers higher level neighbors to lower level neighbors. So some of these promoted nodes may find partners.

This may "un-match" some nodes in A initially matched in level ℓ. These nodes continue proposing as per the Gale-Shapley algorithm and any node in A that is unsuccessful in finding a partner in level ℓ gets promoted to level $\ell + 1$. Any node in A that does not find a partner even as a level $\ell + 1$ node gets promoted to level $\ell + 2$ and so on. It was shown in [14] that $|A|$ levels suffice to construct a maximum matching that is popular within the set of maximum matchings.

Our algorithms. If all the critical nodes are in A then the above algorithm easily generalizes to solving the popular critical matching problem by promoting only critical nodes in A to higher levels and non-critical nodes in A will always remain in level ℓ. However we need to deal with critical nodes in the set B as well. For this, our new idea is the following: critical nodes in B that are left unmatched in the Gale-Shapley algorithm in level ℓ get *demoted* to level $\ell - 1$. It will be the case that every node in A prefers *lower level* neighbors to *higher level* neighbors. So in fact, the Gale-Shapley algorithm should begin by nodes in A proposing to lower level neighbors first (before the ones in level ℓ).

Thus the main difference between our instance G' and the earlier instance from [14] (explicitly described in [16]) is that there is non-uniformity among the nodes now. All the nodes in $A \cup B$ are permitted in only one intermediate level, i.e., level ℓ. Non-critical nodes in A are excluded from levels higher than ℓ and non-critical nodes in B are excluded from levels lower than ℓ. We show that any stable matching in G' corresponds to a min-size popular critical matching in G. We construct another instance G'' such that the entire node set $A \cup B$ is permitted in *two* levels: level ℓ and level $\ell + 1$. We show that any stable matching in G'' corresponds to a max-size popular critical matching in G. When $C = \emptyset$, the instance G'' is the same as the instance from [7] whose stable matchings correspond to max-size popular matchings in G.

Our proofs of correctness. We prove the correctness of our algorithms via the LP method by constructing witnesses that certify "popularity within the set of critical matchings" for our matchings. These witnesses are solutions to certain linear programs. Such witnesses are known for popular matchings [15] and popular maximum matchings [16]. Our witnesses are a little more complicated since our primal LP involves more constraints (due to criticality) and so the dual LP has more variables.

The dual LP solutions that we show (see Lemma 11 and Lemma 16) allow us to give simple proofs of correctness and enable us to show (using complementary slackness) that our two algorithms respectively compute min-size and max-size popular critical matchings in G.

By contrast, the proof of correctness of the popular maximum matching algorithm in [14] was combinatorial; popular maximum matchings were characterized in terms of forbidden alternating paths and cycles and it was shown that there was no forbidden alternating path or cycle with respect to the matching returned.

Organization of the paper. Section 2 describes our witness for a popular critical matching. The min-size and max-size popular critical matching algorithms are given in Section 3 and Section 4, respectively.

2 A witness for a popular critical matching

Our input consists of a marriage instance $G = (A \cup B, E)$ with strict preferences and a set $C \subseteq A \cup B$ of critical nodes. We first characterize critical matchings.

▶ **Lemma 6.** *A matching M in G is critical if and only if there is no alternating path p with respect to M that satisfies either of the conditions given below:*

1. *p is an augmenting path with respect to M and at least one endpoint of p is in C.*
2. *p has even length with exactly one endpoint in C and this node is left unmatched in M.*

Proof. Let M be a matching with an alternating path p such that either (i) p is an augmenting path wrt M and at least one endpoint of p is in C or (ii) p has even length with exactly one endpoint in C and this node is left unmatched in M. Then $M \oplus p$ matches at least one more critical node than M. Thus M cannot be a critical matching.

Conversely, suppose M is not a critical matching. Let N be a critical matching. Consider $M \oplus N$. Since N matches more critical nodes than M, there has to be an alternating path p in $M \oplus N$ where N matches more critical nodes than M. So p has an endpoint in C that is matched in N and not in M. If the other endpoint of p is unmatched in M then p is an augmenting path wrt M; else the other endpoint is matched in M and this endpoint is not in C since N matches more critical nodes than M in the alternating path p.

So either (i) p is an augmenting path wrt M and at least one endpoint of p is in C or (ii) p has even length with exactly one endpoint in C, which is left unmatched in M. ◀

Let M be any critical matching in G. Let k_A (resp., k_B) denote the number of nodes in $C_A = C \cap A$ (resp., $C_B = C \cap B$) that are matched in M. The following lemma will be very useful to us.

▶ **Lemma 7.** *Every matching in G matches at most k_A nodes in C_A and at most k_B nodes in C_B.*

Proof. Suppose not. Let N be a matching in G that matches more than k_A nodes in C_A. Then there is an alternating path p in $M \oplus N$ where N matches more nodes of C_A than the critical matching M. If the length of p is odd then p is an augmenting path wrt M whose at least one endpoint is in C. But this is a forbidden structure for any critical matching (by Lemma 6).

So the length of p is even. Then the other endpoint of p (the one matched in M and unmatched in N) is in A, call this node v. Since N matches more nodes of C_A than M in the path p, the node v cannot be in C_A. Hence p is an even length alternating path with exactly one endpoint in C and this node is left unmatched in M. This is again a forbidden structure for any critical matching (by Lemma 6). Thus we get a contradiction. The proof when N matches more than k_B nodes in C_B is analogous. ◀

A linear program for popular critical matchings. It will be convenient to assume that each node considers itself as its last choice neighbor. Let $\tilde{G}$ denote the graph G augmented with self-loops. Any matching M in G can be regarded as a perfect matching $\tilde{M}$ in $\tilde{G}$ by augmenting M with appropriate self-loops. Corresponding to M, an edge weight function wt_M in the graph $\tilde{G}$ can be defined. Let $\mathsf{wt}_M(u,u) = 0$ if u is left unmatched in M, i.e., if (u,u) is in $\tilde{M}$; else $\mathsf{wt}_M(u,u) = -1$. For any edge $(a,b) \in E$:

$$\text{let } \mathsf{wt}_M(a,b) = \begin{cases} 2 & \text{if } (a,b) \text{ blocks } M; \\ -2 & \text{if both } a \text{ and } b \text{ prefer their respective partners in } M \text{ to each other;} \\ 0 & \text{otherwise.} \end{cases}$$

For any $e \in E$, note that $\mathsf{wt}_M(e)$ is the sum of votes of the endpoints of e for each other versus their respective partners in $\tilde{M}$; each vote is in $\{\pm 1, 0\}$ where 1 is “more preferred to” and so on. For any node u, let $\delta(u)$ be the set of edges incident to u in G.

Consider the following linear program (LP1). Note that Lemma 7 implies that all critical matchings in G match k_A nodes in C_A and k_B nodes in C_B. This is used in constraint (2).

$$\text{maximize} \sum_{e \in \tilde{E}} \mathsf{wt}_M(e) \cdot x_e \tag{LP1}$$

subject to

$$\sum_{e \in \delta(u) \cup \{u,u\}} x_e = 1 \quad \forall\, u \in A \cup B \tag{1}$$

$$\sum_{a \in C_A} \sum_{e \in \delta(a)} x_e = k_A \quad \text{and} \quad \sum_{b \in C_B} \sum_{e \in \delta(b)} x_e = k_B \tag{2}$$

$$x_e \geq 0 \quad \forall\, e \in E \cup \{(u,u) : u \in A \cup B\}. \tag{3}$$

We know from Lemma 7 that $\sum_{a \in C_A} \sum_{e \in \delta(a)} x_e \leq k_A$ and $\sum_{b \in C_B} \sum_{e \in \delta(b)} x_e \leq k_B$ are valid inequalities for the matching polytope of $\tilde{G}$. So the feasible region of (LP1) defines a face of the perfect matching polytope of $\tilde{G}$ and hence it is integral. Every integral point in this face corresponds to a critical matching and every critical matching (augmented with self-loops at unmatched nodes) belongs to this face. Thus (LP1) computes a max-weight matching $\tilde{N}$, where N is a critical matching in G.

Consider the dual LP. This is (LP2) given below. The dual variables are y_u for $u \in A \cup B$ along with z_A and z_B.

$$\text{minimize} \sum_{u \in A \cup B} y_u \;+\; (k_A \cdot z_A) \;+\; (k_B \cdot z_B) \tag{LP2}$$

subject to

$$y_a + y_b \geq \mathsf{wt}_M(a,b) \quad \forall\, (a,b) \in E \text{ where } a \notin C_A, b \notin C_B \tag{4}$$

$$y_a + y_b + z_A \geq \mathsf{wt}_M(a,b) \quad \forall\, (a,b) \in E \text{ where } a \in C_A, b \notin C_B \tag{5}$$

$$y_a + y_b + z_B \geq \mathsf{wt}_M(a,b) \quad \forall\, (a,b) \in E \text{ where } a \notin C_A, b \in C_B \tag{6}$$

$$y_a + y_b + z_A + z_B \geq \mathsf{wt}_M(a,b) \quad \forall\, (a,b) \in E \text{ where } a \in C_A, b \in C_B \tag{7}$$

$$y_u \geq \mathsf{wt}_M(u,u) \quad \forall\, u \in A \cup B. \tag{8}$$

▶ **Proposition 8.** *Let M be a critical matching such that the optimal value of* (LP2) *is at most 0. Then M is a popular critical matching.*

Proof. The optimal value of (LP1) is $\max_N \mathsf{wt}_M(\tilde{N})$, where N is a critical matching in G. It follows from the definition of the function wt_M that $\mathsf{wt}_M(\tilde{N}) = \phi(N,M) - \phi(M,N) = \Delta(N,M)$ for any matching N in G. Thus the optimal value of (LP1) is $\max_N \Delta(N,M)$, where N is a critical matching. If the optimal value of (LP2) is at most 0 then the optimal value of (LP1) is also at most 0 (by weak duality). This means $\Delta(N,M) \le 0$ for every critical matching N. ◀

We will use Proposition 8 to prove the correctness of our algorithms in Section 3 and Section 4. That is, we will construct matchings M such that there exist feasible solutions $(\vec{y},\vec{z})$ to (LP2) with $\sum_{u\in A\cup B} y_u \;+\; (k_A \cdot z_A) \;+\; (k_B \cdot z_B) = 0$.

3 An algorithm for a popular critical matching

Let $G = (A \cup B, E)$ be the given marriage instance and let $C \subseteq A \cup B$ be the set of critical nodes. Recall the overview of our algorithm given in Section 1.2. We want to partition the node set $A \cup B$ into *levels* so that any stable matching in this new graph corresponds to a popular critical matching in G.

Recall that we use $C_A = C \cap A$ (resp., $C_B = C \cap B$) to denote the set of critical nodes in A (resp., B). Let $|C_A| = \alpha$ and $|C_B| = \beta$. There will be $\alpha+\beta+1$ levels indexed $0,\ldots,\alpha+\beta$.

A new instance $G' = (A' \cup B', E')$. We now describe a new instance G' whose stable matchings will map to popular critical matchings in G. The set A' is described below.

- For every $a \in C_A$, the set A' has $\alpha+\beta+1$ copies of a: call these nodes $a_0, a_1, \ldots, a_{\alpha+\beta}$.
- For every $a \in A \setminus C_A$, the set A' has $\beta+1$ copies of a: call these nodes $a_0, a_1, \ldots, a_\beta$.

Thus $A' = \cup_{a\in C_A}\{a_0,a_1,\ldots,a_{\alpha+\beta}\} \cup_{a\in A\setminus C_A} \{a_0,a_1,\ldots,a_\beta\}$. Define the set B' as follows. $B' = \{b' : b \in B\} \cup_{a\in C_A} \{d_1(a),\ldots,d_{\alpha+\beta}(a)\} \cup_{a\in A\setminus C_A} \{d_1(a),\ldots,d_\beta(a)\}$.

The set $\{b' : b \in B\}$ is a copy of the set B. Along with nodes in $\{b' : b \in B\}$, the set B' contains *dummy* nodes (the d-nodes). Such dummy nodes were first used in [7] and they make it easy for us to describe "promotions" from one level to another.

When $a \in C_A$, there are $\alpha+\beta+1$ copies of a in A' and the set B' has $d_1(a),\ldots,d_{\alpha+\beta}(a)$. We will set preferences such that in any stable matching in G', $\alpha+\beta$ copies of a have to be matched to these dummy nodes. Similarly, when $a \in A \setminus C_A$, there are $\beta+1$ copies of a in A' and the set B' has $d_1(a),\ldots,d_\beta(a)$. We will set preferences such that in any stable matching in G', β copies of a have to be matched to these dummy nodes. Thus in any stable matching in G', for each $a \in A$, at most *one* node among all a_i's is "free" to be matched to a neighbor in $\{b' : b \in B\}$.

The edge set. Corresponding to each $(a,b) \in E$, we will have the following edges in E'. There are four cases here depending on whether a is in C_A or not and b is in C_B or not.

1. $a \notin C_A$ and $b \notin C_B$: there is exactly one edge (a_β, b') in E' that corresponds to (a,b).
2. $a \notin C_A$ and $b \in C_B$: there are $\beta+1$ edges (a_i,b') in E' where $0 \le i \le \beta$.
3. $a \in C_A$ and $b \notin C_B$: there are $\alpha+1$ edges (a_i,b') in E' where $\beta \le i \le \alpha+\beta$.
4. $a \in C_A$ and $b \in C_B$: there are $\alpha+\beta+1$ edges (a_i,b') in E' where $0 \le i \le \alpha+\beta$.

For each $a \in A$, the set E' also has the following edges:

- if $a \in C_A$ then $(a_{i-1}, d_i(a))$ and $(a_i, d_i(a))$ for $1 \le i \le \alpha+\beta$;
- if $a \in A \setminus C_A$ then $(a_{i-1}, d_i(a))$ and $(a_i, d_i(a))$ for $1 \le i \le \beta$.

For any i, the preference order of $d_i(a)$ is $a_{i-1} \succ a_i$.

Preference orders. Consider $a \in A$. Let a's preference order in G be $b_1 \succ \cdots \succ b_k$. Suppose $\{c_1, \ldots, c_r\} = \{b_1, \ldots, b_k\} \cap C$. That is, $c_1, \ldots, c_r$ are a's critical neighbors. Let a's preference order among these nodes be $c_1 \succ \cdots \succ c_r$.

- a_0's preference order in G' is $c'_1 \succ \cdots \succ c'_r \succ d_1(a)$.
- For $1 \le i \le \beta - 1$, a_i's preference order is $d_i(a) \succ c'_1 \succ \cdots \succ c'_r \succ d_{i+1}(a)$.
- For $a \notin C_A$: the preference order of a_β is $d_\beta(a) \succ b'_1 \succ \cdots \succ b'_k$.
- For $a \in C_A$:
  - for $\beta \le i \le \alpha + \beta - 1$, the preference order of a_i is $d_i(a) \succ b'_1 \succ \cdots \succ b'_k \succ d_{i+1}(a)$.
  - the preference order of $a_{\alpha+\beta}$ is $d_{\alpha+\beta}(a) \succ b'_1 \succ \cdots \succ b'_k$.

For $a \in A$, other than the dummy nodes, observe that it is only copies of critical neighbors that are present in the preference list of a_i for $0 \le i \le \beta - 1$.

For $a \notin C_A$, observe that copies of all neighbors of a, i.e., $b'_1, \ldots, b'_k$, are present only in the preference list of a_β. For $a \in C_A$, copies of all neighbors of a_i are present in the preference list of a_i for $\beta \le i \le \alpha + \beta$.

Consider any $b \in B$. Let b's preference order in G be $a \succ \cdots \succ z$. Let $\{a', \ldots, z'\} = \{a, \ldots, z\} \cap C$. Let b's preference order among its critical neighbors be $a' \succ \cdots \succ z'$. Suppose $b \notin C_B$. Then the preference order of b' in G' is:

$$\underbrace{a'_{\alpha+\beta} \succ \cdots \succ z'_{\alpha+\beta}}_{\text{level } \alpha+\beta \text{ neighbors}} \succ \cdots \succ \underbrace{a'_{\beta+1} \succ \cdots \succ z'_{\beta+1}}_{\text{level } \beta+1 \text{ neighbors}} \succ \underbrace{a_\beta \succ \cdots \succ z_\beta}_{\text{level } \beta \text{ neighbors}}$$

So b' prefers any subscript or level i neighbor to any level j neighbor for $i > j$. Note that copies of only critical neighbors are present in level i for $\beta + 1 \le i \le \alpha + \beta$ and copies of all neighbors of b, i.e., $a, \ldots, z$, are present only in level β.

Suppose $b \in C_B$. Then the preference order of b' in G' is:

$$\underbrace{a'_{\alpha+\beta} \succ \cdots \succ z'_{\alpha+\beta}}_{\text{level } \alpha+\beta \text{ neighbors}} \succ \cdots \succ \underbrace{a'_{\beta+1} \succ \cdots \succ z'_{\beta+1}}_{\text{level } \beta+1 \text{ neighbors}} \succ \underbrace{a_\beta \succ \cdots \succ z_\beta}_{\text{level } \beta \text{ neighbors}} \succ \cdots \succ \underbrace{a_0 \succ \cdots \succ z_0}_{\text{level } 0 \text{ neighbors}}$$

Note that copies of only critical neighbors are present in level i for $\beta + 1 \le i \le \alpha + \beta$ and copies of all neighbors of b are present in level i for $0 \le i \le \beta$.

The matching M. For any stable matching M' in G', define $M \subseteq E$ to be the set of edges obtained by deleting edges in M' that are incident to dummy nodes and replacing any edge $(a_i, b') \in M'$ with the original edge $(a, b) \in E$.

For any $a \in A$ and all $i \ge 1$, the dummy node $d_i(a)$ is the top choice neighbor for a_i, hence the stable matching M' has to match all dummy nodes. Thus at most one node among all the a_i's can be matched in M' to a neighbor in $\{b' : b \in B\}$. So M is a matching in G. Theorem 9 (proved below) is our main theorem in this section.

▶ **Theorem 9.** *For any stable matching M' in G', the corresponding matching M is a min-size popular critical matching in G.*

Since a stable matching always exists in G', popular critical matchings always exist in G. Thus the first part of Theorem 4 follows. The time taken to construct G' and to compute a stable matching in G' is $O(|C|m + m)$. Thus Theorem 5 follows from Theorem 9.

We will prove Theorem 9 now. As done in [14], it will be useful to partition the set $A \cup B$ into subsets as described below (see Fig. 1). We will partition the set of all nodes in A that are matched in M into $A_0 \cup \cdots \cup A_{\alpha+\beta}$ where for $0 \le i \le \alpha + \beta$: $A_i = \{a \in A : (a_i, b') \in M'$

for some $b \in B\}$, i.e., A_i is the collection of those a's such that a_i is matched in M' to a neighbor in $\{b' : b \in B\}$. Add unmatched nodes in C_A to $A_{\alpha+\beta}$ and add unmatched nodes in $A \setminus C_A$ to A_β.

Similarly, partition the set of all nodes in B that are matched in M into $B_0 \cup \cdots \cup B_{\alpha+\beta}$ where for $0 \le i \le \alpha+\beta$: $B_i = \{b : (a_i, b') \in M'$ for some $a \in A_i\}$, i.e., B_i is the collection of those b's such that the partner of b' in M' is a subscript i node. Add unmatched nodes in C_B to B_0 and add unmatched nodes in $B \setminus C_B$ to B_β.

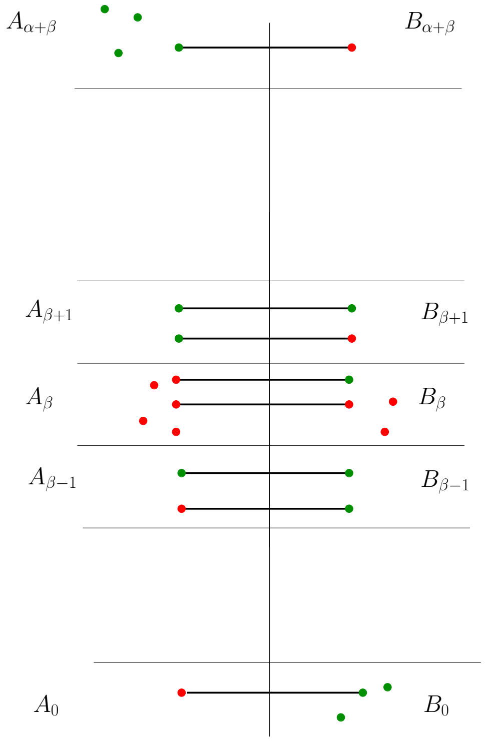


Figure 1 $A = A_0 \cup \cdots \cup A_{\alpha+\beta}$ and $B = B_0 \cup \cdots \cup B_{\alpha+\beta}$ and $M \subseteq \cup_{i=0}^{\alpha+\beta}(A_i \times B_i)$. Red nodes are outside C and green nodes are in C. All red (i.e., non-critical) nodes are in $\cup_{i\le\beta} A_i \cup_{i\ge\beta} B_i$ and unmatched red nodes are in $A_\beta \cup B_\beta$.

▶ **Lemma 10.** *M is a critical matching in G.*

The proof of Lemma 10 (given in the appendix) uses Lemma 6 and is similar to the proof that the popular maximum matching algorithm in [14] finds a maximum matching. Lemma 11 is the main technical result here.

▶ **Lemma 11.** *M is a popular critical matching in G.*

Proof. We will use Proposition 8. Let $(\vec{y}, \vec{z})$ be defined as follows.

1. Set $z_A = -2\alpha$ and $z_B = -2\beta$.
2. Set $y_u = 0$ for all unmatched nodes u. For matched nodes u, we will set y-values as follows. For $0 \le i \le \alpha+\beta$ do:
   - for $a \in A_i$: if $a \in C_A$ then set $y_a = 2\alpha + 2\beta - 2i$; else set $y_a = 2\beta - 2i$.
   - for $b \in B_i$: if $b \in C_B$ then set $y_b = 2i$; else set $y_b = 2i - 2\beta$.

▶ **Lemma 12.** *$\langle\vec{y}, \vec{z}\rangle$ defined above is a feasible solution to* (LP2).

The proof of Lemma 12 is given below (after the proof of Lemma 11). We will now show that $\sum_{u\in A\cup B} y_u + (k_A \cdot z_A) + (k_B \cdot z_B) = 0$. Consider any edge $(a, b) \in M$. So there is some $i \in \{0, \ldots, \alpha + \beta\}$ such that $a \in A_i$ and $b \in B_i$.

1. If $a \notin C_A$ and $b \notin C_B$ then $y_a + y_b = (2\beta - 2i) + (2i - 2\beta) = 0$.
2. If $a \in C_A$ and $b \notin C_B$ then $y_a + y_b + z_A = (2\alpha + 2\beta - 2i) + (2i - 2\beta) - 2\alpha = 0$.
3. If $a \notin C_A$ and $b \in C_B$ then $y_a + y_b + z_B = (2\beta - 2i) + 2i - 2\beta = 0$.
4. If $a \in C_A$ and $b \in C_B$ then $y_a + y_b + z_A + z_B = (2\alpha + 2\beta - 2i) + 2i - 2\alpha - 2\beta = 0$.

Recall that k_A (resp., k_B) is the number of nodes from C_A (resp., C_B) that get matched in any critical matching. Since M is a critical matching (by Lemma 10), it matches k_A nodes from C_A and k_B nodes from C_B. So added up over all edges (a, b) in M, the left hand sides of the four equations above sum to $\sum_{u\in V} y_u + (k_A \cdot z_A) + (k_B \cdot z_B)$, where $V \subseteq A \cup B$ is the set of nodes matched in M. Since all the right hand sides are 0, this sum is 0. For any unmatched node u, we set $y_u = 0$. So $\sum_{u\in A\cup B} y_u + (k_A \cdot z_A) + (k_B \cdot z_B) = 0$. Hence M is a popular critical matching in G (by Proposition 8). ◀

Proof of Lemma 12. For any node u, we claim that $y_u \geq 0$. Recall that $y_a = 2\alpha + 2\beta - 2i$ for a matched critical node $a \in A_i$ and $y_b = 2i$ for a matched critical node $b \in B_i$. Since $0 \leq i \leq \alpha + \beta$, we have $2\alpha + 2\beta - 2i \geq 0$ and $2i \geq 0$. Thus for any matched node $u \in C$, $y_u \geq 0$.

For any matched node $a \in A \setminus C_A$, observe that $a \in A_i$ for some $i \leq \beta$, so $2\beta - 2i \geq 0$. For any matched node $b \in B \setminus C_B$, observe that $b \in B_i$ for some $i \geq \beta$, so $2i - 2\beta \geq 0$. We set $y_u = 0$ for any unmatched node u. Hence $y_u \geq 0 \geq \mathsf{wt}_M(u, u)$ for all $u \in A \cup B$. Thus constraint (8) holds.

We will now show that $\langle \vec{y}, \vec{z} \rangle$ satisfies constraints (4)-(7). For any $a \in C_A$, let $y'_a = y_a + z_A$. For any $b \in C_B$, let $y'_b = y_b + z_B$. For any node $u \notin C$, let $y'_u = y_u$.

- We have $y'_a = 2\beta - 2i$ for any matched $a \in A_i$ and $y'_b = 2i - 2\beta$ for any matched $b \in B_i$.
- For any unmatched $a \in A$: $y'_a = -2\alpha$ if $a \in C_A$ and $y'_a = 0$ otherwise.
- For any unmatched $b \in B$: $y'_b = -2\beta$ if $b \in C_B$ and $y'_b = 0$ otherwise.

We will now show that $y'_a + y'_b \geq \mathsf{wt}_M(a, b)$ for all $(a, b) \in E$. Let $a \in A_i$ and $b \in B_j$. This proof is split into 4 parts: (1) $i \leq j - 1$, (2) $i = j$, (3) $i = j + 1$, and (4) $i \geq j + 2$.

1. Consider any edge (a, b) where $a \in A_i, b \in B_j$ and $i \leq j - 1$.
   - If a and b are matched nodes then $y'_a + y'_b = (2\beta - 2i) + (2j - 2\beta) = 2(j - i) \geq 2 \geq \mathsf{wt}_M(a, b)$ since $\mathsf{wt}_M(e) \in \{\pm 2, 0\}$ for all $e \in E$.
   - Suppose a is unmatched. Then $a \notin C_A$; otherwise $i = \alpha + \beta$ and so $j \geq \alpha + \beta + 1$ which is not possible. So $a \notin C_A$ and we have $y'_a = 0$ and $i = \beta$. Since $j \geq \beta + 1$, we have $y'_b = 2j - 2\beta \geq 2$. Thus $y'_a + y'_b \geq 2 \geq \mathsf{wt}_M(a, b)$.
   - Suppose b is unmatched. Then $b \notin C_B$; otherwise $j = 0$ and so $i \leq -1$ which is not possible. So $b \notin C_B$ and we have $y'_b = 0$ and $j = \beta$. Since $i \leq \beta - 1$, we have $y'_a = 2\beta - 2i \geq 2$. Thus $y'_a + y'_b \geq 2 \geq \mathsf{wt}_M(a, b)$.
2. Let $a \in A_i, b \in B_j$ where $i = j$. For any $b \in B$, within subscript i neighbors, the preference order of b' in G' is the same as b's preference order among these neighbors in G. Thus M restricted to $A_i \cup B_i$ is stable and so $\mathsf{wt}_M(a, b) \in \{-2, 0\}$.
   - If a and b are matched nodes then $y'_a + y'_b = (2\beta - 2i) + (2i - 2\beta) = 0$.
   - Suppose a is unmatched.
     - If $a \in C_A$ then $y'_a = -2\alpha$ and $i = \alpha + \beta$. So $y'_b = 2(\alpha + \beta) - 2\beta = 2\alpha$. Thus $y'_a + y'_b = -2\alpha + 2\alpha = 0$.

  - If $a \notin C_A$ then $y'_a = 0$ and $i = \beta$. The node b has to be matched since M' is stable (and thus maximal) in G'. So $y'_b = 2i - 2\beta = 0$. Thus $y'_a + y'_b = 0$.
- Suppose b is unmatched.
  - If $b \in C_B$ then $y'_b = -2\beta$ and $i = 0$. So $y'_a = 2\beta - 2i = 2\beta$. Thus $y'_a + y'_b = 2\beta - 2\beta = 0$.
  - If $b \notin C_B$ then $y'_b = 0$ and $i = \beta$. The node a has to be matched since M' is stable (and thus maximal) in G'. So $y'_a = 2\beta - 2i = 0$. Thus $y'_a + y'_b = 0$.

Thus we have $y'_a + y'_b = 0 \geq \mathsf{wt}_M(a,b)$ in all the cases.

3. Let $a \in A_i, b \in B_j$ where $i = j+1$. Observe that $(a_j, d_{j+1}(a)) \in M'$, i.e., a_j is matched to its least preferred neighbor $d_{j+1}(a)$. The stability of M' implies that $(u_j, b') \in M'$ for some neighbor u_j that b' prefers to a_j. Also b' prefers a_{j+1} to u_j, so a_{j+1} has to prefer $M'(a_{j+1})$ to b'. Hence both a and b are matched in M to neighbors that they prefer to each other. So $\mathsf{wt}_M(a,b) = -2$. Thus $y'_a + y'_b = (2\beta - 2(j+1)) + (2j - 2\beta) = -2 = \mathsf{wt}_M(a,b)$.
4. If $a \in A_i, b \in B_j$ where $i \geq j+2$ then $(a_{j+1}, d_{j+2}(a)) \in M'$, i.e., a_{j+1} is matched to its least preferred neighbor $d_{j+2}(a)$. This means the edge (a_{j+1}, b') blocks M' – this is because b' prefers a_{j+1} to its assignment in M': this is either a subscript j neighbor or b' is left unmatched in M'. Since the blocking edge (a_{j+1}, b') contradicts M''s stability, there is no $(a,b) \in E$ where $a \in A_i, b \in B_j$ and $i \geq j+2$.

Thus we have $y'_a + y'_b \geq \mathsf{wt}_M(a,b)$ for all $(a,b) \in E$. This completes the proof of Lemma 12. ◀

Min-size popular critical matching. Lemma 12 showed that $(\vec{y}, \vec{z})$ is a feasible solution to (LP2). In fact, $(\vec{y}, \vec{z})$ is an optimal solution to (LP2) since $\tilde{M}$ is a feasible solution to (LP1) and $\mathsf{wt}_M(\tilde{M}) = 0 = \sum_{u \in A \cup B} y_u + (k_A \cdot z_A) + (k_B \cdot z_B)$. This will be useful in Lemma 13.

▶ **Lemma 13.** *M is a min-size popular critical matching in G.*

Proof. Let N be a critical matching of size smaller than $|M|$. Then there is some node u that is matched in M but unmatched in N. So the self-loop (u,u) is in the perfect matching $\tilde{N}$. For any node u matched in M, we have $y_u > \mathsf{wt}_M(u,u)$. This is because $y_u \geq 0$ while $\mathsf{wt}_M(u,u) = -1$. So the self-loop (u,u) is *slack* with respect to the dual optimal solution $(\vec{y}, \vec{z})$. Then complementary slackness implies that $\tilde{N}$ cannot be a primal optimal solution. The optimal value of (LP1) is 0, so this means $\mathsf{wt}_M(\tilde{N}) < 0$, i.e., $\Delta(N, M) < 0$. Hence the critical matching M is more popular than N. Thus N cannot be a popular critical matching. So M is a min-size popular critical matching in G. ◀

4 Finding a max-size popular critical matching

In this section we consider the problem of finding a *max-size* popular critical matching in $G = (A \cup B, E)$ where $C \subseteq A \cup B$ is the given critical set. We will construct a new instance $G'' = (A'' \cup B'', E'')$ which will be a minor variant of the instance G' seen in Section 3. The instance G' was motivated by considering that we ran the Gale-Shapley algorithm with all nodes in level ℓ (note that $\ell = \beta$) and promoted unmatched critical nodes in A to higher levels and demoted unmatched critical nodes in B to lower levels.

The instance G'' can be motivated by considering that we will run the max-size popular matching algorithm [14] (also called the *2-level Gale-Shapley* algorithm) with all the nodes in level β. This promotes certain nodes to level $\beta + 1$; all unmatched nodes in A are in level $\beta + 1$ and all unmatched nodes in B are in level β. Now let us promote unmatched critical nodes in A to higher levels and demote unmatched critical nodes in B downwards.

The instance G''. The instance $G'' = (A'' \cup B'', E'')$ has *one* extra level compared to G'.

- For every $a \in C_A$, the set A'' has $\alpha + \beta + 2$ copies of a: call them $a_0, a_1, \ldots, a_{\alpha+\beta+1}$.
- For every $a \in A \setminus C_A$, the set A'' has $\beta + 2$ copies of a: call them $a_0, a_1, \ldots, a_{\beta+1}$.

So $A'' = \cup_{a \in C_A}\{a_0, a_1, \ldots, a_{\alpha+\beta+1}\} \cup_{a \in A \setminus C_A} \{a_0, a_1, \ldots, a_{\beta+1}\}$. The set B'' is defined as follows. $B'' = \{b' : b \in B\} \cup_{a \in C_A} \{d_1(a), \ldots, d_{\alpha+\beta+1}(a)\} \cup_{a \in A \setminus C_A} \{d_1(a), \ldots, d_{\beta+1}(a)\}$.

As before, $\{b' : b \in B\}$ is a copy of the set B; along with nodes in $\{b' : b \in B\}$, the set B'' contains $\alpha + \beta + 1$ dummy nodes $d_1(a), \ldots, d_{\alpha+\beta+1}(a)$ for $a \in C_A$ and $\beta + 1$ dummy nodes $d_1(a), \ldots, d_{\beta+1}(a)$ for $a \in A \setminus C_A$.

The edge set. Corresponding to each $(a, b) \in E$, we have the following edges in E''. As before, there are four cases depending on whether a (similarly, b) is critical or not.

1. $a \notin C_A$ and $b \notin C_B$: there are *two* edges (a_β, b') and $(a_{\beta+1}, b')$ that correspond to (a, b).
2. $a \notin C_A$ and $b \in C_B$: there are $\beta + 2$ edges (a_i, b') where $0 \le i \le \beta + 1$.
3. $a \in C_A$ and $b \notin C_B$: there are $\alpha + 2$ edges (a_i, b') where $\beta \le i \le \alpha + \beta + 1$.
4. $a \in C_A$ and $b \in C_B$: there are $\alpha + \beta + 2$ edges (a_i, b') where $0 \le i \le \alpha + \beta + 1$.

For $a \in A \setminus C_A$, the set E'' has the edges $(a_{i-1}, d_i(a))$ and $(a_i, d_i(a))$ where $1 \le i \le \beta + 1$. For $a \in C_A$, the set E'' has the edges $(a_{i-1}, d_i(a))$ and $(a_i, d_i(a))$ where $1 \le i \le \alpha + \beta + 1$. For any $i \ge 1$, the preference order of $d_i(a)$ is $a_{i-1} \succ a_i$.

Preference orders. Let a's preference order in G be $b_1 \succ \cdots \succ b_k$. Let $\{c_1, \ldots, c_r\} = \{b_1, \ldots, b_k\} \cap C$. That is, $c_1, \ldots, c_r$ are a's critical neighbors. It will be the case that only these nodes can be neighbors of $a_0, \ldots, a_{\beta-1}$. Let a's preference order among these nodes be $c_1 \succ \cdots \succ c_r$.

- a_0's preference order is $c'_1 \succ \cdots \succ c'_r \succ d_1(a)$.
- For $1 \le i \le \beta - 1$, the preference order of a_i is $d_i(a) \succ c'_1 \succ \cdots \succ c'_r \succ d_{i+1}(a)$.
- For $a \notin C_A$:
  - the preference order of a_β is $d_\beta(a) \succ b'_1 \succ \cdots \succ b'_k \succ d_{\beta+1}(a)$;
  - the preference order of $a_{\beta+1}$ is $d_{\beta+1}(a) \succ b'_1 \succ \cdots \succ b'_k$.
- For $a \in C_A$:
  - for $\beta \le i \le \alpha + \beta$, the preference order of a_i is $d_i(a) \succ b'_1 \succ \cdots \succ b'_k \succ d_{i+1}(a)$;
  - the preference order of $a_{\alpha+\beta+1}$ is $d_{\alpha+\beta+1}(a) \succ b'_1 \succ \cdots \succ b'_k$.

Consider any $b \in B$. Let its preference order in G be $a \succ \cdots \succ z$. Let b's critical neighbors be $a', \ldots, z'$ and let b's preference order among them be $a' \succ \cdots \succ z'$.

Suppose $b \notin C_B$. Then the preference order of b' is

$$\underbrace{a'_{\alpha+\beta+1} \succ \cdots \succ z'_{\alpha+\beta+1}}_{\text{level } \alpha+\beta+1 \text{ neighbors}} \succ \cdots \succ \underbrace{a'_{\beta+2} \succ \cdots \succ z'_{\beta+2}}_{\text{level } \beta+2 \text{ neighbors}} \succ \underbrace{a_{\beta+1} \succ \cdots \succ z_{\beta+1}}_{\text{level } \beta+1 \text{ neighbors}} \succ \underbrace{a_\beta \succ \cdots \succ z_\beta}_{\text{level } \beta \text{ neighbors}}$$

Note that copies of only critical neighbors are present in level i for $\beta + 2 \le i \le \alpha + \beta + 1$ and copies of all neighbors of b, i.e., $a, \ldots, z$, are present only in levels β and $\beta + 1$.

Suppose $b \in C_B$. Then the preference order of b' is

$$\underbrace{a'_{\alpha+\beta+1} \succ \cdots \succ z'_{\alpha+\beta+1}}_{\text{level } \alpha+\beta+1 \text{ neighbors}} \succ \cdots \succ \underbrace{a'_{\beta+2} \succ \cdots \succ z'_{\beta+2}}_{\text{level } \beta+2 \text{ neighbors}} \succ \underbrace{a_{\beta+1} \succ \cdots \succ z_{\beta+1}}_{\text{level } \beta+1 \text{ neighbors}} \succ \cdots \succ \underbrace{a_0 \succ \cdots \succ z_0}_{\text{level } 0 \text{ neighbors}}$$

Note that copies of only critical neighbors are present in level i for $\beta + 2 \le i \le \alpha + \beta + 1$ and copies of all neighbors of b are present in level i for $0 \le i \le \beta + 1$.

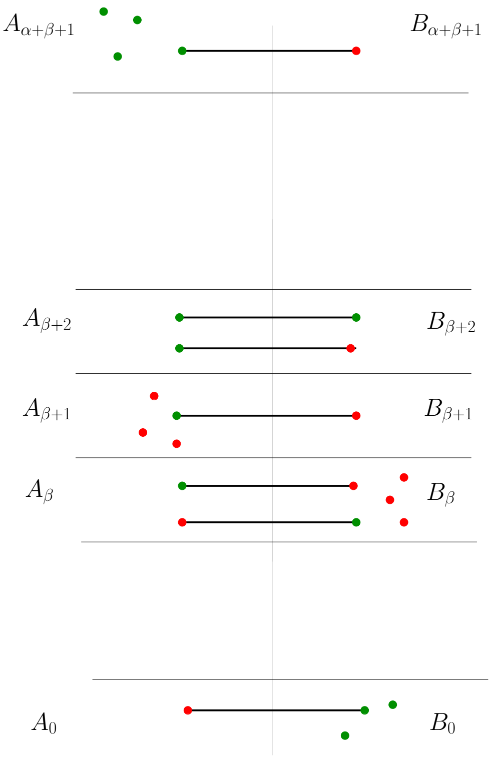


Figure 2 $A = A_0 \cup \cdots \cup A_{\alpha+\beta+1}$ and $B = B_0 \cup \cdots \cup B_{\alpha+\beta+1}$ and $M \subseteq \cup_{i=0}^{\alpha+\beta+1}(A_i \times B_i)$. Red nodes are outside C and green nodes are in C. All red (i.e., non-critical) nodes are in $\cup_{i \le \beta+1} A_i \cup_{i \ge \beta} B_i$; unmatched red nodes are in $A_{\beta+1} \cup B_\beta$.

The matching M. For any stable matching M'' in G'', define $M \subseteq E$ to be the set of edges obtained by deleting edges in M'' that are incident to dummy nodes and replacing any edge $(a_i, b') \in M''$ with the original edge $(a, b) \in E$. For each $a \in A$, the stable matching M'' matches at most one node among all a_i's to a neighbor in $\{b' : b \in B\}$ (the other a_i's have to be matched to dummy nodes). So M is a matching in G.

▶ **Theorem 14.** *For any stable matching M'' in G'', the corresponding matching M is a max-size popular critical matching in G.*

We will prove Theorem 14 by first showing that M is a critical matching (see Lemma 15), then that M is a popular critical matching (see Lemma 16), and finally that M is a max-size popular critical matching (see Lemma 19). The proof of Lemma 15 is similar to the proof of Lemma 10 and is given in the appendix.

▶ **Lemma 15.** *M is a critical matching in G.*

We will now prove that M is a popular critical matching. In order to show this, our analysis is totally analogous to our analysis in Section 3. As done there, we partition the set of all nodes in A that are matched in M into $A_0 \cup \cdots \cup A_{\alpha+\beta+1}$ where for $0 \le i \le \alpha+\beta+1$: $A_i = \{a \in A : (a_i, b') \in M'' \text{ for some } b \in B\}$, i.e., A_i is the set of all a's in A such that a_i is matched in M'' to a neighbor in $\{b' : b \in B\}$. Add unmatched nodes in C_A to the set $A_{\alpha+\beta+1}$ and unmatched nodes in $A \setminus C_A$ to the set $A_{\beta+1}$ (see Fig. 2).

Partition the set of all nodes in B that are matched in M into $B_0 \cup \cdots \cup B_{\alpha+\beta+1}$ where for $0 \le i \le \alpha+\beta+1$: $B_i = \{b : (a_i, b') \in M''$ for some $a \in A_i\}$, i.e., b''s partner in M'' is a subscript i node. Add unmatched nodes in C_B to the set B_0 and unmatched nodes in $B \setminus C_B$ to the set B_β.

▶ **Lemma 16.** *M is a popular critical matching in G.*

Proof. We will use Proposition 8 here. Let $(\vec{y}, \vec{z})$ be defined as follows.

1. Set $z_A = -2\alpha$ and $z_B = -2\beta$. Set $y_u = 0$ for all unmatched nodes u.
2. For matched nodes u, we will set y-values as follows.
   - For $a \in A_i$: if $a \in C_A$ then set $y_a = 2\alpha + 2\beta - 2i + 1$; else set $y_a = 2\beta - 2i + 1$.
   - For $b \in B_i$: if $b \in C_B$ then set $y_b = 2i - 1$; else set $y_b = 2i - 2\beta - 1$.

▶ **Lemma 17.** *$\langle \vec{y}, \vec{z} \rangle$ defined above is a feasible solution to* (LP2).

The proof of Lemma 17 is given below. We will now show that $\sum_{u \in A \cup B} y_u + (k_A \cdot z_A) + (k_B \cdot z_B) = 0$. Consider any edge $(a, b) \in M$. There is some $i \in \{0, \ldots, \alpha+\beta+1\}$ such that $a \in A_i$ and $b \in B_i$.

1. If $a \notin C_A$ and $b \notin C_B$ then $y_a + y_b = (2\beta - 2i + 1) + (2i - 2\beta - 1) = 0$.
2. If $a \in C_A$ and $b \notin C_B$ then $y_a + y_b + z_A = (2\alpha + 2\beta - 2i + 1) + (2i - 2\beta - 1) - 2\alpha = 0$.
3. If $a \notin C_A$ and $b \in C_B$ then $y_a + y_b + z_B = (2\beta - 2i + 1) + (2i - 1) - 2\beta = 0$.
4. If $a \in C_A$ and $b \in C_B$ then $y_a + y_b + z_A + z_B = (2\alpha + 2\beta - 2i + 1) + (2i - 1) - 2\alpha - 2\beta = 0$.

Recall that k_A (resp., k_B) is the number of nodes from C_A (resp., C_B) that get matched in any critical matching. Since M is a critical matching (by Lemma 15), added up over all edges (a, b) in M, the left hand sides of the four equations above sum to $\sum_{u \in V} y_u + (k_A \cdot z_A) + (k_B \cdot z_B)$, where $V \subseteq A \cup B$ is the set of nodes matched in M. Since all the right hand sides are 0, this sum is 0. For any unmatched node u, we set $y_u = 0$. Hence $\sum_{u \in A \cup B} y_u + (k_A \cdot z_A) + (k_B \cdot z_B) = 0$. Thus M is a popular critical matching in G (by Proposition 8). ◀

Proof of Lemma 17. For any unmatched node u, we have $\mathsf{wt}_M(u, u) = 0$ and we set $y_u = 0$. For any matched node u, we have $\mathsf{wt}_M(u, u) = -1$ and we will now show that $y_u \ge -1$. Since $0 \le i \le \alpha + \beta + 1$, we have $2\alpha + 2\beta - 2i + 1 \ge -1$ and $2i - 1 \ge -1$. Thus for any matched critical node u, we have $y_u \ge -1$.

For any matched $a \in A \setminus C_A$, observe that $a \in A_i$ for some $0 \le i \le \beta + 1$, so $y_a = 2\beta - 2i + 1 \ge -1$. For any matched $b \in B \setminus C_B$, observe that $b \in B_i$ for some $\beta \le i \le \alpha + \beta$, so $y_a = 2i - 2\beta - 1 \ge -1$. Hence $y_u \ge \mathsf{wt}_M(u, u)$ for all $u \in A \cup B$. Thus constraint (8) holds.

We will now show that $\langle \vec{y}, \vec{z} \rangle$ satisfies constraints (4)-(7). For any $a \in C_A$, let $y'_a = y_a + z_A$ and for any $b \in C_B$, let $y'_b = y_b + z_B$. For any node $u \notin C$, let $y'_u = y_u$.

- We have $y'_a = 2\beta - 2i + 1$ for any matched $a \in A$ and $y'_b = 2i - 2\beta - 1$ for any matched $b \in B$.
- For any unmatched $a \in A$: $y'_a = -2\alpha$ if $a \in C_A$ and $y'_a = 0$ otherwise.
- For any unmatched $b \in B$: $y'_b = -2\beta$ if $b \in C_B$ and $y'_b = 0$ otherwise.

We are now ready to show that $y'_a + y'_b \ge \mathsf{wt}_M(a, b)$ for all $(a, b) \in E$. Let $a \in A_i$ and $b \in B_j$. As done in the proof of Lemma 12, this proof is split into 4 parts: (1) $i \le j - 1$, (2) $i = j$, (3) $i = j + 1$, and (4) $i \ge j + 2$.

1. Consider any edge (a, b) where $a \in A_i, b \in B_j$, and $i \le j - 1$.
   - If a and b are matched nodes then $y'_a + y'_b = (2\beta - 2i + 1) + (2j - 2\beta - 1) = 2(j - i) \ge 2 \ge \mathsf{wt}_M(a, b)$.

- Suppose a is unmatched. Observe that $a \in A \setminus C_A$; otherwise $i = \alpha + \beta + 1$ and so $j \geq \alpha + \beta + 2$ which is not possible. Since $a \in A \setminus C_A$, we have $y'_a = 0$ and $i = \beta + 1$. Since $j \geq \beta + 2$, we have $y'_b = 2j - 2\beta - 1 \geq 3$. Thus $y'_a + y'_b \geq 3 > \mathsf{wt}_M(a, b)$.
- Suppose b is unmatched. Observe that $b \in B \setminus C_B$; otherwise $j = 0$ and so $i \leq -1$ which is not possible. Since $b \in B \setminus C_B$, we have $y'_b = 0$ and $j = \beta$. Since $i \leq \beta - 1$, we have $y'_a = 2\beta - 2i + 1 \geq 3$. Thus $y'_a + y'_b \geq 3 > \mathsf{wt}_M(a, b)$.

2. Consider any $(a, b) \in E$ where $a \in A_i$ and $b \in B_i$. For any $b \in B$, within subscript i neighbors, the preference order of b' in G'' is the same as b's preference order among these neighbors in G. Thus M restricted to $A_i \cup B_i$ is stable and so $\mathsf{wt}_M(a, b) \leq 0$.
- If a and b are matched nodes then $y'_a + y'_b = (2\beta - 2i + 1) + (2i - 2\beta - 1) = 0$.
- Suppose a is unmatched.
  - If $a \in C_A$ then $y'_a = -2\alpha$ and $i = \alpha + \beta + 1$. So $y'_b = 2(\alpha + \beta + 1) - 2\beta - 1 = 2\alpha + 1$. Thus $y'_a + y'_b = -2\alpha + 2\alpha + 1 = 1$.
  - If $a \notin C_A$ then $y'_a = 0$ and $i = \beta + 1$. So $y'_b = 2(\beta + 1) - 2\beta - 1 = 1$. Thus $y'_a + y'_b = 1$.
- Suppose b is unmatched.
  - If $b \in C_B$ then $y'_b = -2\beta$ and $i = 0$. So $y'_a = 2\beta + 1$. Thus $y'_a + y'_b = 2\beta + 1 - 2\beta = 1$.
  - If $b \notin C_B$ then $y'_b = 0$ and $i = \beta$. So $y'_a = 2\beta - 2\beta + 1 = 1$. Thus $y'_a + y'_b = 1$.

Thus we have $y'_a + y'_b \geq 0 \geq \mathsf{wt}_M(a, b)$ in all the cases.

3. Let $b \in B_j$ where $i = j + 1$. As argued in the proof of Lemma 12, case 3, for any edge (a, b) where $a \in A_{j+1}$ and $b \in B_j$, we have $\mathsf{wt}_M(a, b) = -2$. So both a and b are matched in M to neighbors they prefer to each other. So $y'_a + y'_b = (2\beta - 2i + 1) + (2(i - 1) - 2\beta - 1) = -2 = \mathsf{wt}_M(a, b)$.

4. There is no edge (a, b) where $b \in B_j$ and $i \geq j + 2$; otherwise (a_{j+1}, b') would block M'' as shown in the proof of Lemma 12, case 4.

Thus we have shown that $y'_a + y'_b \geq \mathsf{wt}_M(a, b)$ for all $(a, b) \in E$. This completes the proof of Lemma 17. ◀

Max-size popular critical matching. Observe that $(\vec{y}, \vec{z})$ is an optimal solution to (LP2) since $\tilde{M}$ is a feasible solution to (LP1) and $\mathsf{wt}_M(\tilde{M}) = 0 = \sum_{u \in A \cup B} y_u + (k_A \cdot z_A) + (k_B \cdot z_B)$. We will use the notation y'_v for $v \in A \cup B$ used in the proof of Lemma 17. Recall that for any $a \in C_A$, $y'_a = y_a + z_A$ and for any $b \in C_B$, $y'_b = y_b + z_B$. For any node $u \notin C$, $y'_u = y_u$. We will show the following claim below.

▷ Claim 18. For any edge (a, b) where a or b is unmatched, $y'_a + y'_b > \mathsf{wt}_M(a, b)$.

Proof. Consider any unmatched $a \in A$ and let $(a, b) \in E$. We already know from the proof of Lemma 17 that $y'_a + y'_b \geq \mathsf{wt}_M(a, b)$. Our goal now is to show that $y'_a + y'_b > \mathsf{wt}_M(a, b)$.

If $a \in C_A$ then $a \in A_{\alpha+\beta+1}$. Observe that $b \in B_{\alpha+\beta+1}$, otherwise the edge $(a_{\alpha+\beta+1}, b')$ would block M'. So $y'_a + y'_b = -2\alpha + 2\alpha + 1 = 1$. Since $\mathsf{wt}_M(a, b) \in \{0, \pm 2\}$, this means $y'_a + y'_b > \mathsf{wt}_M(a, b)$.

If $a \notin C_A$ then $a \in A_{\beta+1}$. Observe that $b \in \cup_{i \geq \beta+1} B_i$, otherwise the edge $(a_{\beta+1}, b')$ would block M'. If $b \in B_{\beta+1}$ then $y'_a + y'_b = 0 + 2(\beta + 1) - 2\beta - 1 = 1$ and so $y'_a + y'_b > \mathsf{wt}_M(a, b)$. If $b \in \cup_{i \geq \beta+2} B_i$ then $y'_a + y'_b \geq 0 + 2(\beta + 2) - 2\beta - 1 = 3 > \mathsf{wt}_M(a, b)$.

Consider any unmatched $b \in B$ and let $(a, b) \in E$. If $b \in C_B$ then $b \in B_0$. Observe that $a \in A_0$, otherwise the edge (a_0, b') would block M'. So $y'_a + y'_b = 2\beta + 1 - 2\beta = 1$. Since $\mathsf{wt}_M(a, b) \in \{0, \pm 2\}$, it follows that $y'_a + y'_b > \mathsf{wt}_M(a, b)$.

If $b \notin C_B$ then $b \in B_\beta$. Observe that $a \in \cup_{i \le \beta} A_i$, otherwise the edge (a_β, b') would block M'. If $a \in A_\beta$ then $y'_a + y'_b = 2\beta - 2\beta + 1 = 1$ and so $y'_a + y'_b > \mathsf{wt}_M(a, b)$. If $a \in \cup_{i \le \beta - 1} A_i$ then $y'_a + y'_b \ge 2\beta - 2(\beta - 1) + 1 = 3 > \mathsf{wt}_M(a, b)$.

Thus every edge incident to a node left unmatched in M is slack. ◁

Lemma 19 follows easily from Claim 18.

▶ **Lemma 19.** *M is a max-size popular critical matching in G.*

Proof. Consider any critical matching N in G such that $|N| > |M|$. So N has to match a node that is unmatched in M, i.e., N has to use a slack edge (by Claim 18). Since $(\vec{y}, \vec{z})$ is an optimal solution to (LP2), it follows from complementary slackness that the perfect matching $\tilde{N}$, which is a feasible solution to (LP1), cannot be an optimal solution.

The optimal value of (LP1) is 0, so this means $\mathsf{wt}_M(\tilde{N}) < 0$. In other words, $\Delta(N, M) < 0$, i.e., the critical matching M is more popular than N. Thus no critical matching larger than M can be a popular critical matching. Hence M is a max-size popular critical matching. ◀

The time taken to compute M is $O(|C|m + m)$, so the second part of Theorem 4 follows from Theorem 14. Recall that the first part of Theorem 4 was already shown in Section 3.

References

1 A. Abdulkadiroğlu and T. Sönmez. School choice: a mechanism design approach. *American Economic Review*, 93(3):729–747, 2003.

2 S. Baswana, P. P. Chakrabarti, S. Chandran, Y. Kanoria, and U. Patange. Centralized admissions for engineering colleges in India. *INFORMS Journal on Applied Analytics*, 49(5):338–354, 2019.

3 P. Biro, D. F. Manlove, and S. Mittal. Size versus stability in the marriage problem. *Theoretical Computer Science*, 411:1828–1841, 2010.

4 Canadian Resident Matching Service. How the matching algorithm works. http://carms.ca/algorithm.htm.

5 M.-J.-A.-N. de C. (Marquis de) Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix.* L'Imprimerie Royale, 1785.

6 Á. Cseh. Popular matchings. Trends in Computational Social Choice, Ulle Endriss (ed.), 2017.

7 Á. Cseh and T. Kavitha. Popular edges and dominant matchings. *Mathematical Programming*, 172(1):209–229, 2018.

8 K. Eriksson and O. Häggström. Instability of matchings in decentralized markets with various preference orders. *Mathematical Programming*, 36(3-4):409–420, 2008.

9 D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.

10 D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11(3):223–232, 1985.

11 P. Gärdenfors. Match making: assignments based on bilateral preferences. *Behavioural Science*, 20:166–173, 1975.

12 M. Hirakawa, Y. Yamauchi, S. Kijima, and M. Yamashita. On the structure of popular matchings in the stable marriage problem: Who can join a popular matching? In the 3rd International Workshop on Matching Under Preferences (MATCH-UP), 2015.

13 C.-C. Huang and T. Kavitha. Popular matchings in the stable marriage problem. *Information and Computation*, 222:180–194, 2013.

14 T. Kavitha. A size-popularity tradeoff in the stable marriage problem. *SIAM Journal on Computing*, 43(1):52–71, 2014.

15 T. Kavitha. Popular half-integral matchings. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 22:1–22:13, 2016.

16 T. Kavitha. Maximum matchings and popularity. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 85:1–85:21, 2021.

17 C. Mathieu. Stable matching in practice. In the 26th Annual European Symposium on Algorithms (ESA), Keynote talk, 2018.

18 S. Merrill and B. Grofman. *A unified theory of voting: directional and proximity spatial models.* Cambridge University Press, 1999.

19 M. Nasre and P. Nimbhorkar. Popular matchings with lower quotas. In *Proceedings of the 37th Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 44:1–44:15, 2017.

20 M. Nasre, P. Nimbhorkar, K. Ranjan, and A. Sarkar. Popular matchings in the hospitals-residents problem with two-sided lower quotas. In *Proceedings of the 41st Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2021.

21 National Resident Matching Program. Why the Match? `http://www.nrmp.org/whythematch.pdf`.

22 P. A. Robards. Applying the two-sided matching processes to the United States Navy enlisted assignment process. Master's Thesis, Naval Postgraduate School, Monterey, Canada, 2001.

23 A. E. Roth and X. Xing. Turnaround time and bottlenecks in market clearing: Decentralized matching in the market for clinical psychologists. *Journal of Political Economy*, 105(2):284–329, 1997.

24 M. Soldner. Optimization and measurement in humanitarian operations: Addressing practical needs. PhD thesis, Georgia Institute of Technology, 2014.

25 A.C. Trapp, A. Teytelboym, A. Martinello, T. Andersson, and N. Ahani. Placement optimization in refugee resettlement. Working paper, 2018.

26 W. Yang, J. A. Giampapa, and K. Sycara. Two-sided matching for the US Navy detailing process with market complication. Technical Report CMU-R1-TR-03-49, Robotics Institute, Carnegie Mellon University, 2003.

A Appendix: Missing Proofs

Before we prove Lemma 10, it will be useful to prove the following simple observation.

▶ **Observation 20.** *For any critical node left unmatched in M, all its neighbors are in $A_0 \cup B_{\alpha+\beta}$.*

Proof. If $a \in C_A$ is unmatched in M then $a_{\alpha+\beta}$ has to be unmatched in M'. This is because for $0 \le i \le \alpha+\beta-1$, the node a_i is $d_{i+1}(a)$'s top choice neighbor, hence the stable matching M' has to match a_i. If a has a neighbor b in B_i for $i \le \alpha+\beta-1$ then the edge $(a_{\alpha+\beta}, b')$ blocks M', a contradiction to its stability in G'. Thus $b \in B_{\alpha+\beta}$.

Suppose $b \in C_B$ is unmatched in M and b has a neighbor a in A_i for $i \ge 1$. This means $(a_0, d_1(a))$ is in M'. Recall that $d_1(a)$ is a_0's least preferred neighbor. So the edge (a_0, b') blocks M', a contradiction to its stability in G'. Thus $a \in A_0$. ◁

Proof of Lemma 10. We will show there is no alternating path p with respect to M such that (i) p is an augmenting path wrt M and at least one endpoint of p is in C or (ii) p has even length with exactly one endpoint in C and this node is left unmatched in M. Then it follows from Lemma 6 that M is a critical matching in G.

We will first show there is no augmenting path p wrt M with an endpoint in C_B. It follows from the definition of sets A_i and B_i that $M \subseteq \cup_{i=0}^{\alpha+\beta}(A_i \times B_i)$. An important property here is that there is no edge in $A_i \times B_j$ where $i \ge j+2$. See the proof of Lemma 12, case 4 which shows that such an edge contradicts the stability of M' in G'.

The path p starts in B_0 at an unmatched node $b \in C_B$ and all of b's neighbors are in A_0 (by Observation 20). The matched partners of b's neighbors are in B_0. The node after this can be in A_1 and its partner is in B_1 and so on. So the shortest alternating path from an unmatched $b \in B_0$ to an unmatched $a \in A$ (such a node is in $A_\beta \cup A_{\alpha+\beta}$) moves across sets as follows: [here $(A_i - B_i)$ refers to a matching edge in $A_i \times B_i$]

$$B_0 - (A_0 - B_0) - (A_1 - B_1) - (A_2 - B_2) - \cdots - (A_{\beta-1} - B_{\beta-1}) - \cdots$$

Since all nodes in sets B_i for $0 \le i \le \beta - 1$ are in C_B, this implies there are at least $\beta + 1$ nodes of C_B in p. However $|C_B| = \beta$. So there is no such augmenting path p with respect to M.

The same argument shows that the shortest even length alternating path p with an unmatched node in C_B as one endpoint and any node in $B \setminus C_B$ (such a node is in $\cup_{i \ge \beta} B_i$) as another endpoint needs to have at least $\beta + 1$ nodes of C_B in it. However $|C_B| = \beta$. So there is no such alternating path p with respect to M.

We will now show there is no augmenting path p wrt M with an endpoint in C_A. An argument analogous to the one given above shows that the shortest alternating path from an unmatched $a \in A_{\alpha+\beta}$ to an unmatched node $b \in B$ (such a node is in $B_\beta \cup B_0$) moves across sets as follows: [here $(B_i - A_i)$ refers to a matching edge in $B_i \times A_i$]

$$A_{\alpha+\beta} - (B_{\alpha+\beta} - A_{\alpha+\beta}) - (B_{\alpha+\beta-1} - A_{\alpha+\beta-1}) - \cdots - (B_{\beta+1} - A_{\beta+1}) - \cdots$$

Since all nodes in levels A_i for $\beta + 1 \le i \le \alpha + \beta$ are in C_A, this implies there are at least $\alpha + 1$ nodes of C_A in p. However $|C_A| = \alpha$. So there is no such augmenting path p with respect to M.

The same argument shows that the shortest even length alternating path p with an unmatched node in C_A as one endpoint and any node in $A \setminus C_A$ (such a node is in $\cup_{i \le \beta} A_i$) as another endpoint needs to have at least $\alpha + 1$ nodes of C_A in it. However $|C_A| = \alpha$. So there is no such alternating path p with respect to M.

Thus there is no forbidden alternating path p (as given in Lemma 6) with respect to M. Hence M is a critical matching. ◀

Proof of Lemma 15. We will use Lemma 6 to show that M is a critical matching. We will show there is no alternating path p with respect to M such that: (i) p is an augmenting path wrt M and at least one endpoint of p is in C or (ii) p has even length with exactly one endpoint in C and this node is left unmatched in M.

We will first show there is no augmenting path p wrt M with an endpoint in C_B. Every unmatched node in C_B is in B_0 and its neighbors are in A_0 (analogous to Observation 20).

It follows from the definitions of A_i and B_i that $M \subseteq \cup_{i=0}^{\alpha+\beta+1}(A_i \times B_i)$. Moreover there is no edge in $A_i \times B_j$ where $i \ge j + 2$; otherwise the edge (a_{j+1}, b') would block M''.

Thus the path p starts in B_0 at an unmatched node $b \in C_B$ and the next node is in A_0. The matched partners of b's neighbors are in B_0. The node after this can be in A_1 and its partner is in B_1 and so on. So the shortest alternating path between an unmatched node $b \in B_0$ and an unmatched node $a \in A$ (such a node is in $A_{\beta+1} \cup A_{\alpha+\beta+1}$) moves across sets as follows (see Fig. 2):

$$B_0 - (A_0 - B_0) - (A_1 - B_1) - (A_2 - B_2) - \cdots - (A_{\beta-1} - B_{\beta-1}) - \cdots$$

Since all nodes in levels B_i for $0 \le i \le \beta - 1$ are in C_B, this implies there are at least $\beta + 1$ nodes of C_B in p. However $|C_B| = \beta$. So there is no such augmenting path p with respect to M.

The same argument shows that the shortest even length alternating path p with an unmatched node in C_B (such a node is in B_0) as one endpoint and any node in $B \setminus C_B$ (such a node is in $\cup_{i \ge \beta} B_i$) as another endpoint needs to have at least $\beta + 1$ nodes of C_B in it. However $|C_B| = \beta$. So there is no such alternating path p with respect to M.

We will now show there is no augmenting path p wrt M with an endpoint in C_A. An argument analogous to the one given above shows that the shortest alternating path from an unmatched $a \in C_A$ (note that $a \in A_{\alpha+\beta+1}$) to an unmatched node in B (such a node is in $B_\beta \cup B_0$) moves across sets as follows (see Fig. 2):

$$A_{\alpha+\beta+1} - (B_{\alpha+\beta+1} - A_{\alpha+\beta+1}) - (B_{\alpha+\beta} - A_{\alpha+\beta}) - (B_{\alpha+\beta-1} - A_{\alpha+\beta-1}) - \cdots - (B_{\beta+2} - A_{\beta+2}) - \cdots$$

Since all nodes in levels A_i for $\beta + 2 \le i \le \alpha + \beta + 1$ are in C_A, this implies there are at least $\alpha + 1$ nodes of C_A in p. However $|C_A| = \alpha$. So there is no such augmenting path p with respect to M.

The same argument shows that the shortest even length alternating path p with an unmatched node in C_A (such a node is in $A_{\alpha+\beta+1}$) as one endpoint and any node in $A \setminus C_A$ (such a node is in $\cup_{i \le \beta+1} A_i$) as another endpoint needs to have at least $\alpha + 1$ nodes of C_A in it. However $|C_A| = \alpha$. So there is no such alternating path p with respect to M.

Thus there is no forbidden alternating path p (as given in Lemma 6) with respect to M. Hence M is a critical matching in G. ◀

Fast and Exact Convex Hull Simplification

Georgiy Klimenko ✉
Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

Benjamin Raichel ✉
Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

Abstract

Given a point set P in the plane, we seek a subset $Q \subseteq P$, whose convex hull gives a smaller and thus simpler representation of the convex hull of P. Specifically, let $cost(Q, P)$ denote the Hausdorff distance between the convex hulls $\mathcal{CH}(Q)$ and $\mathcal{CH}(P)$. Then given a value $\varepsilon > 0$ we seek the smallest subset $Q \subseteq P$ such that $cost(Q, P) \leq \varepsilon$. We also consider the dual version, where given an integer k, we seek the subset $Q \subseteq P$ which minimizes $cost(Q, P)$, such that $|Q| \leq k$. For these problems, when P is in convex position, we respectively give an $O(n \log^2 n)$ time algorithm and an $O(n \log^3 n)$ time algorithm, where the latter running time holds with high probability. When there is no restriction on P, we show the problem can be reduced to APSP in an unweighted directed graph, yielding an $O(n^{2.5302})$ time algorithm when minimizing k and an $O(\min\{n^{2.5302}, kn^{2.376}\})$ time algorithm when minimizing ε, using prior results for APSP. Finally, we show our near linear algorithms for convex position give 2-approximations for the general case.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Convex hull, coreset, exact algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.26

Related Version *Full Version*: `https://arxiv.org/abs/2110.00671` [12]

Funding Partially supported by NSF CAREER Award 1750780.

Acknowledgements The authors thank Sariel Har-Peled for helpful discussions related to the paper.

1 Introduction

The convex hull of a set of points in the plane is one of the most well studied objects in computational geometry. As the number points on the convex hull can be linear, for example when the points are in convex position, it is natural to seek the best simplification using only k input points. To measure the quality of the subset we use one of the most common measures, namely the Hausdorff distance. Specifically, given a set P of n points in the plane, here we seek the subset of $Q \subseteq P$ of k points which minimizes the Hausdorff distance between $\mathcal{CH}(Q)$ and $\mathcal{CH}(P)$, where $\mathcal{CH}(X)$ denotes the convex hull of X. This is equivalent to finding the subset $Q \subseteq P$ of k points which minimizes $\varepsilon = \max_{p \in P} ||p - \mathcal{CH}(Q)||$. We refer to this as the *min-*ε problem. We also consider the dual *min-*k problem, where given a distance $\varepsilon \geq 0$, we seek the minimum cardinality subset $Q \subseteq P$ such that $\max_{p \in P} ||p - \mathcal{CH}(Q)|| \leq \varepsilon$. We emphasize that our goal is to find the optimal subset Q exactly. As discussed below, this is a far more demanding problem than allowing approximation in terms of k or ε.

A number of related problems have been considered before, though they all differ in key ways. The three main differences concern the error measure of Q, whether Q is restricted to be a subset from P, and whether a starting point is given. Varying any one of these aspects can significantly change the hardness of the problem.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 26; pp. 26:1–26:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Coresets. In this paper, we require our chosen points to be a subset of P, which from a representation perspective is desirable as the chosen representatives are actual input data points. Such subset problems have thus been extensively studied, and are referred to as coresets (see [9]). Given a point set P, a coreset is subset of P which approximately preserves some geometric property of the input. Thus here we seek a coreset for the Hausdorff distance.

Among coreset problems, ε-kernels for directional width are one of the most well studied. Define the directional width for a direction u as $w(u, P) = \max_{p\in P}\langle u, p\rangle - \min_{p\in P}\langle u, p\rangle$. Then $Q \subseteq P$ is an ε-kernel if for all u, $(1 - \varepsilon)w(u, P) \leq w(u, Q)$. It is known that for any point set $P \subset \mathbb{R}^d$ there is an ε-kernel of size $O(1/\varepsilon^{(d-1)/2})$ [1]. For worst case point sets $\Omega(1/\varepsilon^{(d-1)/2})$ size is necessary, however, for certain inputs, significantly smaller coresets may be possible. (As an extreme example, if the points lie on a line, then the $k = 2$ extreme points achieves $\varepsilon = 0$ error.) Thus [5] considered computing coresets whose size is measured relative to the optimum for a given input point set. Specifically, if there exists an ε-coreset for Hausdorff distance with k points, then in polynomial time they give an ε-coreset with $O(dk \log k)$ size, or alternatively an $(8\varepsilon^{1/3} + \varepsilon)$-coreset with $O(k/\varepsilon^{2/3})$ size. Note that the standard strategy to compute ε-kernels applies a linear transformation to make the point set fat, and then roughly speaking approximates the Hausdorff problem. Thus ε-coresets for Hausdorff distance yield $O(\varepsilon)$-kernels (where the constant relates to the John ellipsoid theorem). However, ε-kernels do not directly give such coresets for Hausdorff distance, as it depends on the fatness of the point set, i.e. Hausdorff is arguably the harder problem.

Most prior work on coresets gave approximate solutions. However, our focus is on exact solutions. Along these lines, a very recent PODS paper [18] considered what they called the *minimum ε-corset* problem, where the goal is to exactly find the minimum sized ε-coreset for a new error measure they introduced. Specifically, $Q \subseteq P$ is an ε-coreset for maxima representation if for all directions u, $(1-\varepsilon)\omega(u, P) \leq \omega(u, Q)$, where $\omega(u, X) = \max_{x\in X}\langle u, x\rangle$. While related to our Hausdorff measure, again like directional width, it differs in subtle ways. For example, observe their measure is not translation invariant. Moreover, they assume the input is α-fat for some constant α, while we do not. For their measure they give a cubic time algorithm in the plane, whereas our focus is on significantly subcubic time algorithms.

In the current paper, we select Q so as to minimize the maximum distance of a point in P to $\mathcal{CH}(Q)$. [13] instead considered the problem of selecting Q so as to minimize the sum of the distances of points in P to $\mathcal{CH}(Q)$. They provided near cubic (or higher) running time algorithms for certain generalized versions of this summed coreset variant.

Other related problems. If one relaxes the problem to no longer require Q to be a subset of P, then related problems have been studied before. Given two convex polygons X and Y, where X lies inside Y, [3] provided a near linear time algorithm for the problem of finding the convex polygon Z with the fewest number of vertices such that $X \subseteq Z \subseteq Y$. The problem of finding the best approximation under Hausdorff distance has also been considered before. Specifically, if Q can be any subset from $\mathcal{CH}(P)$ (i.e. it is not a coreset), then [14] gave a near linear time algorithm for approximating the convex hull under Hausdorff distance, but under the key assumption that they are given a starting vertex which must be in Q. We emphasize that assuming a starting point is given makes a significant difference, and intuitively relates to the difference in hardness between single source shortest paths and all pairs shortest paths.

A number of papers have considered simplifying polygonal chains. Computing the best global Hausdorff simplification is NP-hard [17, 15]. Most prior work instead considered local simplification, where points from the original chain are assigned to the edge of the simplification whose end points they lie between. In general such algorithms take at least quadratic time, with subquadratic algorithms known for certain special or approximate cases. For example, [2] gave an $O(n^{4/3+\delta})$ time algorithm, for any $\delta > 0$, under the L_1 metric. Our problem relates to these works in that we must approximate the chain representing the convex hull. On the one hand, convexity gives us additional structure. However, unlike polygonal chain simplification, we do not have a well defined starting point (i.e. the convex hull is a closed chain), which as remarked above makes a significant difference in hardness.

Our problem also relates to polygon approximation, for which prior work often instead considered approximation in relation to area. For example, given a convex polygon P, [16] gave a near linear time algorithm for finding the three vertices of P whose triangle has the maximum area. To illustrate one the many ways that area approximations differ, observe that the area of the triangle of the three given points of P can be determined in constant time, whereas the computing the furthest point from P to the triangle takes linear time.

Our results. We give fast and exact algorithms for both the min-k and min-ε problems for summarizing the convex hull in the plane. While a number of related problems have been considered before as discussed above, to the best of our knowledge we are the first to consider exact algorithms for this specific version of the problem.

Our main results show that when the input set P is in convex position then the min-k problem can be solved exactly in $O(n \log^2 n)$ deterministic time, and the min-ε problem can be solved exactly in $O(n \log^3 n)$ time with high probability. Note that this version of the problem is equivalent to allowing the points in P to be in arbitrary position, but requiring that the chosen subset Q consist of vertices of the convex hull. (Which follows as the furthest point to $\mathcal{CH}(Q)$ is always a vertex of $\mathcal{CH}(P)$.) Thus this restriction is quite natural, as we are then using vertices of the convex hull to approximate the convex hull, i.e. furthering the coreset motivation.

For the general case when P is arbitrary and Q is any subset of P, we show that in near quadratic time these problems can be reduced to computing all pairs shortest paths in an unweighted directed graph. This yields an $O(n^{2.5302})$ time algorithm for the min-k problem and an $O(\min\{n^{2.5302}, kn^{2.376}\})$ time algorithm for the min-ε problem, by utilizing previous results for APSP in unweighted directed graphs. Moreover, while exact algorithms are our focus, we show that our near linear time algorithms for points in convex position immediately yield 2-approximations for the general case with the same near linear running times. Also, appropriately using single source shortest paths rather than APSP in our graph based algorithms, gives $O(n^2 \log n)$ time solutions which use at most one additional point.

2 Preliminaries

Given a point set X in $\mathbb{R}^2$, let $\mathcal{CH}(X)$ denote its convex hull. For two points $x, y \in \mathbb{R}^2$, let $\overline{xy}$ denote their line segment, that is $\overline{xy} = \mathcal{CH}(\{x, y\})$. Throughout, given points $x, y \in \mathbb{R}^2$, $||x - y||$ denotes their Euclidean distance. Given two compact sets $X, Y \subset \mathbb{R}^2$, $||X - Y|| = \min_{x \in X, y \in Y} ||x - y||$ denotes their distance. For a single point x we write $||x - Y|| = ||\{x\} - Y||$.

For any two finite point set $Q, P \subset \mathbb{R}^2$ we define

$$cost(Q, P) = \max_{p \in P} ||p - \mathcal{CH}(Q)||$$

Note that for $Q \subseteq P$, we have that $\mathcal{CH}(Q) \subseteq \mathcal{CH}(P)$, and moreover the furthest point in $\mathcal{CH}(P)$ from $\mathcal{CH}(Q)$ is always a point in P. Thus the $cost(Q, P)$ is equivalent to the Hausdorff distance between $\mathcal{CH}(Q)$ and $\mathcal{CH}(P)$.

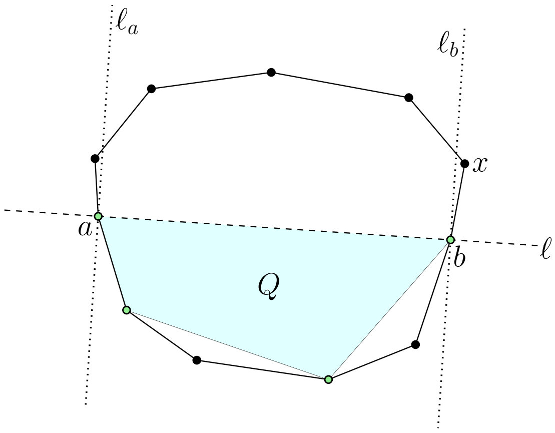


Figure 3.1 An example of the defined objects from Lemma 5.

In this paper we consider the following two related problems, where for simplicity, we assume that P is in general position.

▶ **Problem 1** (min-k). *Given a set $P \subset \mathbb{R}^2$ of n points, and a value $\varepsilon > 0$, find the smallest integer k such that there exists a subset $Q \subseteq P$ where $|Q| \leq k$ and cost$(Q, P) \leq \varepsilon$.*

▶ **Problem 2** (min-ε). *Given a set $P \subset \mathbb{R}^2$ of n points, and an integer k, find the smallest value ε such that there exists subset $Q \subseteq P$ where $|Q| \leq k$ and cost$(Q, P) \leq \varepsilon$.*

For simplicity the above problems are phrased in terms of finding the value of either k or ε, though we remark that our algorithms for these problems also immediately imply the set Q realizing the value can be determined in the same time. Thus in the following when we refer to a solution to these problems, we interchangeably mean either the value or the set realizing the value.

In the following section we restrict the point set P to lie in convex position, thus for simplicity we define the following convex versions of the above problems.

▶ **Problem 3** (cx-min-k). *Given a set $P \subset \mathbb{R}^2$ of n points in convex position, and a value $\varepsilon > 0$, find the smallest integer k such that there exists a subset $Q \subseteq P$ where $|Q| \leq k$ and cost$(Q, P) \leq \varepsilon$.*

▶ **Problem 4** (cx-min-ε). *Given a set $P \subset \mathbb{R}^2$ of n points in convex position, and an integer k, find the smallest value ε such that there exists subset $Q \subseteq P$ where $|Q| \leq k$ and cost$(Q, P) \leq \varepsilon$.*

3 Convex Position

In this section we give near linear time algorithms for the case when P is in convex position, that is for Problem 3 and Problem 4. First, we need several structural lemmas and definitions.

3.1 Structural Properties and Definitions

▶ **Lemma 5.** *Let P be a set of n points in convex position. Consider any subset $Q \subset P$, and let a, b be consecutive in the clockwise ordering of Q. Then for any point $x \in P$ which falls between a and b in the clockwise ordering of P, we have $||x - \mathcal{CH}(Q)|| = ||x - \overline{ab}||$.*

Proof. Let x be any point between a and b in the clockwise ordering of P, and let l denote the line through a and b. Since a and b are consecutive in the clockwise order of Q, $\mathcal{CH}(Q)$ lies entirely in the closed halfspace defined by ℓ and on the opposite side of ℓ as x. So if z denotes the closest point to x in $\mathcal{CH}(Q)$, then the segment $\overline{xz}$ must intersect ℓ.

Consider the lines l_a and l_b which are perpendicular to l and go through a and b respectively. If x lies between l_a and l_b, then its projection onto ℓ lies on the segment $\overline{ab}$, and hence this is in fact its projection onto $\mathcal{CH}(Q)$, and the claim holds. Otherwise, suppose that x and a are in opposite halfplanes defined by the line l_b, see Figure 3.1. (A similar argument will hold when x and b are in opposite halfplanes defined by the line l_a.) Observe, that the closest point in $\ell_b \cap \mathcal{CH}(Q)$ to x is the point b, since x is in the opposite halfspace defined by ℓ as $\mathcal{CH}(Q)$, and ℓ_b is orthogonal to ℓ. Thus if the shortest path to z intersects ℓ_b, then it would imply $z = b$, and so again the claim holds. So suppose z and x are on the same side of ℓ_b. Since $\overline{xz}$ intersects ℓ, z must lie on the opposite side of ℓ as x. Since $z \in \mathcal{CH}(Q)$, this implies there is a point $y \in Q$ which like z is on the same side of ℓ_b as x but on the opposite side of ℓ as x (since there is no point of Q on the same side of ℓ as x). Thus similarly, the segment $\overline{xy}$ intersects ℓ, and let y' denote this intersection point. Since x and y are on the same side of ℓ_b, which is opposite the side of a, this implies b lies on the segment $\overline{ay'}$. As y' lies on the segment $\overline{xy}$, this in turn implies that b lies in the triangle $\Delta(ayx)$. This is a contradiction, since $a, y, x, b \in P$, and so b lying in $\Delta(azx)$ implies P is not in convex position. ◀

Assume that the points in $P = \{p_1, \dots, p_n\}$ are indexed in clockwise order. We now wish to prove a lemma about the optimal cost solution when restricted to points between some index pair i, j. As we wish our definition to work regardless of whether $i \le j$ or $j \le i$, we define the following notation. For a triple of indices i, x, j, we write $i \preceq x \preceq j$ to denote that p_x falls between p_i and p_j in the clockwise ordering. More precisely, if $i \le j$ then this means $i \le x \le j$, and if $j \le i$ then this means that $j \le x \le n$ or $1 \le x \le i$.

▶ **Definition 6.** *For any integer $0 \le k \le n-2$ we define*

$$cost_k(i,j) = \min_{i \preceq l_1 \preceq \ldots \preceq l_k \preceq j} \max_{i \preceq v \preceq j} ||p_v - \mathcal{CH}(p_i, p_{l_1}, \dots, p_{l_k}, p_j)||.$$

That is, $cost_k(i,j)$ is the minimum cost solution when restricted to including p_i, p_j, and k other vertices in clockwise order between p_i and p_j, and where we only evaluate the cost with respect to points in clockwise order between p_i and p_j.

According to the above definition, we have that $cost_0(i,j) = \max_{i \preceq v \preceq j} ||p_v - \mathcal{CH}(p_i, p_j)|| = \max_{i \preceq v \preceq j} ||p_v - \overline{p_i p_j}||$. Observe that the following is implied by Lemma 5.

▶ **Corollary 7.** *Let $Q = \{p_{l_1}, \dots, p_{l_k}\} \subseteq P$ be indexed in clockwise order, and let $l_{k+1} = l_1$. Then,*

$$cost(Q,P) = \max_{p \in P} ||p - \mathcal{CH}(Q)|| = \max_{1 \le i \le k} \max_{l_i \preceq j \preceq l_{i+1}} ||p_j - \overline{p_{l_i} p_{l_{i+1}}}|| = \max_{1 \le i \le k} cost_0(l_i, l_{i+1}).$$

For more general values of k, the following lemma will be used to argue we can use a greedy algorithm.

▶ **Lemma 8.** *For any indices $i' \preceq i \preceq j \preceq j'$, it holds that $cost_k(i,j) \le cost_k(i',j')$.*

Proof. Let $p_{i'}, p_{l_1}, \dots p_{l_k}, p_{j'}$ be the clockwise chain of vertices that realizes $cost_k(i',j')$. That is, $cost_k(i',j') = max_{i' \preceq v \preceq j'} ||p_v - \mathcal{CH}(p_{i'}, p_{l_1}, \dots p_{l_k}, p_{j'})||$. Observe that if we add points to this chain then we can only decrease the cost. Specifically, we consider adding the points p_i and p_j. So let $p_{l_x}, \dots, p_{l_y}$ be the subchain of $p_{l_1}, \dots p_{l_k}$ consisting of all $i \preceq l_i \preceq j$. Then we have,

$$\begin{aligned} cost_k(i',j') &= max_{i' \preceq v \preceq j'} ||p_v - \mathcal{CH}(p_{i'}, p_{l_1}, \dots, p_{l_k}, p_{j'})|| \\ &\geq max_{i' \preceq v \preceq j'} ||p_v - \mathcal{CH}(p_{i'}, p_{l_1}, \dots, p_i, p_x, \dots, p_y, p_j, \dots, p_{l_k}, p_{j'})|| \\ &\geq max_{i \preceq v \preceq j} ||p_v - \mathcal{CH}(p_{i'}, p_{l_1}, \dots, p_i, p_x, \dots, p_y, p_j, \dots, p_{l_k}, p_{j'})|| \\ &\geq max_{i \preceq v \preceq j} ||p_v - \mathcal{CH}(p_i, p_x, \dots, p_y, p_j)|| \geq cost_k(i,j). \end{aligned}$$

The second to last inequality holds by Lemma 5. The last inequality holds as the chain $p_x, \dots, p_y$ has at most k points (since it was a subchain of $p_{l_1}, \dots, p_{l_k}$) and $cost_k(i,j)$ is defined by the minimum cost such chain between i and j. ◀

We now define the notions of friends and greedy sequences, which we use in the next section to design our greedy algorithm.

▶ **Definition 9.** *For an index i and value $\varepsilon \geq 0$, define the ε-friend of i, denoted $f_\varepsilon(i)$, as the index j of the vertex furthest from p_i in the clockwise ordering of P, such that $cost_0(i,j) \leq \varepsilon$.*

Note that $f_\varepsilon(i)$ is always well defined. In particular, $cost_0(i, i+1) = 0$ for any i. Moreover, if the ball of radius ε centered at p_i contains P then $f_\varepsilon(i) = i$, and the point p_i by itself is an optimal solution to Problem 3. Note that we can easily determine if such a point exists in $O(n \log n)$ time by computing the farthest Voronoi diagram of P,[1] and then querying all points in P. For simplicity we will assume $f_\varepsilon(i) \neq i$, which can thus be assured by such a preprocessing step.

▶ **Definition 10.** *Let $Q = \{p_{l_1}, p_{l_2}, \dots, p_{l_k}\}$ be any subset of P, which we assume has been indexed such that $l_1 < l_2 < \dots < l_k$. We call Q a* greedy sequence *if for all $1 \leq i < k$, we have $f_\varepsilon(l_i) = l_{i+1}$, and $f_\varepsilon(l_k) < l_k$. We call a greedy sequence* valid *if $f_\varepsilon(l_k) \geq l_1$.*

Note that in the above definition, the condition that $f_\varepsilon(l_k) < l_k$ ensures that the ε-friend of p_{l_k} goes past the vertex p_n, i.e. this ensure that the sequence is a maximal sequence without wrapping around. Note also there always exists a valid greedy sequence. Specifically, we trivially have that for any greedy sequence $f_\varepsilon(l_k) \geq 1$. Thus the greedy sequence starting at p_1 is valid as in that case $l_1 = 1$.

▶ **Observation 11.** *Let $Q = \{p_{l_1}, p_{l_2}, \dots, p_{l_k}\}$ be a valid greedy sequence. Then since Q is a greedy sequence $cost_0(l_i, l_{i+1}) \leq \varepsilon$ for all $1 \leq i < k$. Furthermore, $cost_0(l_k, l_1) \leq cost_0(l_k, f_\varepsilon(l_k)) \leq \varepsilon$ by Lemma 8 and since Q is valid. Thus by Corollary 7, $cost(Q,P) \leq \varepsilon$.*

▶ **Lemma 12.** *Let P, ε be an instance Problem 3. Any valid greedy sequence of minimum possible cardinality is an optimal solution to the given instance.*

Proof. Let $Q = \{p_{l_1}, p_{l_2}, \dots, p_{l_k}\}$ be an optimal solution to Problem 3, indexed such that $1 \leq l_1 < l_2 < \dots < l_k$. Thus $cost(Q,P) \leq \varepsilon$ and so by Corollary 7, $\max_{1 \leq i \leq k} cost_0(l_i, l_{i+1}) \leq \varepsilon$, where $l_{k+1} = l_1$. Thus if Q is a greedy sequence then it is a valid greedy sequence, and the claim holds. So suppose Q is not a greedy sequence. Now we show that Q can be converted to a valid greedy sequence with the same cardinality.

Let $j > 1$ be the first index such that $l_j \neq f_\varepsilon(l_{j-1})$. Let $w_j = f_\varepsilon(l_{j-1})$ and let $\{w_{j+1}, w_{j+2}, \dots, w_k\}$ be the indices which realize $cost_{k-j}(w_j, l_1)$ according to Definition 6. Then we modify Q by replacing the suffix $\{p_{l_j}, p_{l_{j+1}}, \dots, p_{l_k}\}$ with $\{p_{w_j}, p_{w_{j+1}}, \dots, p_{w_k}\}$.

[1] The farthest Voronoi diagram of P partitions the plane into regions sharing the same farthest point in P. It allows one to find the farthest point in P from a query in logarithmic time. See for example [10].

Notice that the cost of Q after this modification is still $\leq \varepsilon$ because $cost_0(l_{j-1}, w_j) \leq \varepsilon$ as $w_j = f_\varepsilon(l_{j-1})$, and by Lemma 8 we have $cost_{k-j}(w_j, l_1) \leq cost_{k-j}(l_j, l_1)$. Now repeat this procedure until $h = f_\varepsilon(l_{j-1})$ goes beyond index n. Let the resulting new optimal solution be denoted Q'. If $h \geq l_1$, then Q' is a valid greedy sequence by our construction, and we are done. So if the sequence failed to be a valid greedy sequence, then $1 \leq h < l_1$. Thus we can repeat the whole procedure, relabeling vertices of Q' such that $l_1 = h$. This means that each time we repeat this procedure we either produce a valid greedy sequence or we decrease l_1. At some point $l_1 = 1$, at which time the procedure must produce a valid greedy sequence as in this case $h \geq 1 = l_1$.

The above argues that some valid greedy sequence of minimum cardinality is optimal. Note this implies all valid greedy sequences of minimum cardinality are optimal, since they all have the same size, and by Observation 11 their cost is $\leq \varepsilon$. ◀

3.2 The min-k Algorithm

In this section we give an efficient algorithm for Problem 3. The idea is to use the $f_\varepsilon(i)$ values to define a graph. Specifically, the *friend graph* G_f is the directed graph with vertex set P where there is an edge from p_i to p_j if and only if $f_\varepsilon(i) = j$ and $i < j$. Thus every vertex in G_f has outdegree at most 1. Moreover, G_f is acyclic since we only created edges from lower index vertices to higher index ones. These two properties together imply that G_f is a forest, where each sink vertex defines the root of a tree. Thus every vertex in G_f has a well defined depth, where sink vertices have depth one.

Let $Q = \{p_{l_1}, p_{l_2}, \ldots, p_{l_k}\}$ be a greedy sequence, as defined in Definition 10. Then observe that for all $1 \leq i < k$, $p_{l_i} p_{l_{i+1}}$ is an edge of G_f, and hence Q corresponds to a path in G_f. Moreover, the condition that $f_\varepsilon(l_k) < l_k$ in Definition 10 implies that p_{l_k} is a sink vertex in G_f, and hence Q corresponds to a path in G_f from the vertex p_{l_1} to the root of its corresponding tree. Conversely, for the same reasons if we are given a path $p_{l_1}, p_{l_2}, \ldots, p_{l_k}$ in G_f where p_{l_k} is a sink, then this path is a greedy sequence. That is, the set of paths ending in sinks in G_f and the set of greedy sequences are in one-to-one correspondence.

Thus given all the $f_\varepsilon(i)$ values have been precomputed, this suggests a simple linear time algorithm to compute a valid greedy sequence Q with the fewest number of points, which by Lemma 12 is an optimal solution to the given instance of Problem 3. Specifically, find all pairs (p_i, p_r) where $p_i \in P$ and p_r is the root of the tree in G_f which contains p_i. By the above discussion, each such pair (p_i, p_r) corresponds to a greedy sequence, and all greedy sequences are represented by some pair. We now restrict to pairs that are valid according to Definition 10, that is pairs where $f_\varepsilon(r) \geq i$. For each such pair, the length of the corresponding sequence is simply the depth of p_i in the tree rooted at p_r. Thus we return as our solution the depth of p_i from the valid pair (p_i, p_r) where p_i has minimum depth.

All the (p_i, p_r) pairs and the depths can be determined in $O(n)$ time by topologically sorting since G_f is a forest. Determining the valid pairs, and the minimum depth valid pair can then be done with a simple linear scan. We thus have the following.

▶ **Lemma 13.** *Assume that $f_\varepsilon(i)$ for all $1 \leq i \leq n$ has been precomputed. Then Problem 3 can be solved in $O(n)$ time.*

The question now then is how quickly can we compute all of the $f_\varepsilon(i)$ values. To that end, we first argue that with some precomputation the $cost_0(i, j)$ values can be queried efficiently. To do so, we make use a result from [7] which builds a datastructure for a geometric query they call Farthest Vertex in a Halfplane, which we rephrase below using our notation.

▶ **Lemma 14** ([7]). *Let $P \subset \mathbb{R}^2$ be a point set in convex position. P can be preprocessed in $O(n \log n)$ time such that given a query (q, l_q), where q is a point and l_q is a directed line through q, in $O(\log^2 n)$ time one can return the farthest point from q among the points in P to the left of l_q.*

▶ **Lemma 15.** *Let $P = \{p_1, \dots, p_n\} \subset \mathbb{R}^2$ be a point set in convex position, labeled in clockwise order. With $O(n \log n)$ precomputation time, for any query index pair (i, j), $cost_0(i, j)$ can be computed in $O(\log^2 n)$ time.*

Proof. Let $\ell = \ell(p_i, p_j)$ be the line through p_i and p_j, which we view as being oriented in the direction from p_i towards p_j. Also, let r_i and r_j denote the rays originating at p_i and p_j respectively, pointing in the direction orthogonal to ℓ and on the left side side of ℓ. Finally, let $P_{i,j} = \{p_k \in P \mid i \prec k \prec j\}$, and thus $cost_0(i, j) = \max_{x \in P_{i,j}} ||x - \overline{p_i, p_j}||$.

Observe that the projection of any point $x \in P_{i,j}$ onto ℓ either lies on the portion of ℓ before p_i, on the line segment $\overline{p_i p_j}$, or on the portion of ℓ after p_j. Thus we have a natural partition of $P_{i,j}$ into three sets, the subset in the right angle cone C_i bounded by ℓ and r_i, those in the slab $Slab(i, j)$ bounded by ℓ, r_i, and r_j, and those in the right angle cone C_j bounded by ℓ and r_j. Observe that for any point x in C_i or C_j, its closest point on $\overline{p_i p_j}$ is p_i or p_j, respectively, and moreover $||x - \ell|| \le ||x - \overline{p_i p_j}||$. Thus we have that,

$$\begin{aligned} cost_0(i, j) &= \max_{x \in P_{i,j}} ||x - \overline{p_i, p_j}|| \\ &= \max\{\max_{x \in C_i \cap P_{i,j}} ||x - p_i||, \max_{x \in C_j \cap P_{i,j}} ||x - p_j||, \max_{x \in Slab(i,j) \cap P_{i,j}} ||x - \overline{p_i p_j}||\} \\ &= \max\{\max_{x \in C_i \cap P_{i,j}} ||x - p_i||, \max_{x \in C_j \cap P_{i,j}} ||x - p_j||, \max_{x \in P_{i,j}} ||x - \ell||\}. \end{aligned}$$

Therefore, it suffices to describe how to compute each of the three terms in the stated time. Computing $\max_{x \in P_{i,j}} ||x - \ell||$ is straightforward as the points in $P_{i,j}$ are in convex position and in particular if we consider them in their clockwise order, then their distance to ℓ is a concave function. So assume that P is given in an array sorted in clockwise order. (If not, we can compute such an array with $O(n \log n)$ preprocessing time by computing the convex hull.) Then given a query pair (i, j), in $O(\log n)$ time we can binary search over $P_{i,j}$ to find $\max_{x \in P_{i,j}} ||x - \ell||$, since $P_{i,j}$ is a subarray of P. (Note if $j < i$ then technically $P_{i,j}$ is two subarrays.)

Now consider the subset in the right angle cone C_i (a similar argument will hold for C_j). Let C'_i be the cone C_i but reflected over the line ℓ. Suppose that both C_i and C'_i contained points from P, call them p and p', respectively. Then observe that the triangle $\Delta(p, p', p_j)$ would contain the point p_i, which is a contradiction as P was in convex position. Thus either $C_i \cap P = \emptyset$ or $C'_i \cap P = \emptyset$. So let L be the line orthogonal to ℓ, passing through p_i, and oriented so that C_i and C'_i lie to the left (i.e. L is the line supporting the ray r_i from above). By Lemma 14, we can preprocess P in $O(n \log n)$ time, such that in $O(\log^2 n)$ time we can compute the point in P furthest from p_i and to the left of L. If the returned point lies in C'_i then we know $C_i \cap P = \emptyset$ and so $\max_{x \in C_i \cap P_{i,j}} ||x - p_i|| = 0$. If the returned point lies in C_i then it realizes $\max_{x \in C_i \cap P_{i,j}} ||x - p_i||$. ◀

▶ **Theorem 16.** *Problem 3 can be solved in $O(n \log^2 n)$ time.*

Proof. By Lemma 13, given the f_ε values have been computed, Problem 3 can be solved in $O(n)$ time. Thus to prove the theorem it suffices to compute $f_\varepsilon(i)$ for all i in $O(n \log^2 n)$ time. Recall that $f_\varepsilon(i)$ is the index z of the vertex furthest from p_i in the clockwise ordering of P, such that $cost_0(i, z) \le \varepsilon$. First observe that as we increase i, $f_\varepsilon(i)$ moves clockwise.

More precisely, by Lemma 8, $\varepsilon \geq cost_0(i, f_\varepsilon(i)) \geq cost_0(i+1, f_\varepsilon(i)) \geq cost_0(i+1, j)$, for any $i+1 \leq j \leq f_\varepsilon(i)$, and thus $i \preceq f_\varepsilon(i) \preceq f_\varepsilon(i+1)$. Moreover, again by Lemma 8, the indices j such that $cost_0(i,j) \leq \varepsilon$ are consecutive in the clockwise ordering of P.

This suggests a simple strategy to compute the $f_\varepsilon(i)$ values. Namely, to find $f_\varepsilon(1)$, we compute all values $cost(1,j)$, starting with $j = 3$ and increasing j until we find a value j' such that $cost_0(1,j') > \varepsilon$. This implies $f_\varepsilon(1) = j' - 1$, since as mentioned above the values such that $cost_0(1,j) \leq \varepsilon$ are consecutive. More generally, to compute $f_\varepsilon(i+1)$, we compute all values $cost_0(i+1,j)$, starting with $j = f_\varepsilon(i) + 1$ and increasing j until we find a value j' such that $cost_0(i+1,j') > \varepsilon$, which again by the above implies $f_\varepsilon(i+1) = j' - 1$.

The total time is clearly bounded by the time it takes to compute all the queried $cost_0$ values. Observe that when the algorithm queries a value $cost_0(i,j)$ then the previous $cost_0$ query was either to $cost_0(i-1,j)$ or $cost_0(i,j-1)$, implying that in total we compute $O(n)$ $cost_0$ values. By Lemma 15, with $O(n \log n)$ precomputation, any $cost_0$ value can be computed in $O(\log^2 n)$ time. Thus the total time is $O(n \log^2 n)$. ◀

3.3 The min-ε Algorithm

In this section we design an efficient algorithm for Problem 4, where k is given and our goal is to minimize ε. To do so, we will use our algorithm from the previous section for Problem 3, where ε was fixed and we were minimizing k. Specifically, throughout this section, given an instance P, k of Problem 4, we use $Decider(\varepsilon)$ to denote the procedure which runs the algorithm of Theorem 16 on the instance P, ε of Problem 3 and returns True if the solution found uses $\leq k$ points, and returns False otherwise.

Let $\mathcal{E} = \{cost_0(i,j) \mid 1 \leq i,j \leq n\}$. We call $\mathcal{E}$ the set of *critical values*, where observe that by Corollary 7, the optimal solution to the given instance of Problem 4 is a critical value in the set $\mathcal{E}$. Thus a natural approach would be to explicitly compute, sort, and then binary search over $\mathcal{E}$ using $Decider(\varepsilon)$. However, such an approach would require at least quadratic time as $|\mathcal{E}| = \Theta(n^2)$. We now argue that by using random sampling we can achieve near linear running time with high probability. Similar sampling strategies have been used before, and in particular we employ a strategy which was used in [11] for computing the Frechet distance. We first observe that one can efficiently sample values from $\mathcal{E}$.

▶ **Lemma 17.** *With $O(n \log n)$ precomputation time, one can sample a value uniformly at random from $\mathcal{E}$ in $O(\log^2 n)$ time.*

Proof. To sample a pair from $1 \leq i,j \leq n$ uniformly at random, we first sample an integer uniformly at random from $[1,n]$ for i, and then sample an integer uniformly at random from $[1, n-1]$ for j (where j is indexed from the set with i removed). This takes $O(1)$ time given the standard assumption that sampling a random integer in a given range takes $O(1)$ time. (Even if it took $O(\log n)$ time it would not affect the overall time.) Now to sample a value uniformly at random from $\mathcal{E}$ we just need to compute $cost_0(i,j)$. From Lemma 15 this can be done in $O(\log^2 n)$ time with $O(n \log n)$ precomputation time. ◀

Before presenting our algorithm, we require the following subroutine.

▶ **Lemma 18.** *Given an interval $[\alpha, \beta]$, then the set $X = [\alpha, \beta] \cap \mathcal{E}$ can be computed in $O((n \log n + |X|) \log^2 n)$ time. Let $Extract(\alpha, \beta)$ denote this procedure.*

Proof. Fix an index i. By Lemma 8 we know that $cost_0(i,j)$ increases monotonically as we move p_j clockwise. Thus $S_i = \{j \mid cost_0(i,j) \in [\alpha, \beta]\}$ is a contiguous set of indices, and moreover, we can binary search for the smallest index in this set (i.e. the first index j in clockwise order from i such that $cost_0(i,j) \geq \alpha$). After finding this smallest such index, to output the rest of S_i we just simply increment j until $cost_0(i,j) > \beta$. Note that $X = \cup_i S_i$, and thus to find X we then repeat this procedure for all i.

Note that in each step of the algorithm we compute a $cost_0$ value, and thus the total time is bounded by the time is takes to compute all the queried $cost_0$ values. For all n values of i we perform a binary search, thus requiring $O(n \log n)$ $cost_0$ queries for all binary searches. For a given i, after the binary searching, we then perform $|S_i|$ $cost_0$ queries to determine the rest of the set S_i, and thus over all i we perform $|X| = \sum_i |S_i|$ queries. By Lemma 15 each $cost_0$ query takes $O(\log^2 n)$ time, with $O(n \log n)$ preprocessing, and so the total time is thus $O((n \log n + |X|) \log^2 n)$. ◀

We remark that it should be possible to improve the running time of the above algorithm to $O((n + |X|) \log^2 n)$ using the same approach as in the proof of Theorem 16. However, ultimately this will not change the asymptotic running time of our overall algorithm.

Algorithm 1 Algorithm for solving Problem 4.

```
  Input  : An instance $P, k$ of Problem 4.
  Output : The value $\varepsilon$ of the optimal solution.
1 Perform the precomputation step from Lemma 15.
2 Sample a set $S$ of $4n$ values from $\mathcal{E}$.
3 Sort $S$ and binary search using $Decider$. Let $[\alpha, \beta]$ be the resulting interval found
   where $Decider(\alpha) = False$ and $Decider(\beta) = True$.
4 Let $X = Extract(\alpha, \beta)$.
5 Sort $X$ and binary search using $Decider$.
6 Return the smallest value $\varepsilon \in X$ such that $Decider$ was $True$.
```

Our algorithm for solving Problem 4 is shown in Algorithm 1. The correctness of this algorithm is straightforward. By the discussion above the optimal value ε is in $\mathcal{E}$, and the correctness of *Decider* follows from the previous section. Thus when we binary search over S using *Decider*, we know that $\varepsilon \in [\alpha, \beta]$. Thus, by Lemma 18, we know that $X = Extract(\alpha, \beta)$ contains ε. Thus our final binary search over X using *Decider* is guaranteed to find ε.

The more challenging question is what is the running time of Algorithm 1, for which we have the following helper lemma.

▶ **Lemma 19.** *Let $X = Extract(\alpha, \beta)$ be the set computed on line 4 in Algorithm 1. Then for any $c \geq 1$, we have that $Pr[|X| > cn \ln n] < 1/n^c$.*

Proof. Let ε be the optimal value to the given instance of Problem 4. We first argue that with high probability there are at most $(c/2)n \ln n$ values from $\mathcal{E}$ that are contained in $[\alpha, \beta]$ (i.e. in the set X) that are also larger than ε. Let Z be the $(c/2)n \ln n$ values in $\mathcal{E}$ closest to ε but also greater than ε. (Note that if there are less than $(c/2)n \ln n$ values greater than ε, then the claim trivially holds.) Observe that if our random sample S on line 2 contains even a single value from Z then the claim holds as this value then upper bounds β, and so there are at most $|Z| = (c/2)n \ln n$ values from $\mathcal{E}$ in $(\varepsilon, \beta]$. The probability that the $4n$ sized random sample of values from $\mathcal{E}$ does not contain any element from Z is at most

$$(1 - |Z|/|\mathcal{E}|)^{4n} \leq (1 - ((c/2)n \ln n)/n^2)^{4n} = (1 - (c \ln n)/2n)^{4n} \leq e^{-2c \ln n} = 1/n^{2c} < 1/2n^c,$$

where we used the standard inequality $1 + x \leq e^x$ for any value x. Note that a symmetric argument yields the same probability bound for the event that there are more than $(c/2)n \log n$ values from $\mathcal{E}$ contained in $[\alpha, \beta]$ that are smaller than ε. Thus by the union bound, the probability that $|X|$ has more than $cn \ln n$ values is less than $1/n^c$. ◀

▶ **Theorem 20.** *Algorithm 1 solves Problem 4 in $O(cn\log^3 n)$ time with probability $\geq 1-1/n^c$, for any $c \geq 1$.*

Proof. The straightforward correctness of the algorithm has already been discussed above. As for the running time, the precomputation on line 1 takes $O(n\log n)$ time by Lemma 15. By Lemma 17, it then takes $O(n\log^2 n)$ time to sample the $4n$ values in the set S. Sorting S takes $O(n\log n)$ time, and binary searching using *Decider* takes $O((\log n)\cdot n\log^2 n) = O(n\log^3 n)$ time by Theorem 16. By Lemma 18, running *Extract*(α,β) on line 4 to compute X takes $O((n\log n+|X|)\log^2 n)$ time. Finally, sorting and binary searching over X using Decider on line 5 takes $O((n\log^2 n)(\log|X|)+|X|\log|X|) = O((n\log n+|X|)\log^2 n)$, again by Theorem 16.

Thus in total the time is $O((n\log n+|X|)\log^2 n+n\log^3 n)$. By Lemma 19, with probability at least $1-1/n^c$ we have $|X| \leq cn\ln n$, and thus with probability at least $1-1/n^c$ the total running time is $O(cn\log^3 n)$. ◀

▶ Remark 21. Even in the extremely unlikely event that the algorithm exceeds the $O(n\log^3 n)$ time bound, the worst case running time is only $O(n^2\log^2 n)$.

4 The General Case

In this section, we remove the restriction that P lies in convex position, showing that Problem 1 and Problem 2 can be solved efficiently by converting them into a corresponding graph problem.

For any pair of points $a,b \in \mathbb{R}^2$, define $h_l(a,b)$ to be the closed halfspace bounded by the line going through points a and b, picking the halfspace that is to the left of the directed edge (a,b). Throughout we use $P_{a,b} = P\cap h_l(a,b)$ to denote the subset of P falling in $h_l(a,b)$.

We construct a weighted and fully connected directed graph $G_P = (V,E)$ where $V = P$. For an ordered pair of points (a,b) in P, the weight of its corresponding directed edge is defined as $w(a,b) = cost(\{a,b\},P_{a,b})$, i.e. the distance of the furthest point in $P_{a,b}$ from the segment $\overline{ab}$. (Relating to the previous section, when P is in convex position $w(a,b) = cost_0(a,b)$.) For a cycle of vertices $C = \{p_1,\ldots,p_k\}$, let $w(C)$ denote the maximum of the weights of the directed edges around the cycle. Throughout, we only consider non-trivial cycles, that is cycles must have at least two vertices.

The following lemma shows how to compute edge weights. We remark that the first half of its proof is nearly identical to that for Lemma 15, however, the second half differs.

▶ **Lemma 22.** *Let P be a set of n points in $\mathbb{R}^2$. Then one can compute $w(a,b)$ for all pairs $a,b\in P$ simultaneously in $O(n^2\log n)$ time.*

Proof. Let ℓ denote the line through a and b, which we view as being oriented in the direction from a towards b. Also, let r_a and r_b denote the rays originating at a and b respectively, pointing in the direction orthogonal to ℓ and on the side of ℓ containing $P_{a,b}$.

Observe that the projection of any point $x\in P_{a,b}$ onto ℓ either lies on the portion of ℓ before a, on the line segment $\overline{ab}$, or on the portion of ℓ after b. Thus we have a natural partition of $P_{a,b}$ into three sets, the subset in the right angle cone C_a bounded by ℓ and r_a, those in the slab $Slab(a,b)$ bounded by ℓ, r_a, and r_b, and those in the right angle cone C_b bounded by ℓ and r_b. Observe that for any point x in C_a or C_b, its closest point on $\overline{ab}$ is a or b, respectively, and moreover $||x-\ell|| \leq ||x-\overline{ab}||$. Thus we have that,

$$\begin{aligned} w(a,b) &= \max\{\max_{x\in C_a\cap P_{a,b}} ||x-a||, \max_{x\in C_b\cap P_{a,b}} ||x-b||, \max_{x\in Slab(a,b)\cap P_{a,b}} ||x-\overline{ab}||\} \\ &= \max\{\max_{x\in C_a\cap P_{a,b}} ||x-a||, \max_{x\in C_b\cap P_{a,b}} ||x-b||, \max_{x\in P_{a,b}} ||x-\ell||\}. \end{aligned}$$

Therefore, it suffices to describe how to compute each of the three terms in the stated time. To compute $\max_{x\in P_{a,b}} ||x-\ell||$ we use the standard fact that for any point set P and line ℓ, the furthest point in P from ℓ, on either side of ℓ, is a vertex of $\mathcal{CH}(P)$. Thus the furthest point in $P_{a,b}$ from ℓ is a point of $\mathcal{CH}(P)$. So precompute $\mathcal{CH}(P)$, using any standard $O(n\log n)$ time algorithm, after which we can assume the vertices of $\mathcal{CH}(P)$ are stored in an array sorted in clockwise order. Observe that the subset of the vertices of $\mathcal{CH}(P)$ which are in $P_{a,b}$ is a subarray (or technically two subarrays if it wraps around). So we can determine the ends of this subarray by binary searching. The distances of the points in this subarray to ℓ is a concave function, and so we can binary search to find $\max_{x\in P_{a,b}} ||x-\ell||$. These two binary searches take $O(\log n)$ time per pair a, b, and thus $O(n^2\log n)$ time in total.

To compute the $\max_{x\in C_a\cap P_{a,b}} ||x-a||$ values, we do the following (the b values are computed identically). Consider a right angle cone whose origin is at a. We conceptually rotate this cone around a while maintaining the furthest point of P from a in this cone. The furthest point only changes when a point enters or leaves the cone, and these events can thus easily be obtained by simply angularly sorting the points in P around a. (Note each point corresponds to two events, an entering one, and a leaving one at the entering angle minus $\pi/2$.) To efficiently update the furthest point, we maintain a binary max heap on the distances of the points in the current cone to a. Building the initial max heap and sorting takes $O(n\log n)$ time. Thus all possible right angle cone values at a can be computed in $O(n\log n)$ time, as there are a linear number of events and each event takes $O(\log n)$ time. Moreover, if we store these canonical right angle cone values in sorted angular order, then given a query right angle cone determined by a pair $a, b \in P$ (with cone origin a), the nearest canonical cone can be determined by binary searching. Thus in total computing all $\max_{x\in C_a\cap P_{a,b}} ||x-a||$ values for all pairs a and b takes $O(n^2\log n)$ time. Namely, the precomputation of the canonical cones at each point takes $O(n\log n)$ time per point and thus $O(n^2\log n)$ time for all points. Then for the $O(n^2)$ pairs a, b it takes $O(\log n)$ time to search for its canonical cone. ◀

For a set of points Q, let $\mathcal{CH}_L(Q)$ denote the clockwise list of vertices on the boundary of $\mathcal{CH}(Q)$. Observe that any subset $Q \subseteq P$ corresponds to the cycle $\mathcal{CH}_L(Q)$ in G_P. Moreover, any cycle C corresponds to the convex hull $\mathcal{CH}(C)$. The following lemma is adapted from [13], where Problem 1 was considered but where the *cost* function was determined by a sum of the distances rather than the maximum distance.

▶ **Lemma 23.** *Consider an instance P, ε of Problem 1. The following holds:*

1) *For any cycle C in G_P, $w(C) \geq cost(C, P)$,*

2) *There exists some optimal solution Q such that $w(\mathcal{CH}_L(Q)) = cost(Q, P)$.*

Proof. Recall that $cost(C,P) = \max_{p\in P} ||p - \mathcal{CH}(C)||$. Similarly decomposing $w(C)$ gives,

$$w(C) = \max_{(a,b)\in C} cost(\{a,b\}, P_{a,b}) = \max_{p\in P} \max_{\substack{(a,b)\in C \\ \text{s.t. } p\in P_{a,b}}} ||p-\overline{ab}||.$$

To prove the first part of the lemma, we argue that for any point $p \in P$, its contribution to $w(C)$ is at least as large as its contribution to $cost(C,P)$. Assume $p \notin \mathcal{CH}(C)$, since otherwise it does not contribute to $cost(C,P)$. It suffices to argue there exists an edge

$(a,b) \in C$, such that $p \in P_{a,b}$, since $||p - \overline{ab}|| \geq ||p - \mathcal{CH}(C)||$. So assume otherwise that there is some point $p \in P$ such that p lies strictly to the right of all edges in C. Create a line ℓ that passes through p and any interior point of any edge $(a,b) \in C$, but does not pass through any other point in P. The line ℓ splits the plane into two halfspaces. Observe that since C is a cycle, there must be some edge (c,d) of C which also crosses ℓ, where c is in the same halfspace as b and d in the same halfspace as a (i.e. they have opposite orientations with respect to ℓ). Thus if (c,d) crosses ℓ on the same side of p along ℓ as the edge (a,b) then p would lie to the left of (c,d), as it lies to the right of (a,b). On the other hand, if the intersection of (c,d) with ℓ lied on the opposite side of p along ℓ as the intersection point of (a,b) with ℓ, then $p \in \mathcal{CH}(\{a,b,c,d\}) \subseteq \mathcal{CH}(C)$. Thus either way we have a contradiction.

To prove the second part of the lemma, let Q be some optimal solution. For any $p \in P$, if $p \in \mathcal{CH}(Q)$ then it lies to the right of all edges in $\mathcal{CH}_L(Q)$, and so it does not affect $w(\mathcal{CH}_L(Q))$ or $cost(Q,P)$. So consider a point $p \notin \mathcal{CH}(Q)$. Let $\overline{ab}$ be the closest edge of $\mathcal{CH}(Q)$ (where b follows a in clockwise order). Note that $||p - \mathcal{CH}(Q)|| = ||p - \overline{ab}||$ and $p \in P_{a,b}$, so if p lies to right of all other edges in $\mathcal{CH}_L(Q)$, then its contribution to $w(\mathcal{CH}_L(Q))$ is $||p - \overline{ab}||$. So suppose p lies to the left of some other edge $\overline{cd}$ (note it may be that $b = c$). If this happens, then p is in the intersection of the halfspace to the left of the line from a through b and to the left of the line from c through d. This implies that $b, c \in \mathcal{CH}(\{a,d,p\})$. So let $Q' = Q \cup \{p\} \setminus \{b,c\}$. Observe that $\mathcal{CH}(Q) \subset \mathcal{CH}(Q')$ and $|Q'| \leq |Q|$, and hence Q' is an optimal solution as Q was an optimal solution. Now we repeat this procedure while there remains such a point p to the left of two edges. We repeat this procedure only finitely many times as in each iteration the convex hull becomes larger (i.e. $\mathcal{CH}(Q)$ is a strict subset of $\mathcal{CH}(Q')$). If Q denotes the hull after the final iteration, then by the above we have $w(\mathcal{CH}_L(Q)) = cost(Q,P)$. ◀

▶ **Corollary 24.** *Let P, ε be an instance of Problem 1, and let C^* be the cycle with minimum cardinalty among cycles in G_P with $w(C) \leq \varepsilon$. Then C^* is an optimal solution to the given instance of Problem 1.*

Proof. Using part 1) of Lemma 23 we know that $cost(C^*, P) \leq w(C^*) \leq \varepsilon$, so C^* is a solution. Suppose that C^* is not an optimal solution (i.e. it is not of minimum cardinality). Then by part 2) of Lemma 23, there exists some optimal solution Q with $|Q| < |C^*|$ such that $w(\mathcal{CH}_L(Q)) = cost(Q,P) \leq \varepsilon$. So, there exists a cycle $\mathcal{CH}_L(Q)$ with cost $\leq \varepsilon$ and size less than $|C^*|$, which is a contradiction as C^* had minimal cardinality among such cycles. ◀

In the following we will reduce our problem to the all pairs shortest path problem on directed unweighted graphs, which we denote as APSP. Let $A(n)$ be the time required to solve APSP. In [19] it is shown that $A(n) = \tilde{O}(n^{2+\mu})$,[2] where μ satisfies the equation $\omega(1,\mu,1) = 1 + 2\mu$, and where $\omega(1,\mu,1)$ is the exponent of multiplication of a matrix of size $n \times n^{\mu}$ by a matrix of size $n^{\mu} \times n$. [8] shows that $\mu < 0.5302$ and thus $A(n) = O(n^{2.5302})$.

▶ **Theorem 25.** *Any instance P, ε of Problem 1 can be solved in time*

$$O(A(n) + n^2 \log n) = O(n^{2.5302}).$$

Proof. By Corollary 24, in order to solve Problem 1, we just need to find a minimum length cycle with weight at most ε in the graph G_P defined above. By definition a cycle has weight $\leq \varepsilon$ if and only if all of its edge weights are $\leq \varepsilon$. So let G_P^{ε} be the unweighted and directed

[2] We use the standard convention that $\tilde{O}(f(n))$ denotes $O(f(n)\log^c n)$ for some $c > 0$.

graph obtained from G_P by removing all edges with weight $> \varepsilon$. Thus the solution to our problem corresponds to the minimum length cycle in this unweighted graph G_P^{ε}. This can be solved by computing APSP in G_P^{ε}. Specifically, the solution is determined by the ordered pair (a, b) with the shortest path subject to the directed edge (b, a) existing in G_P^{ε} (i.e. it is the shortest path that can be completed into a cycle).

Computing all of the edge weights in G_P can be done in $O(n^2 \log n)$ time by Lemma 22. Converting G_P into G_P^{ε} then takes $O(n^2)$ time. Given the APSP distances, finding the minimum length cycle takes $O(n^2)$ time by scanning all pairs to check for an edge. APSP on directed unweighted graphs can be solved in $A(n) = O(n^{2.5032})$ time as described above. So, the total time is $O(A(n) + n^2 \log n) = O(n^{2.5302})$. ◀

Let $A_k(n)$ denote the time it takes to solve APSP on directed unweighted graphs where path lengths are bounded by k (i.e. the path length is infinite if there is no k length path). [4] showed that $A_k(n) = O(n^{\omega} k \log^2 k)$, where ω is the exponent of (square) matrix multiplication. [6] showed that $\omega < 2.376$.

▶ **Theorem 26.** *Any instance P, k of Problem 2 can be solved in time*

$$O(\min\{A(n), A_k(n)\}(\log n) + n^2 \log n) = O(\min\{n^{2.5302}, kn^{2.376}\}).$$

Proof. The idea is to binary search using Theorem 25. Namely, the optimal solution to the instance P, k of Problem 2 has cost $\leq \varepsilon$ if and only the optimal solution to the instance P, ε of Problem 1 uses $\leq k$ points. Moreover, the weight of any cycle in G_P is determined by the weight of some edge, and thus by the above discussion the optimal solution to the given instance of Problem 2 will be the weight of some edge. There are $O(n^2)$ edge weights, which we can enumerate, sort, and binary search over using Theorem 25. Computing all of the edge weights in G_P and sorting them can be done in $O(n^2 \log n)$ time by Lemma 22. Thus by Theorem 25, the total time is $O(A(n)(\log n) + n^2 \log n) = O(n^{2.5302})$ (Note we only compute all edge weights a single time, so each step of the binary search then costs $O(A(n))$ time.)

Alternatively, since we know the value of k, we can get a potentially faster time for when k is small, by only considering length at most k paths. Specifically, in each call to our decision procedure (i.e. Theorem 25) instead of computing APSP, compute the APSP restricted to length k paths. Then, by the discussion before the theorem, the running time becomes $O(A_k(n)(\log n) + n^2 \log n) = O((n^{\omega} k \log^2 k)(\log n) + n^2 \log n)) = O(kn^{2.376})$. ◀

4.1 Faster Approximations

While our focus in the paper is on exact algorithms, in this section we show how the results above imply faster approximate solutions for the general case. First, we show that the results from Section 3 for points in convex position immediately yield near linear time 2-approximations for the general case. More precisely, we have the following, where $V(\mathcal{CH}(P))$ denotes the vertices of the convex hull of P (and recall $V(\mathcal{CH}(P)) \subseteq P$).

▶ **Lemma 27.** *Let P be a point set in the plane. Suppose there exists some subset $Q \subseteq P$ such that cost$(Q, P) \leq \varepsilon$ and $|Q| \leq k$. Then there exists a subset $Q' \subseteq V(\mathcal{CH}(P))$ such that cost$(Q', P) \leq \varepsilon$ and $|Q'| \leq 2k$.*

Proof. Let $Q = \{q_1, \ldots, q_k\}$, where the points are labeled in clockwise order. First, we convert Q into a subset of points on the boundary of $\mathcal{CH}(P)$. Specifically, consider the segment $\overline{q_{i-1}q_i}$. Consider the ray with base point q_{i-1}, and passing through q_i, and let z be the point of intersection of this ray with the boundary of $\mathcal{CH}(P)$. Let $Q_z = \{q_1, \ldots, q_{i-1}, z, q_{i+1}, \ldots, q_k\}$, and observe that $\mathcal{CH}(Q) \subseteq \mathcal{CH}(Q_z)$ as q_i lies on the segment $\overline{q_{i-1}z}$. Let $\overline{xy}$ be the edge of $\mathcal{CH}(P)$ which contains z, and let $Q'_z = \{q_1, \ldots, q_{i-1}, x, y, q_{i+1}, \ldots, q_k\}$. (Note if $z \in V(\mathcal{CH}(P))$ then we set $Q'_z = Q_z$.) Since $z \in \overline{xy}$, we have that $\mathcal{CH}(Q) \subseteq \mathcal{CH}(Q_z) \subseteq \mathcal{CH}(Q'_z)$. Thus $cost(Q'_z, P) \leq \varepsilon$ and $|Q'_z| \leq k+1$. So if we repeat this procedure for all i then we will end up with a set Q' such that $cost(Q', P) \leq \varepsilon$, $|Q'| \leq 2k$, and $Q' \subseteq V(\mathcal{CH}(P))$. ◀

Given an instance P, ε of Problem 1, where the optimal solution Q has size k, the above implies there is set $Q' \subseteq V(\mathcal{CH}(P))$ such that $cost(Q', P) \leq \varepsilon$ and $|Q'| \leq 2k$. Such a set can be found using the algorithm of Theorem 16 for the instance $V(\mathcal{CH}(P)), \varepsilon$ of Problem 3, as Q' is a candidate solution for this instance. Also, recall for $X \subseteq P$, the furthest point in P from $\mathcal{CH}(X)$ is always in $V(\mathcal{CH}(P))$, and so if $cost(X, V(\mathcal{CH}(P))) \leq \varepsilon$ then $cost(X, P) \leq \varepsilon$.

Similarly, given an instance P, k of Problem 2, where the optimal solution Q has cost ε, the above implies there is set $Q' \subseteq V(\mathcal{CH}(P))$ such that $cost(Q', P) \leq \varepsilon$ and $|Q'| \leq 2k$. Such a set can be found using the algorithm of Theorem 20 for the instance $V(\mathcal{CH}(P)), 2k$ of Problem 4, again as Q' is a candidate solution. Thus we have the following.

▶ **Theorem 28.** *Let P, ε be an instance of Problem 1, where the optimal solution Q has size k. Then in $O(n \log^2 n)$ time one can compute a set $Q' \subseteq V(\mathcal{CH}(P))$ such that $cost(Q', P) \leq \varepsilon$ and $|Q'| \leq 2k$.*

Similarly, let P, k be an instance of Problem 2, where the optimal solution Q has cost ε. Then with probability $\geq 1 - 1/n^c$, for any constant c, in $O(n \log^3 n)$ time one can compute a set $Q' \subseteq V(\mathcal{CH}(P))$ such that $cost(Q', P) \leq \varepsilon$ and $|Q'| \leq 2k$.

Finally, we remark that if one allows approximating the best k point solution with $k+1$ points (i.e. a $(1 + 1/k)$-approximation), then our graph algorithms from the previous subsection imply near quadratic time approximations (i.e. compared to the theorem above, we are trading near linear running time for approximation quality). The idea is, rather than solving APSP, if we chose an appropriate starting point, we can instead solve for single source shortest paths. Similar observations have been made before for related problems [3, 14].

For a given instance P, ε of Problem 1, let Q be an optimal solution where $|Q| = k$. Let p be an arbitrary point in $V(\mathcal{CH}(P))$. Let $Q' = Q \cup \{p\}$. Observe that $cost(Q', P) \leq cost(Q, P) \leq \varepsilon$ and $|Q'| \leq k+1$. Thus the optimal solution to this instance of Problem 1, but where we require it include p, is a valid solution to the instance without this requirement, and uses at most one more point.

Now we sketch how the results from Section 4 directly extend to the case where we want the optimal solution restricted to including p. Specifically, for the analogue of Corollary 24, let C^* be the minimum cardinality cycle in G_P with weight at most ε such that the cycle includes p. To argue that C^* is an optimal solution to the given instance of Problem 1 among those which must include the point p, we need to extend Lemma 23 to require including p. Part 1) of the lemma immediately extends. The proof of Part 2) starts with some optimal solution Q. It then performs a transformation of Q into a set Q' so that points are not to the left of two edges, which one can argue implies $w(\mathcal{CH}_L(Q')) = cost(Q', P)$. This transformation has the properties that $|Q'| \leq |Q|$ and $\mathcal{CH}(Q) \subseteq \mathcal{CH}(Q')$, and hence $cost(Q', P) \leq cost(Q, P)$, and so since Q was optimal so is Q'. If instead we perform this same transformation on an optimal solution restricted to containing p, call it X, then the same argument implies we produce a set X' such that $w(\mathcal{CH}_L(X')) = cost(X', P)$, $|X'| \leq |X|$, and $\mathcal{CH}(X) \subseteq \mathcal{CH}(X')$. Moreover, because $\mathcal{CH}(X) \subseteq \mathcal{CH}(X')$ and $p \in V(\mathcal{CH}(P))$, crucially we have $p \in \mathcal{CH}_L(X')$. Thus the modified Lemma 23 and hence Corollary 24, where p is included, both hold.

To find the optimal solution to Problem 1 containing p, we now use the same approach as in Theorem 25. The difference now however, is that we only need to compute single source shortest paths in G_P^ε rather than APSP, since we can use p as our starting point. Let $S(n)$

be the time to compute single source shortest paths. In an unweighted graph using BFS gives $S(n) = O(|E| + |V|) = O(n^2)$. Thus replacing $A(n)$ with $S(n)$ in the running time statement of Theorem 25 gives $O(S(n) + n^2 \log n) = O(n^2 \log n)$. Similarly, replacing $A(n)$ with $S(n)$ for Theorem 26 gives $O(S(n) \log n + n^2 \log n) = O(n^2 \log n)$. Thus we have the following.

▶ **Theorem 29.** *Let P, ε be an instance of Problem 1, with optimal solution Q. Then in $O(n^2 \log n)$ time one can compute a set $Q' \subseteq P$ such that $cost(Q', P) \leq \varepsilon$ and $|Q'| \leq |Q| + 1$.*

Similarly, let P, k be an instance of Problem 2, where the optimal solution Q has cost ε. Then in $O(n^2 \log n)$ time one can compute a set $Q' \subseteq P$ such that $cost(Q', P) \leq \varepsilon$ and $|Q'| \leq |Q| + 1$.

References

1 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004. doi:10.1145/1008731.1008736.

2 Pankaj K. Agarwal and Kasturi R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discret. Comput. Geom.*, 23(2):273–291, 2000. doi:10.1007/PL00009500.

3 Alok Aggarwal, Heather Booth, Joseph O'Rourke, Subhash Suri, and Chee-Keng Yap. Finding minimal convex nested polygons. *Inf. Comput.*, 83(1):98–110, 1989. doi:10.1016/0890-5401(89)90049-7.

4 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997. doi:10.1006/jcss.1997.1388.

5 Avrim Blum, Sariel Har-Peled, and Benjamin Raichel. Sparse approximation via generating point sets. *ACM Trans. Algorithms*, 15(3):32:1–32:16, 2019. doi:10.1145/3302249.

6 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

7 Ovidiu Daescu, Ningfang Mi, Chan-Su Shin, and Alexander Wolff. Farthest-point queries with geometric and combinatorial constraints. *Comput. Geom.*, 33(3):174–185, 2006. doi:10.1016/j.comgeo.2005.07.002.

8 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 514–523, 2012. doi:10.1109/FOCS.2012.80.

9 Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Toth. *Handbook of Discrete and Computational Geometry, Third Edition.* CRC Press, 2017.

10 Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors. *Handbook of Discrete and Computational Geometry, Third Edition.* Chapman and Hall/CRC, 2018.

11 Sariel Har-Peled and Benjamin Raichel. The fréchet distance revisited and extended. *ACM Trans. Algorithms*, 10(1):3:1–3:22, 2014. doi:10.1145/2532646.

12 G. Klimenko and B. Raichel. Fast and exact convex hull simplification, October 2021. arXiv:2110.00671.

13 Georgiy Klimenko, Benjamin Raichel, and Gregory Van Buskirk. Sparse convex hull coverage. In *Canadian Conference on Computational Geometry (CCCG)*, pages 15–25, 2020.

14 Mario Alberto López and Shlomo Reisner. Hausdorff approximation of convex polygons. *Comput. Geom.*, 32(2):139–158, 2005. doi:10.1016/j.comgeo.2005.02.002.

15 Mees van de Kerkhof, Irina Kostitsyna, Maarten Löffler, Majid Mirzanezhad, and Carola Wenk. Global curve simplification. In *27th Annual European Symposium on Algorithms (ESA)*, volume 144 of *LIPIcs*, pages 67:1–67:14, 2019. doi:10.4230/LIPIcs.ESA.2019.67.

16 Ivor van der Hoog, Vahideh Keikha, Maarten Löffler, Ali Mohades, and Jérôme Urhausen. Maximum-area triangle in a convex polygon, revisited. *Inf. Process. Lett.*, 161:105943, 2020. doi:10.1016/j.ipl.2020.105943.

17 Marc J. van Kreveld, Maarten Löffler, and Lionov Wiratma. On optimal polyline simplification using the hausdorff and fréchet distance. *J. Comput. Geom.*, 11(1):1–25, 2020. URL: `https://journals.carleton.ca/jocg/index.php/jocg/article/view/415`.

18 Yanhao Wang, Michael Mathioudakis, Yuchen Li, and Kian-Lee Tan. Minimum coresets for maxima representation of multidimensional data. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 138–152. ACM, 2021. `doi:10.1145/3452021.3458322`.

19 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. `doi:10.1145/567112.567114`.

Lower Bounds and Improved Algorithms for Asymmetric Streaming Edit Distance and Longest Common Subsequence

Xin Li ✉
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Yu Zheng ✉
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Abstract

In this paper, we study *edit distance* (ED) and *longest common subsequence* (LCS) in the asymmetric streaming model, introduced by Saks and Seshadhri [26]. As an intermediate model between the random access model and the streaming model, this model allows one to have streaming access to one string and random access to the other string. Meanwhile, ED and LCS are both fundamental problems that are often studied on large strings, thus the (asymmetric) streaming model is ideal for studying these problems.

Our first main contribution is a systematic study of space lower bounds for ED and LCS in the asymmetric streaming model. Previously, there are no explicitly stated results in this context, although some lower bounds about LCS can be inferred from the lower bounds for *longest increasing subsequence* (LIS) in [28, 16, 14]. Yet these bounds only work for large alphabet size. In this paper, we develop several new techniques to handle ED in general and LCS for small alphabet size, thus establishing strong lower bounds for both problems. In particular, our lower bound for ED provides an *exponential* separation between edit distance and Hamming distance in the asymmetric streaming model. Our lower bounds also extend to LIS and *longest non-decreasing subsequence* (LNS) in the standard streaming model. Together with previous results, our bounds provide an almost complete picture for these two problems.

As our second main contribution, we give improved algorithms for ED and LCS in the asymmetric streaming model. For ED, we improve the space complexity of the constant factor approximation algorithms in [15, 13] from $\tilde{O}(\frac{n^\delta}{\delta})$ to $O(\frac{d^\delta}{\delta}\ \mathsf{polylog}(n))$, where n is the length of each string and d is the edit distance between the two strings. For LCS, we give the first $1/2+\varepsilon$ approximation algorithm with space n^δ for any constant $\delta > 0$, over a binary alphabet. Our work leaves a plethora of intriguing open questions, including establishing lower bounds and designing algorithms for a natural generalization of LIS and LNS, which we call *longest non-decreasing subsequence with threshold* (LNST).

2012 ACM Subject Classification Theory of computation → Lower bounds and information complexity

Keywords and phrases Asymmetric Streaming Model, Edit Distance, Longest Common Subsequence, Space Lower Bound

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.27

Related Version *Full Version*: `https://arxiv.org/abs/2103.00713`

Funding *Xin Li*: Supported by NSF CAREER Award CCF-1845349.
Yu Zheng: Supported by NSF CAREER Award CCF-1845349

1 Introduction

Edit distance (ED) and longest common subsequence (LCS) are two classical problems studied in the context of measuring similarities between two strings. Edit distance is defined as the smallest number of edit operations (insertions, deletions, and substitutions) to transform one

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 27; pp. 27:1–27:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

string to the other, while longest common subsequence is defined as the longest string that appears as a subsequence in both strings. These two problems have found wide applications in areas such as bioinformatics, text and speech processing, compiler design, data analysis, image analysis and so on. In turn, these applications have led to an extensive study of both problems.

With the era of information explosion, nowadays these two problems are often studied on very large strings. For example, in bioinformatics a human genome can be represented as a string with 3 billion letters (base pairs). Such data provides a huge challenge to the algorithms for ED and LCS, as the standard algorithms for these two problems using dynamic programming need $\Theta(n^2)$ time and $\Theta(n)$ space where n is the length of each string. These bounds quickly become infeasible or too costly as n becomes large, such as in the human genome example. Especially, some less powerful computers may not even have enough memory to store the data, let alone processing it.

One appealing approach to dealing with big data is designing *streaming algorithms*, which are algorithms that process the input as a data stream. Typically, the goal is to compute or approximate the solution by using sublinear space (e.g., n^α for some constant $0 < \alpha < 1$ or even $\mathsf{polylog}(n)$) and a few (ideally one) passes of the data stream. These algorithms have become increasingly popular, and attracted a lot of research activities recently.

Designing streaming algorithms for ED and LCS, however, is not an easy task. For ED, only a couple of positive results are known. In particular, assuming that the edit distance between the two strings is bounded by some parameter k, [10] gives a randomized one pass algorithm achieving an $O(k)$ approximation of ED, using linear time and $O(\log n)$ space, in a variant of the streaming model where one can scan the two strings simultaneously in a coordinated way. In the same model [10] also give randomized one pass algorithms computing ED exactly, using space $O(k^6)$ and time $O(n+k^6)$. This was later improved to space $O(k)$ and time $O(n+k^2)$ in [11, 7]. Furthermore, [7] give a randomized one pass algorithm computing ED exactly, using space $\tilde{O}(k^8)$ and time $\tilde{O}(k^2 n)$, in the standard streaming model. We note that all of these algorithms are only interesting if k is small, e.g., $k \leq n^\alpha$ where α is some small constant, otherwise the space complexity can be as large as n. For LCS, strong lower bounds are given in [22, 28], which show that for exact computation, even constant pass randomized algorithms need space $\Omega(n)$; while any constant pass deterministic algorithm achieving a $\frac{2}{\sqrt{n}}$ approximation of LCS also needs space $\Omega(n)$, if the alphabet size is at least n.

Motivated by this situation and inspired by the work of [4], Saks and Seshadhri [26] studied the asymmetric data streaming model. This model is a relaxation of the standard streaming model, where one has streaming access to one string (say x), and random access to the other string (say y). In this model, [26] gives a deterministic one pass algorithm achieving a $1+\varepsilon$ approximation of $n - \mathsf{LCS}$ using space $O(\sqrt{(n\log n)/\varepsilon})$, as well as a randomized one pass algorithm algorithm achieving an εn *additive* approximation of LCS using space $O(k\log^2 n/\varepsilon)$ where k is the maximum number of times any symbol appears in y. Another work by Saha [25] also gives an algorithm in this model that achieves an εn *additive* approximation of ED using space $O(\frac{\sqrt{n}}{\epsilon})$.

The asymmetric streaming model is interesting for several reasons. First, it still inherits the spirit of streaming algorithms, and is particularly suitable for a distributed setting. For example, a local, less powerful computer can use the streaming access to process the string x, while sending queries to a remote, more powerful server which has access to y. Second, because it is a relaxation of the standard streaming model, one can hope to design better algorithms for ED or to beat the strong lower bounds for LCS in this model. The latter point is indeed verified by two recent works [15, 13] (recently accepted to ICALP as a combined paper [12]), which give a deterministic one pass algorithm achieving a $O(2^{1/\delta})$ approximation of ED, using space $\tilde{O}(n^{\delta}/\delta)$ and time $\tilde{O}_{\delta}(n^4)$ for any constant $\delta > 0$, as well as deterministic one pass algorithms achieving $1 \pm \varepsilon$ approximation of ED and LCS, using space $\tilde{O}(\frac{\sqrt{n}}{\varepsilon})$ and time $\tilde{O}_{\varepsilon}(n^2)$.

A natural question is how much we can improve these results. Towards answering this question, we study both lower bounds and upper bounds for the space complexity of ED and LCS in the asymmetric streaming model, and we obtain several new, non-trivial results.

Related work. On a different topic, there are many works that study the time complexity of ED and LCS. In particular, while [6, 1] showed that ED and LCS cannot be computed exactly in truly sub-quadratic time unless the strong Exponential time hypothesis [19] is false, a successful line of work [9, 8, 20, 5, 18, 23, 24] has led to randomized algorithms that achieve constant approximation of ED in near linear time, and randomized algorithms that provide various non-trivial approximation of LCS in linear or sub-quadratic time. Another related work is [4], where the authors proved a lower bound on the *query complexity* for computing ED in the *asymmetric query* model, where one have random access to one string but only limited number of queries to the other string.

1.1 Our Contribution

We initiate a systematic study on lower bounds for computing or approximating ED and LCS in the asymmetric streaming model. To simplify notation we always use $1+\varepsilon$ approximation for some $\varepsilon > 0$, i.e., outputting an λ with $\mathsf{OPT} \leq \lambda \leq (1+\varepsilon)\mathsf{OPT}$, where OPT is either $\mathsf{ED}(x, y)$ or $\mathsf{LCS}(x, y)$. We note that for LCS, this is equivalent to a $1/(1+\varepsilon)$ approximation in the standard notation.

Previously, there are no explicitly stated space lower bounds in this model, although as we will discuss later, some lower bounds about LCS can be inferred from the lower bounds for *longest increasing subsequence* LIS in [28, 16, 14]. As our first contribution, we prove strong lower bounds for ED in the asymmetric streaming model.

▶ **Theorem 1.** *There is a constant $c > 1$ such that for any $k, n \in \mathbb{N}$ with $n \geq ck$, given an alphabet Σ, any R-pass randomized algorithm in the asymmetric streaming model that decides if $\mathsf{ED}(x, y) \geq k$ for two strings $x, y \in \Sigma^n$ with success probability $\geq 2/3$ must use space $\Omega(\mathsf{min}(k, |\Sigma|)/R)$.*

This theorem implies the following corollary.

▶ **Corollary 2.** *Given an alphabet Σ, the following space lower bounds hold for any constant pass randomized algorithm with success probability $\geq 2/3$ in the asymmetric streaming model.*

1. *$\Omega(n)$ for computing $\mathsf{ED}(x, y)$ of two strings $x, y \in \Sigma^n$ if $|\Sigma| \geq n$.*
2. *$\Omega(\frac{1}{\varepsilon})$ for $1 + \varepsilon$ approximation of $\mathsf{ED}(x, y)$ for two strings $x, y \in \Sigma^n$ if $|\Sigma| \geq 1/\varepsilon$.*

Our theorems thus provide a justification for the study of *approximating* ED in the asymmetric streaming model. Furthermore, we note that previously, unconditional lower bounds for ED in various computational models are either weak, or almost identical to the bounds for Hamming distance. For example, a simple reduction from the equality function implies the deterministic two party communication complexity (and hence also the space lower bound in the standard streaming model) for computing or even approximating ED is

$\Omega(n)$.[1] However the same bound holds for Hamming distance. Thus it has been an intriguing question to prove a rigorous, unconditional separation of the complexity of ED and Hamming distance. To the best of our knowledge the only previous example achieving this is the work of [3] and [2], which showed that the randomized two party communication complexity of achieving a $1+\varepsilon$ approximation of ED is $\Omega(\frac{\log n}{(1+\varepsilon)\log\log n})$, while the same problem for Hamming distance has an upper bound of $O(\frac{1}{\varepsilon^2})$. Thus if ε is a constant, this provides a separation of $\Omega(\frac{\log n}{\log\log n})$ vs. a constant. However, this result also has some disadvantages: (1) It only works in the randomized setting; (2) The separation becomes obsolete when ε is small, e.g., $\varepsilon = 1/\sqrt{\log n}$; and (3) The lower bound for ED is still weak and thus it does not apply to the streaming setting, as there even recoding the index needs space $\log n$.

Our result from Corollary 2, on the other hand, complements the above result in the aforementioned aspects by providing another strong separation of ED and Hamming distance. Note that even exact computation of the Hamming distance between x and y is easy in the asymmetric streaming model with one pass and space $O(\log n)$. Thus our result provides an *exponential* gap between edit distance and Hamming distance, in terms of the space complexity in the asymmetric streaming model (and also the communication model since our proof uses communication complexity), even for deterministic exact computation.

Next we turn to LCS, which can be viewed as a generalization of LIS. For example, if the alphabet $\Sigma = [n]$, then we can fix the string y to be the concatenation from 1 to n, and it's easy to see that $\mathsf{LCS}(x,y) = \mathsf{LIS}(x)$. Therefore, the lower bound of computing LIS for randomized streaming in [28] with $|\Sigma| \geq n$ also implies a similar bound for LCS in the asymmetric streaming model. However, the bound in [28] does not apply to the harder case where x is a *permutation* of y, and their lower bound where $|\Sigma| < n$ is actually for *longest non-decreasing subsequence*, which does not give a similar bound for LCS in the asymmetric streaming model. [2] Therefore, we first prove a strong lower bound for LCS in general.

▶ **Theorem 3.** *There is a constant $c > 1$ such that for any $k, n \in \mathbb{N}$ with $n \geq ck$, given an alphabet Σ, any R-pass randomized algorithm in the asymmetric streaming model that decides if $\mathsf{LCS}(x,y) \geq k$ for two strings $x, y \in \Sigma^n$ with success probability $\geq 2/3$ must use space $\Omega\big(\mathsf{min}(k, |\Sigma|)/R\big)$. Moreover, this holds even if x is a permutation of y when $|\Sigma| \geq n$ or $|\Sigma| \leq k$.*

Similar to the case of ED, this theorem also implies the following corollary.

▶ **Corollary 4.** *Given an alphabet Σ, the following space lower bounds hold for any constant pass randomized algorithm with success probability $\geq 2/3$ in the asymmetric streaming model.*

1. *$\Omega(n)$ for computing $\mathsf{LCS}(x,y)$ of two strings $x, y \in \Sigma^n$ if $|\Sigma| \geq n$.*
2. *$\Omega(\frac{1}{\varepsilon})$ for $1+\varepsilon$ approximation of $\mathsf{LCS}(x,y)$ for two strings $x, y \in \Sigma^n$ if $|\Sigma| \geq 1/\varepsilon$.*

We then consider *deterministic* approximation of LCS. Here, the work of [16, 14] gives a lower bound of $\Omega\left(\frac{1}{R}\sqrt{\frac{n}{\varepsilon}}\log\left(\frac{|\Sigma|}{\varepsilon n}\right)\right)$ for any R pass streaming algorithm achieving a $1+\varepsilon$ approximation of LIS, which also implies a lower bound of $\Omega\left(\frac{1}{R}\sqrt{\frac{n}{\varepsilon}}\log\left(\frac{1}{\varepsilon}\right)\right)$ for asymmetric streaming LCS when $|\Sigma| \geq n$. These bounds match the upper bound in [17] for LIS and LNS, and in [15, 13] for LCS. However, a major drawback of this bound is that it gives nothing when $|\Sigma|$ is small (e.g., $|\Sigma| \leq \varepsilon n$). For even smaller alphabet size, the bound does not even

[1] We include this bound in the appendix for completeness, as we cannot find any explicit statement in the literature.

[2] One can get a similar reduction to LCS, but now y needs to be the sorted version of x, which gives additional information about x in the asymmetric streaming model since we have random access to y.

give anything for exact computation. For example, in the case of a binary alphabet, we know that $\mathsf{LIS}(x) \leq 2$ and thus taking $\varepsilon = 1/2$ corresponds to exact computation. Yet the bound gives a negative number.

This is somewhat disappointing as in most applications of ED and LCS, the alphabet size is actually a fixed constant. These include for example the English language and the human DNA sequence (where the alphabet size is 4 for the 4 bases). Therefore, in this paper we focus on the case where the alphabet size is small, and we have the following theorem.

▶ **Theorem 5.** *Given an alphabet Σ, for any $\varepsilon > 0$ where $\frac{|\Sigma|^2}{\varepsilon} = O(n)$, any R-pass deterministic algorithm in the asymmetric streaming model that computes a $1+\varepsilon$ approximation of $\mathsf{LCS}(x, y)$ for two strings $x, y \in \Sigma^n$ must use space $\Omega\left(\frac{|\Sigma|}{\varepsilon}/R\right)$.*

Thus, even for a binary alphabet, achieving $1+\varepsilon$ approximation for small ε (e.g., $\varepsilon = 1/n$ which corresponds to exact computation) can take space as large as $\Omega(n)$ for any constant pass algorithm. Further note that by taking $|\Sigma| = \sqrt{\varepsilon n}$, we recover the $\Omega\left(\frac{\sqrt{n}}{\varepsilon}/R\right)$ bound with a much smaller alphabet.

Finally, we turn to LIS and longest non-decreasing subsequence (LNS), as well as a natural generalization of LIS and LNS which we call *longest non-decreasing subsequence with threshold* (LNST). Given a string $x \in \Sigma^n$ and a threshold $t \leq n$, $\mathsf{LNST}(x, t)$ denotes the length of the longest non-decreasing subsequence in x such that each symbol appears at most t times. It is easy to see that the case of $t = 1$ corresponds to LIS and the case of $t = n$ corresponds to LNS. Thus LNST is indeed a generalization of both LIS and LNS. It is also a special case of LCS when $|\Sigma| t \leq n$ as we can take y to be the concatenation of t copies of each symbol, in the ascending order (and possibly padding some symbols not in x). How hard is LNST? We note that in the case of $t = 1$ (LIS) and $t = n$ (LNS) a simple dynamic programming can solve the problem in one pass with space $O(|\Sigma| \log n)$, and $1+\varepsilon$ approximation can be achieved in one pass with space $\tilde{O}(\sqrt{\frac{n}{\varepsilon}})$ by [17]. Thus one can ask what is the situation for other t. Again we focus on the case of a small alphabet and have the following theorem.

▶ **Theorem 6.** *Given an alphabet Σ, for deterministic $(1+\varepsilon)$ approximation of $\mathsf{LNST}(x, t)$ for a string $x \in \Sigma^n$ in the streaming model with R passes, we have the following space lower bounds:*

1. *$\Omega(\min(\sqrt{n}, |\Sigma|)/R)$ for any constant t (this includes LIS), when ε is any constant.*
2. *$\Omega(|\Sigma| \log(1/\varepsilon)/R)$ for $t \geq n/|\Sigma|$ (this includes LNS), when $|\Sigma|^2/\varepsilon = O(n)$.*
3. *$\Omega\left(\frac{\sqrt{|\Sigma|}}{\varepsilon}/R\right)$ for $t = \Theta(1/\varepsilon)$, when $|\Sigma|/\varepsilon = O(n)$.*

Thus, case 1 and 2 show that even for any constant approximation, any constant pass streaming algorithm for LIS and LNS needs space $\Omega(|\Sigma|)$ when $|\Sigma| \leq \sqrt{n}$, matching the $O(|\Sigma| \log n)$ upper bound up to a logarithmic factor. Taking $\varepsilon = 1/\sqrt[3]{n}$ and $|\Sigma| \leq \sqrt[3]{n}$ for example, we further get a lower bound of $\Omega(|\Sigma| \log n)$ for approximating LNS using any constant pass streaming algorithm. This matches the $O(|\Sigma| \log n)$ upper bound. These results complement the bounds in [16, 14, 17] for the important case of small alphabet, and together they provide an almost complete picture for LIS and LNS. Case 3 shows that for certain choices of t and ε, the space we need for LNST can be significantly larger than those for LIS and LNS. It is an intriguing question to completely characterize the behavior of LNST for all regimes of parameters.

We also give improved algorithms for asymmetric streaming ED and LCS. For ED, [15, 13] gives a $O(2^{1/\delta})$-approximation algorithm with $\tilde{O}(n^\delta)$ space for any constant $\delta \in (0, 1)$. We further reduced the space needed from $\tilde{O}(\frac{n^\delta}{\delta})$ to $O(\frac{d^\delta}{\delta}\ \mathsf{polylog}(n))$ where $d = \mathsf{ED}(x, y)$. Specifically, we have the following theorem.

▶ **Theorem 7.** *Assume* $\mathsf{ED}(x,y)=d$, *in the asymmetric streaming model, there are one-pass deterministic algorithms in polynomial time with the following parameters:*

1. *A* $(3+\varepsilon)$*-approximation of* $\mathsf{ED}(x,y)$ *using* $O(\sqrt{d}\,\mathsf{polylog}(n))$ *space.*
2. *For any constant* $\delta \in (0,1/2)$*, a* $2^{O(\frac{1}{\delta})}$*-approximation of* $\mathsf{ED}(x,y)$ *using* $O(\frac{d^{\delta}}{\delta}\mathsf{polylog}(n))$ *space.*

For LCS over a large alphabet, the upper bounds in [15, 13] match the lower bounds implied by [16, 14]. We thus again focus on small alphabet. Note that our Theorem 5 does not give anything useful if $|\Sigma|$ is small and ε is large (e.g., both are constants). Thus a natural question is whether one can get better bounds. In particular, is the dependence on $1/\varepsilon$ linear as in our theorem, or is there a threshold beyond which the space jumps to say for example $\Omega(n)$? We note that there is a trivial one pass, $O(\log n)$ space algorithm even in the standard streaming model that gives a $|\Sigma|$ approximation of LCS (or $1/|\Sigma|$ approximation in standard notation), and no better approximation using sublinear space is known even in the asymmetric streaming model. Thus one may wonder whether this is the threshold. We show that this is not the case, by giving a one pass algorithm in the asymmetric streaming model over the binary alphabet that achieves a $2-\varepsilon$ approximation of LCS (or $1/2+\varepsilon$ approximation in standard notation), using space n^{δ} for any constant $\delta > 0$.

▶ **Theorem 8.** *For any constant* $\delta \in (0,1/2)$*, there exists a constant* $\varepsilon > 0$ *and a one-pass deterministic algorithm that outputs a* $2-\varepsilon$ *approximation of* $\mathsf{LCS}(x,y)$ *for any two strings* $x, y \in \{0,1\}^n$*, with* $\tilde{O}(n^{\delta}/\delta)$ *space and polynomial time in the asymmetric streaming model.*

Finally, as mentioned before, we now have an almost complete picture for LIS and LNS, but for the more general LNST the situation is still far from clear. Since LNST is a special case of LCS, if $|\Sigma|t = O(n)$ then the upper bound of $\tilde{O}(\frac{\sqrt{n}}{\varepsilon})$ in [15, 13] still applies and this matches our lower bound in case 3, Theorem 6 by taking $|\Sigma| = \varepsilon n$. One can then ask the natural question of whether we can get a matching upper bound for the case of small alphabet. We are not able to achieve this, but we provide a simple algorithm that can use much smaller space for certain regimes of parameters in this case.

▶ **Theorem 9.** *Given an alphabet* Σ *with* $|\Sigma| = r$*. For any* $\varepsilon > 0$ *and* $t \geq 1$*, there is a one-pass streaming algorithm that computes a* $(1+\varepsilon)$ *approximation of* $\mathsf{LNST}(x,t)$ *for any* $x \in \Sigma^n$ *with* $\tilde{O}\Big(\big(\min(t, r/\varepsilon)+1\big)^r\Big)$ *space.*

1.2 Overview of our Techniques

Here we provide an informal overview of the techniques used in this paper.

1.2.1 Lower Bounds

Our lower bounds use the general framework of communication complexity. To limit the power of random access to the string y, we always fix y to be a specific string, and consider different strings x. In turn, we divide x into several blocks and consider the two party/multi party communication complexity of $\mathsf{ED}(x,y)$ or $\mathsf{LCS}(x,y)$, where each party holds one block of x. However, we need to develop several new techniques to handle edit distance and small alphabets.

Edit distance. We start with edit distance. One difficulty here is to handle substitutions, as with substitutions edit distance becomes similar to Hamming distance, and this is exactly one of the reasons why strong complexity results separating edit distance and Hamming distance are rare. Indeed, if we define $\mathsf{ED}(x, y)$ to be the smallest number of insertions and deletions (without substitutions) to transform x into y, then $\mathsf{ED}(x, y) = 2n - 2\mathsf{LCS}(x, y)$ and thus a lower bound for exactly computing LCS (e.g., those implied from [16, 14]) would translate directly into the same bound for exactly computing ED. On the other hand, with substitutions things become more complicated: if $\mathsf{LCS}(x, y)$ is small (e.g., $\mathsf{LCS}(x, y) \leq n/2$) then in many cases (such as examples obtained by reducing from [16, 14]) the best option to transform x into y is just replacing each symbol in x by the corresponding symbol in y if they are different, which makes $\mathsf{ED}(x, y)$ exactly the same as their Hamming distance.

To get around this, we need to ensure that $\mathsf{LCS}(x, y)$ is large. We demonstrate our ideas by first describing an $\Omega(n)$ lower bound for the deterministic two party communication complexity of $\mathsf{ED}(x, y)$, using a reduction from the equality function which is well known to have an $\Omega(n)$ communication complexity bound. Towards this, fix $\Sigma = [3n] \cup \{a\}$ where a is a special symbol, and fix $y = 1 \circ 2 \circ \cdots \circ 3n$. We divide x into two parts $x = (x_1, x_2)$ such that x_1 is obtained from the string $(1, 2, 4, 5, \cdots, 3i - 2, 3i - 1, \cdots, 3n - 2, 3n - 1)$ by replacing some symbols of the form $3j - 1$ by a, while x_2 is obtained from the string $(2, 3, 5, 6, \cdots, 3i - 1, 3i, \cdots, 3n - 1, 3n)$ by replacing some symbols of the form $3j - 1$ by a. Note that the way we choose (x_1, x_2) ensures that $\mathsf{LCS}(x, y) \geq 2n$ before replacing any symbol by a.

Intuitively, we want to argue that the best way to transform x into y, is to delete a substring at the end of x_1 and a substring at the beginning of x_2, so that the resulted string becomes an increasing subsequence as long as possible. Then, we insert symbols into this string to make it match y except for those a symbols. Finally, we replace the a symbols by substitutions. If this is true then we can finish the argument as follows. Let $T_1, T_2 \subset [n]$ be two subsets with size $t = \Omega(n)$, where for any $i \in \{1, 2\}$, all symbols of the form $3j - 1$ in x_i with $j \in T_i$ are replaced by a. Now if $T_1 = T_2$ then it doesn't matter where we choose to delete the substrings in x_1 and x_2, the number of edit operations is always $3n - 2 + t$ by a direct calculation. On the other hand if $T_1 \neq T_2$ and assume for simplicity that the smallest element they differ is an element in T_2, then there is a way to save one substitution, and the the number of edit operations becomes $3n - 3 + t$.

The key part is now proving our intuition. For this, we consider all possible $r \in [3n]$ such that x_1 is transformed into $y[1 : r]$ and x_2 is transformed into $y[r + 1 : 3n]$, and compute the two edit distances respectively. To analyze the edit distance, we first show by a greedy argument that without loss of generality, we can assume that we apply deletions first, followed by insertions, and substitutions at last. This reduces the edit distance problem to the following problem: for a fixed number of deletions and insertions, what is the best way to minimize the Hamming distance (or maximize the number of agreements of symbols at the same indices) in the end. Now we break the analysis of $\mathsf{ED}(x_1, y[1 : r])$ into two cases. Case 1 is where the number of deletions (say d_d) is large. In this case, the number of insertions (say d_i) must also be large, and we argue that the number of agreements is at most $\mathsf{LCS}(x_1, y[1 : r]) + d_i$. Case 2 is where d_d is small. In this case, d_i must also be small. Now we crucially use the structure of x_1 and y, and argue that symbols in x_1 larger than $3d_i$ (or original index beyond $2d_i$) are guaranteed to be out of agreement. Thus the number of agreements is at most $\mathsf{LCS}(x_1[1 : 2d_i], y[1 : r]) + d_i$. In each case combining the bounds gives us a lower bound on the total number of operations. The situation for x_2 and $y[r + 1 : 3n]$ is completely symmetric and this proves our intuition.

In the above construction, x and y have different lengths ($|x| = 4n$ while $|y| = 3n$). We can fix this by adding a long enough string z with distinct symbols than those in $\{x, y\}$ to the end of both x and y, and then add n symbols of a at the end of z for y. We argue that the best way to do the transformation is to transform x into y, and then insert n symbols of a. To show this, we first argue that at least one symbol in z must be kept, for otherwise the number of operations is already larger than the previous transformation. Then, using a greedy argument we show that the entire z must be kept, and thus the natural transformation is the optimal.

To extend the bound to randomized algorithms, we modify the above construction and reduce from *Set Disjointness* (DIS), which is known to have randomized communication complexity $\Omega(n)$. Given two strings $\alpha, \beta \in \{0,1\}^n$ representing the characteristic vectors of two sets $A, B \subseteq [n]$, $\mathsf{DIS}(\alpha, \beta) = 0$ if and only if $A \cap B \neq \emptyset$, or equivalently, $\exists j \in [n], \alpha_j = \beta_j = 1$. For the reduction, we first create two new strings $\alpha', \beta' \in \{0,1\}^{2n}$ which are "balanced" versions of α, β. Formally, $\forall j \in [n], \alpha'_{2j-1} = \alpha_j$ and $\alpha'_{2j} = 1 - \alpha_j$. We create β' slightly differently, i.e., $\forall j \in [n], \beta'_{2j-1} = 1 - \beta_j$ and $\beta'_{2j} = \beta_j$. Now both α' and β' have n 1's, we can use them as the characteristic vectors of the two sets T_1, T_2 in the previous construction. A similar argument now leads to the bound for randomized algorithms.

Longest common subsequence. Our lower bounds for randomized algorithms computing LCS exactly are obtained by a similar and simpler reduction from DIS: we still fix y to be an increasing sequence of length $8n$ and divide y evenly into $4n$ blocks of constant size. Now x_1 consists of the blocks with an odd index, while x_2 consists of the blocks with an even index. Thus x is a permutation of y. Next, from $\alpha, \beta \in \{0,1\}^n$ we create $\alpha', \beta' \in \{0,1\}^{2n}$ in a slightly different way and use α', β' to modify the $2n$ blocks in x_1 and x_2 respectively. If a bit is 1 then we arrange the corresponding block in the increasing order, otherwise we arrange the corresponding block in the decreasing order. A similar argument as before now gives the desired $\Omega(n)$ bound. We note that [28] has similar results for LIS by reducing from DIS. However, our reduction and analysis are different from theirs. Thus we can handle LCS, and even the harder case where x is a permutation of y.

We now turn to LCS over a small alphabet. To illustrate our ideas, let's first consider $\Sigma = \{0,1\}$ and choose $y = 0^{n/2}1^{n/2}$. It is easy to see that $\mathsf{LCS}(x,y) = \mathsf{LNST}(x, n/2)$. We now represent each string $x \in \{0,1\}^n$ as follows: at any index $i \in [n] \cup \{0\}$, we record a pair (p,q) where $p = \mathsf{min}(\text{the number of 0's in } x[1:i], n/2)$ and $q = \mathsf{min}(\text{the number of 1's in } x[i+1:n], n/2)$. Thus, if we read x from left to right, then upon reading a 0, p may increase by 1 and q does not change; while upon reading a 1, p does not change and q may decrease by 1. Hence if we use the horizontal axis to stand for p and the vertical axis to stand for q, then these points (p,q) form a polygonal chain. We call $p+q$ the *value* at point (p,q) and it is easy to see that $\mathsf{LCS}(x,y)$ must be the value of an endpoint of some chain segment.

Using the above representation, we now fix $\Sigma = \{0,1,2\}$ and choose $y = 0^{n/3}1^{n/3}2^{n/3}$, so $\mathsf{LCS}(x,y) = \mathsf{LNST}(x, n/3)$. We let $x = (x_1, x_2)$ such that $x_1 \in \{0,1\}^{n/2}$ and $x_2 \in \{1,2\}^{n/2}$. Since any common subsequence between x and y must be of the form $0^a1^b2^c$ it suffices to consider common subsequence between x_1 and $0^{n/3}1^{n/3}$, and that between x_2 and $1^{n/3}2^{n/3}$, and combine them together. Towards that, we impose the following properties on x_1, x_2: (1) The number of 0's, 1's, and 2's in each string is at most $n/3$; (2) In the polygonal chain representation of each string, the values of the endpoints strictly increase when the number of 1's increases; and (3) For any endpoint in x_1 where the number of 1's is some r, there is a corresponding endpoint in x_2 where the number of 1's is $n/3 - r$, and the values of these two endpoints sum up to a fixed number $t = \Omega(n)$. Note that property (2) implies that $\mathsf{LCS}(x,y)$ must be the sum of the values of an endpoint in x_1 where the number of 1's is some r, and an endpoint in x_2 where the number of 1's is $n/3 - r$, while property (3) implies that for any string x_1, there is a unique corresponding string x_2, and $\mathsf{LCS}(x,y) = t$ (regardless of the choice of r).

We show that under these properties, all possible strings $x = (x_1, x_2)$ form a set S with $|S| = 2^{\Omega(n)}$, and this set gives a *fooling set* for the two party communication problem of computing $\mathsf{LCS}(x, y)$. Indeed, for any $x = (x_1, x_2) \in S$, we have $\mathsf{LCS}(x, y) = t$. On the other hand, for any $(x_1, x_2) \neq (x'_1, x'_2) \in S$, the values must differ at some point for x_1 and x'_1. Hence by switching, either (x_1, x'_2) or (x'_1, x_2) will have a LCS with y that has length at least $t + 1$. Standard arguments now imply an $\Omega(n)$ communication complexity lower bound. A more careful analysis shows that we can even replace the symbol 2 by 0, thus resulting in a binary alphabet.

The above argument can be easily modified to give a $\Omega(1/\varepsilon)$ bound for $1+\varepsilon$ approximation of LCS when $\varepsilon < 1$, by taking the string length to be some $n' = \Theta(1/\varepsilon)$. To get a better bound, we combine our technique with the technique in [14] and consider the following direct sum problem: we create r copies of strings $\{x^i, i \in [r]\}$ and $\{y^i, i \in [r]\}$ where each copy uses distinct alphabets with size 2. Assume for x^i and y^i the alphabet is $\{a_i, b_i\}$, now x^i again consists of r copies of $(x^i_{j1}, x^i_{j2}), j \in [r]$, where each $x^i_{j\ell} \in \{a_i, b_i\}^{n'/2}$ for $\ell \in [2]$; while y^i consists of r copies $y^i_j = a_i^{n'/3} b_i^{n'/3} a_i^{n'/3}, j \in [r]$. The direct sum problem is to decide between the following two cases for some $t = \Omega(n')$: (1) $\exists i$ such that there are $\Omega(r)$ copies (x^i_{j1}, x^i_{j2}) in x^i with $\mathsf{LCS}((x^i_{j1} \circ x^i_{j2}), y^i_j) \geq t + 1$, and (2) $\forall i$ and $\forall j$, $\mathsf{LCS}((x^i_{j1} \circ x^i_{j2}), y^i_j) \leq t$. We do this by arranging the x^i's row by row into an $r \times 2r$ matrix (each entry is a length $n'/2$ string) and letting x be the concatenation of the *columns*. We call these strings the *contents* of the matrix, and let y be the concatenation of the y^i's. Now intuitively, case (1) and case (2) correspond to deciding whether $\mathsf{LCS}(x, y) \geq 2rt + \Omega(r)$ or $\mathsf{LCS}(x, y) \leq 2rt$, which implies a $1 + \Omega(1/t) = 1 + \varepsilon$ approximation. The lower bound follows by analyzing the $2r$-party communication complexity of this problem, where each party holds a column of the matrix.

However, unlike the constructions in [16, 14] which are relatively easy to analyze because all symbols in x (respectively y) are *distinct*, the repeated symbols in our construction make the analysis of LCS much more complicated (we can also use distinct symbols but that will only give us a bound of $\frac{\sqrt{|\Sigma|}}{\varepsilon}$ instead of $\frac{|\Sigma|}{\varepsilon}$). To ensure that the LCS is to match each (x^i_{j1}, x^i_{j2}) to the corresponding y^i_j, we use another r symbols $\{c_i, i \in [r]\}$ and add *buffers* of large size (e.g., size n') between adjacent copies of (x^i_{j1}, x^i_{j2}). We do the same thing for y^i_j correspondingly. Moreover, it turns out we need to arrange the buffers carefully to avoid unwanted issues: in each row x^i, between each copy of (x^i_{j1}, x^i_{j2}) we use a buffer of new symbol. Thus the buffers added to each row x^i are $c_1^{n'}, c_2^{n'}, \cdots, c_r^{n'}$ sequentially and this is the same for every row. That is, in each row the contents use the same alphabet $\{a_i, b_i\}$ but the buffers use different alphabets $\{c_i, i \in [r]\}$. Now we have a $r \times 3r$ matrix and we again let x be the concatenation of the *columns* while let y be the concatenation of the y^i's. Note that we are using an alphabet of size $|\Sigma| = 3r$. We use a careful analysis to argue that case (1) and case (2) now correspond to deciding whether $\mathsf{LCS}(x, y) \geq 2rn' + rt + \Omega(r)$ or $\mathsf{LCS}(x, y) \leq 2rn' + rt$, which implies a $1 + \varepsilon$ approximation. The lower bound follows by analyzing the $3r$-party communication complexity of this problem, and we show a lower bound of $\Omega(r/\varepsilon) = \Omega(|\Sigma|/\varepsilon)$ by generalizing our previous fooling set construction to the multi-party case, where we use a good error correcting code to create the $\Omega(r)$ gap.

The above technique works for $\varepsilon < 1$. For the case of $\varepsilon \geq 1$ our bound for LCS can be derived directly from our bound for LIS, which we describe next.

Longest increasing/non-decreasing subsequence. Our $\Omega(|\Sigma|)$ lower bound over small alphabet is achieved by modifying the construction in [14] and providing a better analysis. Similar as before, we consider a matrix $B \in \{0, 1\}^{\frac{r}{c} \times r}$ where c is a large constant and $r = |\Sigma|$. We now consider the r-party communication problem where each party holds one column of

B, and the problem is to decide between the following two cases for a large enough constant l: (1) for each row in B, there are at least l 0's between any two 1's, and (2) there exists a row in B which has more than αr 1's, where $\alpha \in (1/2, 1)$ is a constant. We can use a similar argument as in [14] to show that the total communication complexity of this problem is $\Omega(r^2)$ and hence at least one party needs $\Omega(r)$. The difference is that [14] sets $l = 1$ while we need to pick l to be a larger constant to handle the case $\varepsilon \geq 1$. For this we use the Lovász Local Lemma with a probabilistic argument to show the existence of a large fooling set. To reduce to LIS, we define another matrix $\tilde{B}$ such that $\tilde{B}_{i,j} = (i-1)\frac{r}{c} + j$ if $B_{i,j} = 1$ and $\tilde{B}_{i,j} = 0$ otherwise. Now let x be the concatenation of all columns of $\tilde{B}$. We show that case (2) implies $\mathsf{LIS}(x) \geq \alpha r$ and case (1) implies $\mathsf{LIS}(x) \leq (1/c + 1/l)r$. This implies a $1 + \varepsilon$ approximation for any constant $\varepsilon > 0$ by setting c and l appropriately.

The construction is slightly different for LNS. This is because if we keep the 0's in $\tilde{B}$, they will already form a very long non-decreasing subsequence and we will not get any gap. Thus, we now let the matrix B have size $r \times cr$ where c can be any constant. We replace all 0's in column i with a symbol b_i for $i \in [cr]$, such that $b_1 > b_2 > \cdots > b_{cr}$. Similarly we replace all 1's in row j with a symbol a_j for $j \in [r]$, such that $a_1 < a_2 < \cdots < a_r$. Also, we let $a_1 > b_1$. We can show that the two cases now correspond to $\mathsf{LNS}(x) > \alpha cr$ and $\mathsf{LNS}(x) \leq (2 + c/l)r$.

We further prove an $\Omega(|\Sigma| \log(1/\varepsilon))$ lower bound for $1 + \varepsilon$ approximation of LNS when $\varepsilon < 1$. This is similar to our previous construction for LCS, except we don't need buffers here, and we only need to record the number of some symbols. More specifically, let $l = \Theta(1/\varepsilon)$ and S be the set of all strings $x = (x_1, x_2)$ over alphabet $\{a, b\}$ with length $2l$ such that $x_1 = a^{\frac{3}{4}l+t}b^{\frac{1}{4}l-t}$ and $x_2 = a^{\frac{3}{4}l-t}b^{\frac{1}{4}l+t}$ for any $t \in [\frac{l}{4}]$. Thus S has size $\frac{l}{4} = \Omega(1/\varepsilon)$ and $\forall x \in S$, the number of a's in x is exactly $\frac{3}{2}l$. Further, for any $(x_1, x_2) \neq (x'_1, x'_2) \in S$, either (x_1, x'_2) or (x'_1, x_2) has more than $\frac{3}{2}l$ a's. We now consider the $r \times 2r$ matrix where each row i consists of $\{(x^i_{j1}, x^i_{j2}), j \in [r]\}$ such that each $x^i_{j\ell}$ has length l for $\ell \in [2]$, and for the same row i all $\{(x^i_{j1}, x^i_{j2})\}$ use the same alphabet $\{a_i, b_i\}$ while for different rows the alphabets are disjoint. To make sure the LNS of the concatenation of the columns is roughly the sum of the number of a_i's, we require that $b_r < b_{r-1} < \cdots < b_1 < a_1 < a_2 < \cdots < a_r$. Now we analyze the $2r$ party communication problem of deciding whether the concatenation of the columns has $\mathsf{LNS} \geq crl + \Omega(r)$ or $\mathsf{LNS} \leq crl$ for some constant c, which implies a $1 + \varepsilon$ approximation. The lower bound is again achieved by generalizing the set S to a fooling set for the $2r$ party communication problem using an error correcting code based approach.

In Theorem 6, we give three lower bounds for LNST. The first two lower bounds are adapted from our lower bounds for LIS and LNS, while the last lower bound is adapted from our lower bound for LCS by ensuring all symbols in different rows or columns of the matrix there are different.

Improved algorithms. We defer the technique overview of our improved algorithms to Section B.

1.3 Open Problems

Our work leaves a plethora of intriguing open problems. The main one is to close the gap between our lower bounds and the upper bounds of known algorithms, especially for the case of small alphabets and large (say constant) approximation. We believe that in this case it is possible to improve both the lower bounds and the upper bounds. Another interesting problem is to completely characterize the space complexity of LNST.

2 Organization of the paper

The rest of the paper is organized as follows. In Section 3, we give a formal description of the problems we study. We then present our lower bounds for edit distance in Section 4 and lower bounds for LCS in Section 5. In the appendix, we give our lower bounds for LIS, LNS, LNST, and an algorithm for LNST in Section A. In Section B, we present our improved algorithms for asymmetric streaming edit distance and LCS. Finally in Section C, we present a proof for the strong linear lower bound for approximating ED in the standard streaming model. Due to the page limit, some of the proofs are deferred to the full version.

3 Preliminaries

We use the following conventional notations. Let $x \in \Sigma^n$ be a string of length n over alphabet Σ. By $|x|$, we mean the length of x. We denote the i-th character of x by x_i and the substring from the i-th character to the j-th character by $x[i:j]$. We denote the concatenation of two strings x and y by $x \circ y$. By $[n]$, we mean the set of positive integers no larger than n.

Edit Distance. The *edit distance* (or *Levenshtein distance*) between two strings $x, y \in \Sigma^*$, denoted by $\mathsf{ED}(x,y)$, is the smallest number of edit operations (insertion, deletion, and substitution) needed to transform one into another.

Longest Common Subsequence. We say the string $s \in \Sigma^t$ is a *subsequence* of $x \in \Sigma^n$ if there exists indices $1 \le i_1 < i_2 < \cdots < i_t \le n$ such that $s = x_{i_1} x_{i_2} \cdots x_{i_t}$. A string s is called a *common subsequence* of strings x and y if s is a subsequence of both x and y. Given two strings x and y, we denote the length of the longest common subsequence (LCS) of x and y by $\mathsf{LCS}(x,y)$.

Longest Increasing Subsequence. In the longest increasing subsequence problem, we assume there is a given total order on the alphabet set Σ. We say the string $s \in \Sigma^t$ is an *increasing subsequence* of $x \in \Sigma^n$ if there exists indices $1 \le i_1 < i_2 < \cdots < i_t \le n$ such that $s = x_{i_1} x_{i_2} \cdots x_{i_t}$ and $x_{i_1} < x_{i_2} < \cdots < x_{i_t}$. We denote the length of the longest increasing subsequence (LIS) of string x by $\mathsf{LIS}(x)$.

Longest Non-decreasing Subsequence. The longest non-decreasing subsequence is a variant of the longest increasing problem. The difference is that in a non-decreasing subsequence $s = x_{i_1} x_{i_2} \cdots x_{i_t}$, we only require $x_{i_1} \le x_{i_2} \le \cdots \le x_{i_t}$.

4 Lower Bounds for Edit Distance

We show a reduction from the Set Disjointness problem (DIS) to computing ED between two strings in the asymmetric streaming model. For this, we define the following two party communication problem between Alice and Bob.

Given an alphabet Σ and three integers n_1, n_2, n_3. Suppose Alice has a string $x_1 \in \Sigma^{n_1}$ and Bob has a string $x_2 \in \Sigma^{n_1}$. There is another fixed reference string $y \in \Sigma^{n_3}$ that is known to both Alice and Bob. Alice and Bob now tries to compute $\mathsf{ED}((x_1 \circ x_2), y)$.We call this problem $\mathsf{ED}_{cc}(y)$. We prove the following theorem.

▶ **Theorem 10.** *Suppose each input string to* DIS *has length n and let $\Sigma = [6n] \cup \{a\}$. Fix $y = (1, 2, \cdots, 6n)$. Then $R^{1/3}(\mathsf{ED}_{cc}(y)) \ge R^{1/3}(\mathsf{DIS})$.*

To prove this theorem, we first construct the strings x_1, x_2 based on the inputs $\alpha, \beta \in \{0,1\}^n$ to DIS. From α, Alice constructs the string $\alpha' \in \{0,1\}^{2n}$ such that $\forall j \in [n], \alpha'_{2j-1} = \alpha_j$ and $\alpha'_{2j} = 1 - \alpha_j$. Similarly, from β, Bob constructs the string $\beta' \in \{0,1\}^{2n}$ such that

$\forall j \in [n], \beta'_{2j-1} = 1 - \beta_j$ and $\beta'_{2j} = \beta_j$. Now Alice lets x_1 be a modification from the string $(1, 2, 4, 5, \cdots, 3i-2, 3i-1, \cdots, 6n-2, 6n-1)$ such that $\forall j \in [2n]$, if $\alpha'_j = 0$ then the symbol $3j-1$ (at index $2j$) is replaced by a. Similarly, Bob lets x_2 be a modification from the string $(2, 3, 5, 6, \cdots, 3i-1, 3i, \cdots, 6n-1, 6n)$ such that $\forall j \in [2n]$, if $\beta'_j = 0$ then the symbol $3j-1$ (at index $2j-1$) is replaced by a.

Given the construction, we have the following lemma.

▶ **Lemma 11.** *If* $\mathsf{DIS}(\alpha, \beta) = 1$ *then* $\mathsf{ED}((x_1 \circ x_2), y) \geq 7n - 2$.

To prove the lemma we observe that in a series of edit operations that transforms (x_1, x_2) to y, there exists an index $r \in [6n]$ s.t. x_1 is transformed into $[1 : r]$ and x_2 is transformed into $[r+1 : n]$. We analyze the edit distance in each part. We first have the following claim:

▷ Claim 12. For any two strings u and v, there is a sequence of optimal edit operations (insertion/deletion/substitution) that transforms u to v, where all deletions happen first, followed by all insertions, and all substitutions happen at the end of the operations.

We defer the proof of this claim to the full version. For any i, let $\Gamma_1(i)$ denote the number of a symbols up to index $2i$ in x_1. Note that $\Gamma_1(i)$ is equal to the number of 0's in $\alpha'[1 : i]$. We have the following lemma.

▶ **Lemma 13.** *For any* $p \in [n]$, *let* $r = 3p - q$ *where* $0 \leq q \leq 2$, *then* $\mathsf{ED}(x_1, [1 : r]) = 4n - p - q + \Gamma_1(p)$ *if* $q = 0, 1$ *and* $\mathsf{ED}(x_1, [1 : r]) = 4n - p + \Gamma_1(p-1)$ *if* $q = 2$.

Proof. By Claim 12 we can first consider deletions and insertions, and then compute the Hamming distance after these operations (for substitutions).

We consider the three different cases of q. Let the number of insertions be d_i and the number of deletions be d_d. Note that $d_i - d_d = r - 4n$. We define the number of agreements between two strings to be the number of positions where the two corresponding symbols are equal.

The case of $q = 0$ and $q = 1$. Here again we have two cases.

Case (a): $d_d \geq 4n - 2p$. In this case, notice that the LCS after the operations between x_1 and y is at most the original $\mathsf{LCS}(x_1, y) = 2p - \Gamma_1(p)$. With d_i insertions, the number of agreements can be at most $\mathsf{LCS}(x_1, y) + d_i = 2p - \Gamma_1(p) + d_i$, thus the Hamming distance at the end is at least $r - 2p + \Gamma_1(p) - d_i$. Therefore, in this case the number of edit operations is at least $d_i + d_d + r - 2p + \Gamma_1(p) - d_i \geq 4n - p - q + \Gamma_1(p)$, and the equality is achieved when $d_d = 4n - 2p$.

Case (b): $d_d < 4n - 2p$. In this case, notice that all original symbols in x_1 larger than $3d_i$ (or beyond index $2d_i$ before the insertions) are guaranteed to be out of agreement. Thus the only possible original symbols in x_1 that are in agreement with y after the operations are the symbols with original index at most $2d_i$. Note that the LCS between $x_1[1 : 2d_i]$ and y is $2d_i - \Gamma_1(d_i)$. Thus with d_i insertions the number of agreements is at most $3d_i - \Gamma_1(d_i)$, and the Hamming distance at the end is at least $r - 3d_i + \Gamma_1(d_i)$.

Therefore the number of edit operations is at least $d_i + d_d + r - 3d_i + \Gamma_1(d_i) = r - d_i + (d_d - d_i) + \Gamma_1(d_i) = 4n - d_i + \Gamma_1(d_i)$. Now notice that $d_i = d_d + r - 4n < p$ and the quantity $d_i - \Gamma_1(d_i)$ is non-decreasing as d_i increases. Thus the number of edit operations is at least $4n - p + \Gamma_1(p) \geq 4n - p - q + \Gamma_1(p)$.

The other case of q is similar, as follows.

The case of $q = 2$. Here again we have two cases.

Case (a): $d_d \geq 4n - 2p + 1$. In this case, notice that the LCS after the operations between x_1 and y is at most the original $\mathsf{LCS}(x_1, y) = 2(p-1) - \Gamma_1(p-1) + 1 = 2p - 1 - \Gamma_1(p-1)$. With d_i insertions, the number of agreements can be at most $\mathsf{LCS}(x_1, y) + d_i = 2p - 1 - \Gamma_1(p-1) + d_i$, thus the Hamming distance at the end is at least $r - 2p + 1 + \Gamma_1(p-1) - d_i$. Therefore, in this case the number of edit operations is at least $d_i + d_d + r - 2p + 1 + \Gamma_1(p-1) - d_i \geq 4n - p + \Gamma_1(p-1)$, and the equality is achieved when $d_d = 4n - 2p + 1$.

Case (b): $d_d \leq 4n - 2p$. In this case, notice that all original symbols in x_1 larger than $3d_i$ (or beyond index $2d_i$ before the insertions) are guaranteed to be out of agreement. Thus the only possible original symbols in x_1 that are in agreement with y after the operations are the symbols with original index at most $2d_i$. Note that the LCS between $x[1 : 2d_i]$ and y is $2d_i - \Gamma_1(d_i)$. Thus with d_i insertions the number of agreements is at most $3d_i - \Gamma_1(d_i)$, and the Hamming distance at the end is at least $r - 3d_i + \Gamma_1(d_i)$.

Therefore the number of edit operations is at least $d_i + d_d + r - 3d_i + \Gamma_1(d_i) = r - d_i + (d_d - d_i) + \Gamma_1(d_i) = 4n - d_i + \Gamma_1(d_i)$. Now notice that $d_i = d_d + r - 4n < p - 1$ and the quantity $d_i - \Gamma_1(d_i)$ is non-decreasing as d_i increases. Thus the number of edit operations is at least $4n - (p-1) + \Gamma_1(p-1) > 4n - p + \Gamma_1(p-1)$. ◀

We can now prove a similar lemma for x_2. For any i, let $\Gamma_2(i)$ denote the number of a symbols from index $2i + 1$ to $4n$ in x_2. Note that $\Gamma_2(i)$ is equal to the number of 0's in $\beta'[i + 1 : 2n]$.

▶ **Lemma 14.** *Let $r = 3p + q$ where $0 \leq q \leq 2$, then $\mathsf{ED}(x_2, [r + 1 : 6n]) = 2n + p - q + \Gamma_2(p)$ if $q = 0, 1$ and $\mathsf{ED}(x_2, [r + 1 : 6n]) = 2n + p + \Gamma_2(p + 1)$ if $q = 2$.*

Proof. We can reduce to Lemma 13. To do this, use $6n + 1$ to minus every symbol in x_2 and in $[r + 1 : 6n]$, while keeping all the a symbols unchanged. Now, reading both strings from right to left, x_2 becomes the string $\overline{x_2} = 1, 2, \cdots, 3i - 2, 3i - 1, \cdots, 6n - 2, 6n - 1$ with some symbols of the form $3j - 1$ replaced by a's. Similarly $[r + 1 : 6n]$ becomes $[1 : 6n - r]$ where $6n - r = 3(2n - p) - q$.

If we regard $\overline{x_2}$ as x_1 as in Lemma 13 and define $\Gamma_1(i)$ as in that lemma, we can see that $\Gamma_1(i) = \Gamma_2(2n - i)$.

Now the lemma basically follows from Lemma 13. In the case of $q = 0, 1$, we have

$$\mathsf{ED}(x_2, [r+1 : 6n]) = \mathsf{ED}(\overline{x'}, [1 : 6n-r]) = 4n-(2n-p)-q+\Gamma_1(2n-p) = 2n+p-q+\Gamma_2(p).$$

In the case of $q = 2$, we have

$$\mathsf{ED}(x_2, [r+1 : 6n]) = \mathsf{ED}(\overline{x'}, [1 : 6n-r]) = 4n-(2n-p)+\Gamma_1(2n-p-1) = 2n+p+\Gamma_2(p+1). ◀$$

We can now prove Lemma 11.

Proof of Lemma 11. We show that for any $r \in [6n]$, $\mathsf{ED}(x_1, [1 : r]) + \mathsf{ED}(x_2, [r + 1 : 6n]) \geq 7n - 2$. First we have the following claim.

▷ Claim 15. If $\mathsf{DIS}(\alpha, \beta) = 1$, then for any $i \in [2n]$, we have $\Gamma_1(i) + \Gamma_2(i) \geq n$.

To see this, note that when i is even, we have $\Gamma_1(i) = i/2$ and $\Gamma_1(i) = n - i/2$ so $\Gamma_1(i) + \Gamma_2(i) = n$. Now consider the case of i being odd and let $i = 2j - 1$ for some $j \in [2n]$. We know $\Gamma_1(i - 1) = (i - 1)/2 = j - 1$ and $\Gamma_2(i + 1) = n - (i + 1)/2 = n - j$, so we only need to look at $x_1[2i - 1, 2i]$ and $x_2[2i + 1, 2i + 2]$ and count the number of symbols a's in them. If the number of a's is at least 1, then we are done.

The only possible situation where the number of a's is 0 is that $\alpha'_i = \beta'_{i+1} = 1$ which means $\alpha_j = \beta_j = 1$ and this contradicts the fact that $\mathsf{DIS}(\alpha, \beta) = 1$.

We now have the following cases.

Case (a): $r = 3p$. In this case, by Lemma 13 and Lemma 14 we have $\mathsf{ED}(x_1, [1 : r]) = 4n - p + \Gamma_1(p)$ and $\mathsf{ED}(x_2, [r + 1 : 6n]) = 2n + p + \Gamma_2(p)$. Thus we have $\mathsf{ED}(x_1, [1 : r]) + \mathsf{ED}(x_2, [r + 1 : 6n]) = 6n + n = 7n$.

Case (b): $r = 3p - 1 = 3(p - 1) + 2$. In this case, by Lemma 13 and Lemma 14 we have $\mathsf{ED}(x_1, [1 : r]) = 4n - p - 1 + \Gamma_1(p)$ and $\mathsf{ED}(x_2, [r + 1 : 6n]) = 2n + (p - 1) + \Gamma_2(p)$, thus we have $\mathsf{ED}(x_1, [1 : r]) + \mathsf{ED}(x_2, [r + 1 : 6n]) = 6n - 2 + n = 7n - 2$.

Case (c): $r = 3p - 2 = 3(p - 1) + 1$. In this case, by Lemma 13 and Lemma 14 we have $\mathsf{ED}(x_1, [1 : r]) = 4n - p + \Gamma_1(p - 1)$ and $\mathsf{ED}(x_2, [r + 1 : 6n]) = 2n + (p - 1) - 1 + \Gamma_2(p - 1)$, thus we have $\mathsf{ED}(x_1, [1 : r]) + \mathsf{ED}(x_2, [r + 1 : 6n]) = 6n - 2 + n = 7n - 2$. ◀

We now prove Theorem 10.

Proof of Theorem 10. We begin by upper bounding $\mathsf{ED}((x_1 \circ x_2), y)$ when $\mathsf{DIS}(\alpha, \beta) = 0$.

▷ Claim 16. If $\mathsf{DIS}(\alpha, \beta) = 0$ then $\mathsf{ED}((x_1 \circ x_2), y) \leq 7n - 3$.

To see this, note that if $\mathsf{DIS}(\alpha, \beta) = 0$ then there exists a $j \in [n]$ such that $\alpha_j = \beta_j = 1$. Thus $\alpha'_{2j-1} = 1$, $\beta'_{2j-1} = 0$ and $\alpha'_{2j} = 0$, $\beta'_{2j} = 1$. Note that the number of 0's in $\alpha'[1 : 2j - 1]$ is $j - 1$ and thus $\Gamma_1(2j - 1) = j - 1$. Similarly the number of 0's in $\beta'[2j : 2n]$ is $n - j$ and thus $\Gamma_2(2j - 1) = n - j$. To transform (x_1, x_2) to y, we choose $r = 6j - 2$, transform x_1 to $y[1 : r]$, and transform x_2 to $y[r + 1 : 6n]$.

By Lemma 13 and Lemma 14 we have $\mathsf{ED}(x_1, [1 : r]) = 4n - 2j + \Gamma_1(2j - 1)$ and $\mathsf{ED}(x_2, [r + 1 : 6n]) = 2n + (2j - 1) - 1 + \Gamma_2(2j - 1)$. .Thus we have $\mathsf{ED}(x_1, [1 : r]) + \mathsf{ED}(x_2, [r + 1 : 6n]) = 6n - 2 + \Gamma_1(2j - 1) + \Gamma_2(2j - 1) = 6n - 2 + n - 1 = 7n - 3$. Therefore $\mathsf{ED}((x_1, x_2), y) \leq 7n - 3$.

Therefore, in the case of $\mathsf{DIS}(\alpha, \beta) = 1$, we have $\mathsf{ED}((x_1 \circ x_2), y) \geq 7n - 2$ while in the case of $\mathsf{DIS}(\alpha, \beta) = 0$, we have $\mathsf{ED}((x_1 \circ x_2), y) \leq 7n - 3$. Thus any protocol that solves $\mathsf{ED}_{cc}(y)$ can also solve DIS, hence the theorem follows. ◀

In the proof of Theorem 10, the two strings $x = (x_1 \circ x_2)$ and y have different lengths, however we can extend it to the case where the two strings have the same length and prove the following theorem.

▶ **Theorem 17.** *Suppose each input string to* DIS *has length* n *and let* $\Sigma = [16n] \cup \{a\}$. *Fix* $\tilde{y} = (1, 2, \cdots, 16n, a^{2n})$, *let* $\tilde{x}_1 \in \Sigma^{4n}$ *and* $\tilde{x}_2 \in \Sigma^{14n}$. *Define* $\mathsf{ED}_{cc}(\tilde{y})$ *as the two party communication problem of computing* $\mathsf{ED}((\tilde{x}_1 \circ \tilde{x}_2), \tilde{y})$. *Then* $R^{1/3}(\mathsf{ED}_{cc}(\tilde{y})) \geq R^{1/3}(\mathsf{DIS})$.

We defer the proof of Theorem 17 to the full version. From Theorem 17 we immediately have the following theorem.

▶ **Theorem 18.** *Any* R*-pass randomized algorithm in the asymmetric streaming model that computes* $\mathsf{ED}(x, y)$ *exactly between two strings* x, y *of length* n *with success probability at least* $2/3$ *must use space at least* $\Omega(n/R)$.

We can generalize the theorem to the case of deciding if $\mathsf{ED}(x, y)$ is at least a given number k. We have the following theorem. The proof is deferred to the full version.

▶ **Theorem 19** (Restatement of Theorem 1). *There is a constant* $c > 1$ *such that for any* $k, n \in \mathbb{N}$ *with* $n \geq ck$, *given an alphabet* Σ, *any* R*-pass randomized algorithm in the asymmetric streaming model that decides if* $\mathsf{ED}(x, y) \geq k$ *between two strings* $x, y \in \Sigma^n$ *with success probability at least* $2/3$ *must use space at least* $\Omega(\mathsf{min}(k, |\Sigma|)/R)$.

For $0 < \varepsilon < 1$, by taking $k = 1/\varepsilon$ we also get the following corollary:

▶ **Corollary 20.** *Given an alphabet Σ, for any $0 < \varepsilon < 1$, any R-pass randomized algorithm in the asymmetric streaming model that achieves a $1+\varepsilon$ approximation of $\mathsf{ED}(x, y)$ between two strings $x, y \in \Sigma^n$ with success probability at least $2/3$ must use space at least $\Omega(\min(1/\varepsilon, |\Sigma|)/R)$.*

5 Lower Bounds for LCS

In this section, we study the space lower bounds for asymmetric streaming LCS.

5.1 Exact computation

5.1.1 Binary alphabet, deterministic algorithm

In this section, without loss of generality, we assume n can be diveded by 60 and let $l = \frac{n}{30} - 1$. We assume the alphabet is $\Sigma = \{a, b\}$. Consider strings x of the form

$$x = b^{10}a^{s_1}b^{10}a^{s_2}b^{10} \cdots b^{10}a^{s_l}b^{10}. \tag{1}$$

That is, x contains l blocks of consecutive a symbols. Between each block of a symbols, we insert 10 b's and we also add 10 b's to the front, and the end of x. $s_1, \dots, s_l$ are l integers such that

$$\sum_{i=1}^{l} s_i = \frac{n}{6} + 5, \tag{2}$$

$$1 \le s_i \le 9, \ \forall i \in [l]. \tag{3}$$

Thus, the length of x is $\sum_{i=1}^{l} \frac{n}{6} + 5 + 10(l+1) = \frac{n}{2} + 5$ and it contains exactly $\frac{n}{3}$ b's.

Let S be the set of all $x \in \{a, b\}^{\frac{n}{2}+5}$ of form 1 that satisfying equations 2, 3. For each string $x \in S$, we can define a string $f(x) \in \{a, b\}^{\frac{n}{2}-5}$ as following. Assume $x = b^{10}a^{s_1}b^{10}a^{s_2}b^{10} \cdots b^{10}a^{s_l}b^{10}$, we set $f(x) = a^{s_1}b^{10}a^{s_2}b^{10} \cdots b^{10}a^{s_l}b^{10}$. That is, $f(x)$ simply removed the first 10 b's of x. We denote $\bar{S} = \{f(x)|x \in S\}$.

▷ Claim 21. $|S| = |\bar{S}| = 2^{\Omega(n)}$.

Proof. Notice that for $x^1, x^2 \in S$, if $x^1 \neq x^2$, then $f(x^1) \neq f(x^2)$. We have $|S| = |\bar{S}|$.

The size of S equals to the number of choices of l integers $s_1, s_2, \dots, s_l$ that satisfies 2 and 3. For an lower bound of $|S|$, we can pick $\frac{n}{60}$ of the integers to be 9, and set the remaining to be 1 or 2. Thus the number of such choices is at least $\binom{l}{\frac{n}{60}} = \binom{\frac{n}{30}-1}{\frac{n}{60}} = 2^{\Omega(n)}$. ◁

Given the construction of set S, we have the following lemma. We defer the proof to the full version.

▶ **Lemma 22.** *Let $y = a^{n/3}b^{n/3}a^{n/3}$. For every $x \in S$,*

$$\mathsf{LCS}(x \circ f(x), y) = \frac{n}{2} + 5.$$

For any two distinct $x^1, x^2 \in S$,

$$\max\{\mathsf{LCS}(x^1 \circ f(x^2), y), \mathsf{LCS}(x^2 \circ f(x^1), y)\} > \frac{n}{2} + 5.$$

The above lemma implies the following lower bound.

▶ **Lemma 23.** *In the asymmetric streaming model, any deterministic protocol that computes* $\mathsf{LCS}(x, y)$ *for any* $x, y \in \{0, 1\}^n$*, in* R *passes of* x *needs* $\Omega(n/R)$ *space.*

Proof. Consider a two party game where player 1 holds a string $x^1 \in S$ and player 2 holds a string $x^2 \in S$. The goal is to verify whether $x^1 = x^2$. It is known that the total communication complexity of testing the equality of two elements from set S is $\Omega(\log |S|)$, see [21] for example. We can reduce this to computing the length of LCS. To see this, we first compute $\mathsf{LCS}(x^1 \circ f(x^2), y)$ and $\mathsf{LCS}(x^2 \circ f(x^1), y)$ with $y = a^{n/3}b^{n/3}a^{n/3}$. By lemma 22, if both $\mathsf{LCS}(x^1 \circ f(x^2), y) = \mathsf{LCS}(x^2 \circ f(x), y) = \frac{n}{2} + 5$, we know $x^1 = x^2$, otherwise, $x^1 \neq x^2$. Here, y is known to both parties.

The above reduction shows the total communication complexity of this game is $\Omega(n)$ since $|S| = 2^{\Omega(n)}$. If we only allow R rounds of communication, the size of the longest message sent by the players is $\Omega(n/R)$. Thus, in the asymmetric model, any protocol that computes $\mathsf{LCS}(x, y)$ in R passes of x needs $\Omega(n/R)$ space. ◀

5.1.2 $\Omega(n)$ size alphabet, randomized algorithm

If the alphabet set Σ has size $\Omega(n)$, then we show there is a space lower bound of $\Omega(n)$ for asymmetric streaming algorithms that computes $\mathsf{LCS}(x, y)$ for $x, y \in \Sigma^n$. We have the following theorem.

▶ **Theorem 24** (Restatement of Theorem 3)**.** *There is a constant* $c > 1$ *such that for any* $k, n \in \mathbb{N}$ *with* $n > ck$*, given an alphabet* Σ*, any* R*-pass randomized algorithm in the asymmetric streaming model that decides if* $\mathsf{LCS}(x, y) \geq k$ *between two strings* $x, y \in \Sigma^n$ *with success probability at least* $2/3$ *must use at least* $\Omega\big(\mathsf{min}(k, |\Sigma|)/R\big)$ *space.*

The proof relies on a reduction from set-disjointness problem. We defer the proof of Theorem 3 to the full version.

5.2 Approximation

For deterministic approximation of LCS in the asymmetric streaming model, we have the following lower bound.

▶ **Theorem 25** (Restatement of Theorem 5)**.** *Assume* $\varepsilon > 0$*, and* $\frac{|\Sigma|^2}{\varepsilon} \leq n$ *. In the asymmetric streaming model, any deterministic protocol that computes an* $1+\varepsilon$ *approximation of* $\mathsf{LCS}(x, y)$ *for any* $x, y \in \Sigma^n$*, with constant number of passes of* x *needs* $\Omega(\frac{|\Sigma|}{\varepsilon})$ *space.*

This lower bound is achieved by combining our construction in Section 5.1.1 with the techniques from [14]. We defer the proof to the full version.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for lcs and other sequence similarity measures. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*. IEEE, 2015.

2 Alexandr Andoni, T.S. Jayram, and Mihai Patrascu. Lower bounds for edit distance and product metrics via poincare type inequalities. In *Proceedings of the twenty first annual ACM-SIAM symposium on Discrete algorithms*, pages 184–192, 2010.

3 Alexandr Andoni and Robert Krauthgamer. The computational hardness of estimating edit distance. *SIAM Journal on Discrete Mathematics*, 39(6):2398–2429, 2010.

4 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Foundations of Computer Science (FOCS), 2010 IEEE 51st Annual Symposium on*. IEEE, 2010.

5 Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In *Proceedings of the 61st Annual Symposium on Foundations of Computer Science (FOCS)*, 2020.

6 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing (STOC)*. IEEE, 2015.

7 Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science*, pages 51–60. IEEE, 2016.

8 Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proceedings of the 52nd annual ACM symposium on Theory of computing (STOC)*, 2020.

9 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucky, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *Foundations of Computer Science (FOCS), 2018 IEEE 59th Annual Symposium on*. IEEE, 2019.

10 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Low distortion embedding from edit to hamming distance using coupling. In *Proceedings of the 48th IEEE Annual Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 2016.

11 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for computing edit distance without exploiting suffix trees. *arXiv preprint*, 2016. `arXiv:1607.03718`.

12 Kuan Cheng, Alireza Farhadi, MohammadTaghi Hajiaghayi, Zhengzhong Jin, Xin Li, Aviad Rubinstein, Saeed Seddighin, and Yu Zheng. Streaming and small space approximation algorithms for edit distance and longest common subsequence. In *International Colloquium on Automata, Languages, and Programming*. Springer, 2021.

13 Kuan Cheng, Zhengzhong Jin, Xin Li, and Yu Zheng. Space efficient deterministic approximation of string measures. *arXiv preprint*, 2020. `arXiv:2002.08498`.

14 Funda Ergun and Hossein Jowhari. On distance to monotonicity and longest increasing subsequence of a data stream. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 730–736, 2008.

15 Alireza Farhadi, MohammadTaghi Hajiaghayi, Aviad Rubinstein, and Saeed Seddighin. Streaming with oracle: New streaming algorithms for edit distance and lcs. *arXiv preprint*, 2020. `arXiv:2002.11342`.

16 Anna Gál and Parikshit Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. *SIAM Journal on Computing*, 39(8):3463–3479, 2010.

17 Parikshit Gopalan, TS Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 318–327. Society for Industrial and Applied Mathematics, 2007.

18 MohammadTaghi Hajiaghayi, Masoud Seddighin, Saeed Seddighin, and Xiaorui Sun. Approximating lcs in linear time: beating the $\sqrt{n}$ barrier. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1181–1200. Society for Industrial and Applied Mathematics, 2019.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity. *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

20 Michal Kouckỳ and Michael E Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proceedings of the 52nd annual ACM symposium on Theory of computing (STOC)*, 2020.

21 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge Press, 1997.

22 David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. In *COCOON*, 2005.

23 Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for lcs and lis with truly improved running times. In *Foundations of Computer Science (FOCS), 2019 IEEE 60th Annual Symposium on*. IEEE, 2019.

24 Aviad Rubinstein and Zhao Song. Reducing approximate longest common subsequence to approximate edit distance. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1591–1600. SIAM, 2020.

25 Barna Saha. Fast & space-efficient approximations of language edit distance and RNA folding: An amnesic dynamic programming approach. In *FOCS*, 2017.

26 Michael Saks and C Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1698–1709. SIAM, 2013.

27 Leonard J Schulman and David Zuckerman. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE transactions on information theory*, 45(7):2552–2557, 1999.

28 Xiaoming Sun and David P Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 336–345, 2007.

A Lower Bounds for LIS and LNS

In this section, we introduce our space lower bound for LIS and LNS. We first introduce some definition and related results regarding the multi-party communication model.

We will consider the one-way t-party communication model where t players $P_1, P_2, \ldots, P_t$ each holds input $x_1, x_2, \ldots, x_t$ respectively. The goal is to compute the function $f(x_1, x_2, \ldots, x_t)$. In the one-way communication model, each player speaks in turn and player P_i can only send message to player P_{i+1}. We sometimes consider multiple round of communication. In an R round protocol, during round $r \leq R$, each player speaks in turn P_i sends message to P_{i+1}. At the end of round $r < R$, player P_t sends a message to P_1. At the end of round R, player P_t must output the answer of the protocol.

We define the *total communication complexity* of f in the t-party one-way communication model, denoted by $CC_t^{tot}(f)$, as the minimum number of bits required to be sent by the players in every deterministic communication protocol that always outputs a correct answer. We define $CC_t^{max}(f)$, the maximum communication complexity of f, as the maximum number of bits required to be sent by some player in protocol P, where P ranges over all deterministic protocol that outputs a correct answer. We have $CC_t^{max}(f) \geq \frac{1}{tR} CC_t^{tot}(f)$ where R is the number of rounds.

Let X be a subset of U^t where U is some finite universe and t is an integer. Define the **span** of X by $\mathsf{Span}(X) = \{y \in U^t | \forall\ i \in [t],\ \exists\ x \in X \text{ s. t. } y_i = x_i\}$. The notion k-fooling set introduced in [14] is defined as following.

▶ **Definition 26** (k-fooling set)**.** *Let $f : U^t \to \{0,1\}$ where U is some finite universe. Let $S \subseteq U^t$. For some integer k, we say S is a k-fooling set for f iff $f(x) = 0$ for each $x \in S$ and for each subset S' of S with cardinality k, the span of S' contains a member y such that $f(y) = 1$.*

We have the following.

▶ **Lemma 27** (Fact 4.1 from [14])**.** *Let S be a k-fooling set for f, we have $CC_t^{tot}(f) \geq \log(\frac{|S|}{k-1})$.*

We now present our lower bounds. Consider the following problem. Let $s \in \{0,1\}^t$ be a binary string of length t. For each integer $l \geq 1$, we can define a function $h^{(l)}$ whose domain is a subset of $\{0,1\}^t$. Let $\alpha \in (1/2, 1)$ be some constant. We have following definition

$$h^{(l)}(a) = \begin{cases} 1, & \text{if there are at least } l \text{ zeros between any two nonzero positions in } s. \\ 0, & \text{if } s \text{ contains at least } \alpha t \text{ nonzeros.} \end{cases} \tag{4}$$

We leave $h^{(l)}$ undefined otherwise. Let $B \in \{0,1\}^{s \times t}$ be a matrix and denote the i-th row of B by $R_i(B)$. We can define $g^{(l)}$ as the direct sum of s copies of $h^{(l)}$. Let

$$g^{(l)}(B) = h^{(l)}(R_1(B)) \vee h^{(l)}(R_2(B)) \vee \cdots \vee h^{(l)}(R_s(B)). \tag{5}$$

That is, $g^{(l)}(B) = 1$ if and only if there is some $i \in [s]$ such that $h^{(l)}(R_i(B)) = 1$.

In the following, we consider computing $h^{(l)}$ and $g^{(l)}$ in the t-party one-way communication model. When computing $h^{(l)}(a)$, player P_i holds the i-th element of $a \in \{0,1\}^t$ for $i \in [t]$.In this setting, when computing $g^{(l)}(B)$, player P_i holds the i-th column of matrix B for $i \in [t]$. In the following, we use $CC_t^{tot}(h^{(l)})$ to denote the total communication complexity of $h^{(l)}$ and respectively use $CC_t^{tot}(g^{(l)})$ to denote the total communication complexity of $g^{(l)}$. We also consider multiple rounds of communication and we denote the number of rounds by R. For a more detailed discussion of the multiparty communication model, we refer readers to the full version of this paper.

We can show the following lemma using Lovás Local Lemma.

▶ **Lemma 28.** *For any constant $l \geq 1$, there exists a constant k (depending on l), such that there is a k-fooling set for function $h^{(l)}$ of size c^t for some constant $c > 1$.*

We note that Lemma 4.2 of [14] proved a same result for the case $l = 1$. We defer the proof to the full version.

The following lemma is essentially the same as Lemma 4.3 in [14].

▶ **Lemma 29.** *Let $F \subseteq \{0,1\}^t$ be a k-fooling set for $h^{(l)}$. Then the set of all matrix $B \in \{0,1\}^{s \times t}$ such that $R_i(B) \in F$ is a k^s-fooling set for $g^{(l)}$.*

Combining Lemma 28 and Lemma 29, we have the following.

▶ **Lemma 30.** *$CC_t^{max}(g^{(l)}) = \Omega(s/R)$.*

Proof. By Lemma 28 and Lemma 29, there is a k^s-fooling set for function $g^{(l)}$ of size c^{ts} for some large enough constant k and some constant $c > 1$. By Lemma 27, in the t-party one-way communication model, $CC_t^{tot}(g^{(l)}) = \Omega(\log \frac{c^{ts}}{k^s - 1}) = \Omega(ts)$. Thus, we have $CC_t^{max}(g^{(l)}) \geq \frac{1}{tR} CC_t^{tot}(g^{(l)}) = \Omega(s/R)$. ◀

A.1 Lower bound for streaming LIS over small alphabet

With Lemma 30, we can show the following lower bound.

▶ **Lemma 31.** *For $x \in \Sigma^n$ with $|\Sigma| = O(\sqrt{n})$ and any constant $\varepsilon > 0$, any deterministic algorithm that makes R passes of x and outputs a $(1+\varepsilon)$-approximation of $\mathsf{LIS}(x)$ requires $\Omega(|\Sigma|/R)$ space.*

Proof sketch of Lemma 31. We assume the alphabet set $\Sigma = \{0, 1, \ldots, 2r\}$ which has size $|\Sigma| = 2r + 1$. Let c be a large constant and assume r can be divided by c for similicity. We set $s = \frac{r}{c}$ and $t = r$. Consider a matrix B of size $s \times t$. We denote the element on i-th row and j-th column by $B_{i,j}$. ALso, we require that $B_{i,j}$ is either $(i-1)\frac{r}{c}+j$ or 0. For each row of B, say $R_i(B)$, either there are at least l 0's between any two nonzeros or it has more than αr nonzeros. We let $\tilde{B} \in \{0,1\}^{s \times r}$ be a binary matrix such that $\tilde{B}_{i,j} = 1$ if $B_{i,j} \neq 0$ and $\tilde{B}_{i,j} = 0$ if $B_{i,j} = 0$ for $(i,j) \in [s] \times [r]$.

Without loss of generality, we can view any row or any column in B as a string. More specifically, let $R_i(B) = B_{i,1}B_{i,2}\dots B_{i,r}$ for $i \in [s]$, and $C_i(B) = B_{1,i}B_{2,i}\dots B_{s,i}$ for $i \in [r]$. We let $\sigma(B) = C_1(B) \circ C_2(B) \circ \cdots \circ C_r(B)$. Thus, $\sigma(B)$ is a string of length sr. For convenience, we denote $\sigma = \sigma(B)$. Here, we required the length of $\sigma = r^2/c \leq n$. If $|\sigma| < n$, we can pad σ with 0 symbols to make it has length n. This will not affect the length of the longest increasing subsequence of σ.

We can show that if there is some row of B containing more than αt nonzeros, then $\mathsf{LIS}(\sigma) \geq \alpha r$. If not, then $\mathsf{LIS}(\sigma(B)) \leq (\frac{1}{r} + \frac{1}{c})r$.

Thus, if $g^{(l)}(\tilde{B}) = 0$, we have $\mathsf{LIS}(\sigma(B)) \geq \alpha r$. And if $g^{(l)}(\tilde{B}) = 1$, $\mathsf{LIS}(\sigma(B)) \leq (\frac{1}{c} + \frac{1}{l})r$. Here, c and l can be any large constant up to our choice and $\alpha \in (1/2, 1)$ is fixed. For any $\varepsilon > 0$, we can choose c and l such that $(1+\varepsilon)(\frac{1}{c} + \frac{1}{l}) \leq \alpha$. This gives us a reduction from computing $g^{(l)}(\tilde{B})$ to compute a $(1+\varepsilon)$-approximation of $\mathsf{LIS}(\sigma(B))$.

In the t-party game for computing $g^{(l)}(\tilde{B})$, each player holds one column of $\tilde{B}$. Thus, player P_i also holds $C_i(B)$ since $C_i(B)$ is determined by $C_i(\tilde{B})$. If the t players can compute a $(1+\varepsilon)$ approximation of $\sigma(B)$ in the one-way communication model, we can distinguish the case of $g^{(l)}(\tilde{B}) = 0$ and $g^{(l)}(\tilde{B}) = 1$. Thus, any R passes deterministic streaming algorihtm that approximate LIS within a $1+\varepsilon$ factor requires at least $CC_t^{max}(g^{(l)})$. By Lemma 30, $CC_t^{max}(g^{(l)}) = \Omega(s/R) = \Omega(|\Sigma|/R)$. ◀

A.2 Longest Non-decreasing Subsequence

We can proof a similar space lower bound for approximating the length of longest non-decreasing subsequence in the streaming model. We have the following two lemmas. The proof is deferred to the full version.

▶ **Lemma 32.** *For $x \in \Sigma^n$ with $|\Sigma| = O(\sqrt{n})$ and any constant $\varepsilon > 0$, any deterministic algorithm that makes R passes of x and outputs a $(1+\varepsilon)$-approximation of* $\mathsf{LNS}(x)$ *requires $\Omega(|\Sigma|/R)$ space.*

▶ **Lemma 33.** *Let $x \in \Sigma^n$ and $\varepsilon > 0$ such that $|\Sigma|^2/\varepsilon = O(n)$. Then any deterministic algorithm that makes constant pass of x and outputs a $(1+\varepsilon)$ approximation of* $\mathsf{LNS}(x)$ *takes $\Omega(r \log \frac{1}{\varepsilon})$ space.*

A.3 Longest Non-decreasing Subsequence with Threshold

We also consider a variant of LNS problem we call longest non-decreasing subsequence with threshold (LNST). In this problem, we are given a sequence $x \in \Sigma^n$ and a threshold $t \in [n]$, the longest non-decreasing subsequence with threshold t is the longest non-decreasing subsequence of x such that each symbol appeared in it is repeated at most t times. We denote the length of such a subsequence by $\mathsf{LNST}(x,t)$.

By combining the techniques from the previous sections, we can show Theorem 6. We also presented upper bound for LNST in Theorem 9 by giving a simple algorithm. We omit the algorithm and the formal proofs here.

B Algorithms for Edit Distance and LCS

In this section, we give an informal description of our improved algorithms for edit distance and LCS in the asymmetric streaming model. The formal proofs are deferred to the full version.

Algorithm for edit distance. Our algorithm for edit distance builds on and improves the algorithm in [15, 13]. The key idea of that algorithm is to use triangle inequality. Given a constant δ, the algorithm first divides x evenly into $b = n^\delta$ blocks. Then for each block x^i of x, the algorithm recursively finds an α-approximation of the closest substring to x^i in y. That is, the algorithm finds a substring $y[l_i : r_i]$ and a value d_i such that for any substring $y[l : r]$ of y, $\mathsf{ED}(x^i, y[l_i : r_i]) \le d_i \le \alpha\mathsf{ED}(x^i, y[l : r])$. Let $\tilde{y}$ be the concatenation of $y[l_i : r_i]$ from $i = 1$ to b. Then using triangle inequality, [15] showed that $\mathsf{ED}(y, \tilde{y}) + \sum_{i=1}^{b} d_i$ is a $2\alpha + 1$ approximation of $\mathsf{ED}(x, y)$. The $\tilde{O}(n^\delta)$ space is achieved by recursively applying this idea, which results in a $O(2^{1/\delta})$ approximation.

To further reduce the space complexity, our key observation is that, instead of dividing x into blocks of equal length, we can divide it according to the positions of the edit operations that transform x to y. More specifically, assume we are given a value k with $\mathsf{ED}(x, y) \le k \le c\mathsf{ED}(x, y)$ for some constant c, we show how to design an approximation algorithm using space $\tilde{O}(\sqrt{k})$. Towards this, we can divide x and y each into $\sqrt{k}$ blocks $x = x^1 \circ \cdots \circ x^{\sqrt{k}}$ and $y = y^1 \circ \cdots \circ y^{\sqrt{k}}$ such that $\mathsf{ED}(x^i, y^i) \le \frac{\mathsf{ED}(x,y)}{\sqrt{k}} \le \sqrt{k}$ for any $i \in [\sqrt{k}]$. However, such a partition of x and y is not known to us. Instead, we start from the first position of x and find the largest index l_1 such that $\mathsf{ED}(x[1 : l_1], y[p_1, q_1]) \le \sqrt{k}$ for some substring $y[p_1 : q_1]$ of y. To do this, we start with $l = \sqrt{k}$ and try all substrings of y with length in $[l - \sqrt{k}, l + \sqrt{k}]$. If there is some substring of y within edit distance $\sqrt{k}$ to $x[1 : l]$, we set $l_1 = l$ and store all the edit operations that transform $y[p_1 : q_1]$ to $x[1 : l_1]$ where $y[p_1 : q_1]$ is the substring closest to $x[1 : l_1]$ in edit distance. We continue doing this with $l = l + 1$ until we can not find a substring of y within edit distance $\sqrt{k}$ to $x[1 : l]$.

One problem here is that l can be much larger than $\sqrt{k}$ and we cannot store $x[1 : l]$ with $\tilde{O}(\sqrt{k})$ space. However, since we have stored some substring $y[p_1 : q_1]$ (we only need to store the two indices p_1, q_1) and the at most $\sqrt{k}$ edit operations that transform $y[p_1 : q_1]$ to $x[1 : l - 1]$, we can still query every bit of $x[1 : l]$ using $\tilde{O}(\sqrt{k})$ space.

After we find the largest possible index l_1, we store l_1, (p_1, q_1) and $d_1 = \mathsf{ED}(x[1 : l_1], y[p_1 : q_1])$. We then start from the $(l_1 + 1)$-th position of x and do the same thing again to find the largest l_2 such that there is a substring of y within edit distance $\sqrt{k}$ to $x[l_1 + 1 : l_1 + l_2]$. We continue doing this until we have processed the entire string x. Assume this gives us T pairs of indices (p_i, q_i) and integers l_i, d_i from $i = 1$ to T, we can use $O(T \log n)$ space to store them. We show by induction that $x^1 \circ \cdots \circ x^i$ is a substring of $x[1 : \sum_{j=1}^{i} l_j]$ for $i \in [T - 1]$. Recall that $x = x^1 \circ \cdots \circ x^{\sqrt{k}}$ and each $l_i > 0, i \in [T - 1]$. Thus, the process must end within $\sqrt{k}$ steps and we have $T \le \sqrt{k}$. Then, let $\tilde{y}$ be the concatenation of $y[p_i : q_i]$ from $i = 1$ to T. Using techniques developed in [15], we can show $\mathsf{ED}(y, \tilde{y}) + \sum_{i=1}^{T} d_i$ is a 3 approximation of $\mathsf{ED}(x, y)$. For any small constant $\varepsilon > 0$, we can compute a $1 + \varepsilon$ approximation of $\mathsf{ED}(y, \tilde{y})$ with $\mathsf{polylog}(n)$ space using the algorithm in [13]. This gives us a $3 + \varepsilon$ approximation algorithm with $O(\sqrt{\mathsf{ED}(x, y)}\ \mathsf{polylog}(n))$ space.

Similar to [15], we can use recursion to further reduce the space. Let δ be a small constant and a value $k = \Theta(\mathsf{ED}(x, y))$ be given as before. There is a way to partition x and y each into k^δ blocks such that $\mathsf{ED}(x^i, y^i) \le \frac{\mathsf{ED}(x,y)}{k^\delta} \le k^{1-\delta}$. Now similarly, we want to find the largest index l^0 such that there is a substring of y within edit distance $k^{1-\delta}$ to $x[1 : l^0]$. However naively this would require $\Theta(k^{1-\delta})$ space to compute the edit distance. Thus again we turn to approximation.

We introduce a recursive algorithm called $\mathsf{FindLongestSubstring}$. It takes two additional parameters as inputs: an integer u and a parameter s for the amount of space we can use. It outputs a three tuple: an index l, a pair of indices (p, q) and an integer d. Let l^0 be the largest index such that there is a substring of y within edit distance u to $x[1 : l^0]$.

We show the following two properties of $\mathsf{FindLongestSubstring}$: (1) $l \geq l^0$, and (2) for any substring $y[p^* : q^*]$, $\mathsf{ED}(x[1 : l], y[p : q]) \leq d \leq c(u, s)\mathsf{ED}(x[1 : l], y[p^* : q^*])$. Here, $c(u, s)$ is a function of (u, s) that measures the approximation factor. If $u \leq s$, $\mathsf{FindLongestSubstring}$ outputs $l = l^0$ and the substring of y that is closest to $x[1 : l]$ using $O(s \log n)$ space by doing exact computation. In this case we set $c(u, s) = 1$. Otherwise, it calls $\mathsf{FindLongestSubstring}$ itself up to s times with parameters u/s and s. This gives us $T \leq s$ outputs $\{l_i, (p_i, q_i), d_i\}$ for $i \in [T]$. Let $\tilde{y}$ be the concatenation of $y[p_i : q_i]$ for $i = 1$ to T. We find the pair of indices (p, q) such that $y[p : q]$ is the substring that minimizes $\mathsf{ED}(\tilde{y}, y[p : q])$. We output $l = \sum_{j=1}^{T} l_j$, (p, q), and $d = \mathsf{ED}(\tilde{y}, y[p : q]) + \sum_{i=1}^{T} d_i$. We then use induction to show property (1) and (2) hold for these outputs, where $c(u, s) = 2(c(u/s, s) + 1)$ if $u > s$ and $c(u, s) = 1$ if $u \leq s$. Thus we have $c(u, s) = 2^{O(\log_s u)}$.

This gives an $O(k^\delta/\delta \text{ polylog}(n))$ space algorithm as follows. We run algorithm $\mathsf{FindLongestSubstring}$ with $u = k^{1-\delta}$ and $s = k^\delta$ to find T tuples: $\{l_i, (p_i, q_i), d_i\}$. Again, let $\tilde{y}$ be the concatenation of $y[p_i : q_i]$ from $i = 1$ to T. Similar to the $O(\sqrt{k} \text{ polylog}(n))$ space algorithm, we can show $T \leq k^\delta$ and $\mathsf{ED}(y, \tilde{y}) + \sum_{i=1}^{T} d_i$ is a $2c(k^{1-\delta}, k^\delta) + 1 = 2^{O(1/\delta)}$ approximation of $\mathsf{ED}(x, y)$. Since the depth of recursion is at most $1/\delta$ and each level of recursion needs $O(k^\delta \text{ polylog}(n))$ space, $\mathsf{FindLongestSubstring}$ uses $O(k^\delta/\delta \text{ polylog}(n))$ space.

The two algorithms above both require a given value k. To remove this constraint, our observation is that the two previous algorithms actually only need the number k to satisfy the following relaxed condition: there is a partition of x into k^δ blocks such that for each block x^i, there is a substring of y within edit distance $k^{1-\delta}$ to x^i. Thus, when such a k is not given, we can do the following. We first set k to be a large constant k_0. While the algorithm reads x from left to right, let T' be the number of $\{l_i, (p_i, q_i), d_i\}$ we have stored so far. Each time we run $\mathsf{FindLongestSubstring}$ at this level, we increase T' by 1. If the current k satisfies the relaxed condition, then by a similar argument as before T' should never exceed k^δ. Thus whenever $T' = k^\delta$, we increase k by a $2^{1/\delta}$ factor. Assume that k is updated m times in total and after the i-th update, k becomes k_i. We show that $k_m = O(\mathsf{ED}(x, y))$ (but k_m may be much smaller than $\mathsf{ED}(x, y)$). To see this, suppose $k_j > 2^{1/\delta}\mathsf{ED}(x, y)$ for some $j \leq m$. Let t_j be the position of x where k_{j-1} is updated to k_j. We know it is possible to divide $x[t_j : n]$ into $\mathsf{ED}(x, y)^\delta$ blocks such that for each part, there is a substring of y within edit distance $\mathsf{ED}(x, y)^{1-\delta} \leq k_j^{1-\delta}$ to it. By property (1) and a similar argument as before, we will run $\mathsf{FindLongestSubstring}$ at most $\mathsf{ED}(x, y)^\delta$ times until we reach the end of x. Since $k_j^\delta - k_{j-1}^\delta > \mathsf{ED}(x, y)^\delta$, T' must be always smaller than k_j^δ and hence k_j will not be updated. Therefore we must have $j = m$. This shows $k_{m-1} \leq 2^{1/\delta}\mathsf{ED}(x, y)$ and $k_m \leq 2^{2/\delta}\mathsf{ED}(x, y)$. Running $\mathsf{FindLongestSubstring}$ with $k \leq k_m$ takes $O(k_m^\delta/\delta \text{ polylog}(n)) = O(\mathsf{ED}(x, y)^\delta/\delta \text{ polylog}(n))$ space and the number of intermediate results ((p_i, q_i) and d_i's) is $O(k_m^\delta) = O(\mathsf{ED}(x, y)^\delta)$. This gives us a $2^{O(1/\delta)}$ approximation algorithm with space complexity $O(\mathsf{ED}(x, y)^\delta/\delta \text{ polylog}(n))$.

Algorithm for LCS. We show that the reduction from LCS to ED discovered in [24] can work in the asymmetric streaming model with a slight modification. Combined with our algorithm for ED, this gives a n^δ space algorithm for LCS that achieves a $1/2 + \varepsilon$ approximation for binary strings. We defer the detailed analysis and proof to the full version.

C Lower Bound for ED in the Standard Streaming Model

▶ **Theorem 34.** *There exists a constant $\varepsilon > 0$ such that for strings $x, y \in \{0,1\}^n$, any deterministic R pass streaming algorithm achieving an εn additive approximation of* $\mathsf{ED}(x, y)$ *needs $\Omega(n/R)$ space.*

Proof. Consider an asymptotically good insertion-deletion code $C \subseteq \{0,1\}^n$ over a binary alphabet (See [27] for example). Assume C has rate α and distance β. Both α and β are some constants larger than 0, and we have $|C| = 2^{\alpha n}$. Also, for any $x, y \in C$ with $x \neq y$, we have $\mathsf{ED}(x, y) \geq \beta n$. Let $\varepsilon = \beta/2$ and consider the two party communication problem where player 1 holds $x \in C$ and player 2 holds $y \in C$. The goal is to decide whether $x = y$. Any deterministic protocol has communication complexity at least $\log |C| = \Omega(n)$. Note that any algorithm that approximates $\mathsf{ED}(x, y)$ within an εn additive error can decide whether $x = y$. Thus the theorem follows. ◀

We note that the same bound holds for Hamming distance by the same argument.

An ETH-Tight Algorithm for Multi-Team Formation

Daniel Lokshtanov
University of California, Santa Barbara, CA, USA

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

Subhash Suri
University of California, Santa Barbara, CA, USA

Jie Xue
New York University Shanghai, China

Abstract

In the Multi-Team Formation problem, we are given a ground set C of n candidates, each of which is characterized by a d-dimensional attribute vector in $\mathbb{R}^d$, and two positive integers α and β satisfying $\alpha\beta \leq n$. The goal is to form α disjoint teams $T_1, ..., T_\alpha \subseteq C$, each of which consists of β candidates in C, such that the total score of the teams is maximized, where the score of a team T is the sum of the h_j maximum values of the j-th attributes of the candidates in T, for all $j \in \{1, ..., d\}$. Our main result is an $2^{2^{O(d)}} n^{O(1)}$-time algorithm for Multi-Team Formation. This bound is ETH-tight since a $2^{2^{d/c}} n^{O(1)}$-time algorithm for any constant $c > 12$ can be shown to violate the Exponential Time Hypothesis (ETH). Our algorithm runs in polynomial time for all dimensions up to $d = c \log \log n$ for a sufficiently small constant $c > 0$. Prior to our work, the existence of a polynomial time algorithm was an open problem even for $d = 3$.

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

Keywords and phrases Team formation, Parameterized algorithms, Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.28

Acknowledgements We would like to thank an anonymous reviewer for pointing us to the statement of [10], allowing us to drastically simplify a previous version of the paper.

1 Introduction

The problem of team formation arises in many organizational settings – project management, product development, team sports, academic committees, legal defence teams, to name a few – and remains an important area of research in mathematical social sciences [12, 16, 22, 24]. Within computer science and operations research, several application domains – distributed robotics, AI, multi-agent systems, online crowdsourcing, databases – also use team formation models for execution of complex tasks that require cooperation or coalition of multiple agents with different capabilities [5, 18, 21, 23]. The basic setting of a Team Formation problem includes a ground set C of n candidates and a number $\beta \leq n$. The goal is to form a team $T \subseteq C$ of a size β such that $\mathsf{scr}(T)$ is maximized, where $\mathsf{scr}(\cdot)$ is a pre-defined scoring function. A concrete example of a scoring function frequently used in the literature [11, 25] (often in conjunction with other, more complex measures of team performance) is the *skill coverage* function. There is a set U of useful skills, each candidate $a \in C$ has a subset S_a of these skills, and we evaluate the team by the number of different skills covered by the team members. In other words, $\mathsf{scr}(T) = |\bigcup_{a \in T} S_a|$. It is easy to see that Team Formation with the skill

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 28; pp. 28:1–28:9
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

coverage scoring function is equivalent to the-well studied Maximum Coverage problem, which is NP-complete [7], admits a $(1-\frac{1}{e})$-approximation algorithm [15], and is NP-hard to approximate [4] within any factor smaller than $1-\frac{1}{e}$.

A natural generalization of Team Formation is the Multi-team Formation problem, where we want to form α disjoint teams $T_1, \ldots, T_\alpha \subseteq C$ each of size β that collectively maximize the total score $\sum_{i=1}^{\alpha} \mathsf{scr}(T_i)$. This generalization is well-motivated in practice: in many applications, we want to form multiple teams from a common pool of candidates, where candidate can belong to at most one team. Multi-team Formation has some resemblance to the *coalition structure generation* problem in multi-agent systems and AI, where the goal is to partition a set of candidates into groups, called coalitions [19]. However, in these applications, the scoring function for evaluating a coalition is assumed to be an arbitrary black box function. As a result, the size of each team (coalition) is not explicitly specified but rather determined by the objective function of maximizing the total coalition structure value – e.g. if putting all the candidates into a single coalition maximizes the total value, then that is the optimal solution. In [14], a dynamic programming algorithm is described for computing an optimal coalition structure in time $O(3^n)$. Unlike the (single) Team Formation problem, Multi-team Formation has not yet received much attention, and beyond the exponential bound of Michalak et al. [14], no algorithmic result appears to be known for forming multiple teams except for the recent work of Schibler et al. [20].

In this paper, we follow Schibler et al. [20] and investigate Multi-team Formation with a fundamental scoring function, called *sum-of-maxima* scoring, to be defined below. A common model for characterizing a candidate is a multi-dimensional attribute vector in which each entry measures a certain performance of the candidate. For instance, in college admissions, such a vector may include scores of different standardized tests, grade point averages, etc. In project management, the categories may include various technical skills as well as non-technical attributes such as leadership qualities. Following Page's influential work on team performance [16], it is generally acknowledged that simply adding up all the scores is a poor measure of team performance – instead, strength in multiple dimensions (skill diversity) is essential. When the candidates are characterized by attribute vectors, one natural scoring is to take the *best attribute* of the candidates in the team T in each dimension and set the score of T to be the sum of these best attributes. Kleinberg and Raghu [8], in their work on team performance metrics and testing, suggested extending this further to sum-of-top-h scores in each dimension, for some $h \leq \beta$, ensuring both coverage of all the skills (dimensions) and robustness (no single point of failure). We allow a slightly more general scoring rule, where for each dimension j, a possibly different number h_j of top attributes are considered. We call this the *sum-of-maxima* scoring. Formally, each candidate $a \in C$ is characterized by a d-dimensional attribute vector $(\kappa_1(a), \ldots, \kappa_d(a)) \in \mathbb{R}^d$. For a given vector $\mathbf{h} = (h_1, \ldots, h_d) \in \mathbb{Z}_+^d$, the sum-of-$\mathbf{h}$-maxima scoring function is defined as

$$\mathsf{som}_{\mathbf{h}}(T) = \sum_{j=1}^{d} \max^{h_j} \{\kappa_j(a) : a \in T\}, \tag{1}$$

where the notation $\max^{h_j} S$ denotes the sum of the largest h_j numbers in the *multiset* S of numbers (if $|S| < h_j$, then $\max^{h_j} S$ is the sum of all numbers in S). It is easy to see that the sum-of-maxima scoring function generalizes skill coverage. In particular, the Maximum Coverage problem is a special case of Multi-team Formation where $\mathbf{h}$ is the vector of all 1's, all the candidate attributes are binary, and $\alpha = 1$.

In the rest of the paper, the Multi-team Formation problem we discuss is always with respect to sum-of-maxima scoring. Since Multi-team Formation generalizes Maximum Coverage, it is clearly NP-hard (when the dimension d is unbounded). Schibler et al. [20]

proved that Multi-team Formation is NP-hard when $d = \Theta(\log n)$, even with binary attributes and team size $\beta \geq 4$. These hardness claims, however, depend on the rather unrealistic assumption that the dimension d of attribute vectors must be quite large – in most applications, the number of attributes (e.g., standardized test scores) is much more modest. Therefore, it is interesting to study the complexity of Multi-team Formation when d is small. Indeed, Schibler et al. [20] gave a polynomial-time algorithm for the case of $d = 2$ and leave as an open problem whether the problem is polynomial time solvable for any constant $d \geq 3$. Our main result is a new algorithm for Multi-team Formation, which runs in polynomial time for any $d \leq c \cdot \log\log n$ where $c > 0$ is a sufficiently small constant (and hence for any constant d). Specifically, we prove the following theorem.

▶ **Theorem 1.** *There exists a $2^{2^{O(d)}} n^{O(1)}$-time algorithm for Multi-team Formation.*

In the view of Parameterized Complexity, this is the first Fixed-Parameter Tractable (FPT) algorithm for Multi-team Formation parameterized by the dimension d. The analysis of the algorithm of Theorem 1 involves a novel application of *Graver Bases*, a notion that has successfully been applied to yield fixed parameter tractability results for a number of problems in Mathematical Programming. To the best of our understanding, however, none of the existing state-of-the art results [3, 6, 9] can be applied in a black box fashion to yield an FPT algorithm for Multi-team Formation parameterized by d. It remains an interesting research question to generalize Theorem 1 to an FPT algorithm for solving a class of mathematical programs that is powerful enough to encompass Multi-team Formation.

The time complexity of our algorithm grows double exponentially with d and, under plausible complexity theoretic assumptions, it cannot be substantially improved. In particular, a fresh look at the NP-hardness reduction of Schibler et al. [20] reveals that any algorithm that solves Multi-team Formation in $2^{2^{d/c}} \cdot n^{O(1)}$ time for a sufficiently large constant c will violate the Exponential Time Hypothesis (ETH).

▶ **Theorem 2.** *The existence of a $2^{2^{d/c}} n^{O(1)}$-time algorithm for Multi-team Formation with any constant $c > 12$ violates the Exponential Time Hypothesis (ETH).*

Therefore, our algorithm is ETH-tight, and adds Multi-team Formation to the small club of problems (together with Edge Clique Cover [2] and Distinct Vectors [17]) for which both a double exponential time algorithm and a double exponential time lower bound were known.

2 An ETH-tight algorithm

In this section, we present our algorithm for Multi-team Formation in Theorem 1, and also prove Theorem 2 (which is easy). We begin by introducing some basic notations. Let $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Z}_+$, $\mathbb{R}$ to denote the set of natural numbers (including 0), integers, positive integers, and real numbers, respectively. For two vectors $\mathbf{u}, \mathbf{v}$ of the same dimension, we use $\langle \mathbf{u}, \mathbf{v} \rangle$ to denote the inner product of $\mathbf{u}, \mathbf{v}$. For a number $k \in \{0, \dots, 2^d - 1\}$, let $\mathsf{bin}(k)$ be the d-bit binary representation of k, which is a d-dimensional binary vector, and $\mathsf{bin}_j(k)$ be the j-th entry of $\mathsf{bin}(k)$, i.e., the j-th (highest) digit of the d-bit binary representation of k.

Recall that in Multi-team Formation, the input includes a set C of n candidates where each $a \in C$ is characterized by a d-dimensional attribute vector $\kappa(a) = (\kappa_1(a), \dots, \kappa_d(a)) \in \mathbb{R}^d$, a vector $\mathbf{h} = (h_1, \dots, h_d) \in \mathbb{Z}_+^d$ used for defining the scoring function $\mathsf{som}_{\mathbf{h}}$, and two integers $\alpha, \beta > 0$ satisfying $\alpha\beta \leq n$. The goal is to form α disjoint teams $T_1, \dots, T_\alpha \subseteq C$ of size β such that $\sum_{i=1}^{\alpha} \mathsf{som}_{\mathbf{h}}(T_i)$ is maximized. Without loss of generality, we may assume that $h_j \leq \beta$ for all $j \in \{1, \dots, d\}$, because $\mathsf{som}_{\mathbf{h}}(T)$ remains unchanged for all $T \subseteq C$ with $|T| = \beta$ if we replace all $h_j > \beta$ with β, as one can easily verified. Let opt denote the optimum of the input instance.

Consider a solution $T_1, \dots, T_\alpha \subseteq C$ of the problem. The total score of this solution, $\sum_{i=1}^{\alpha} \mathsf{som}_{\mathbf{h}}(T_i)$, is the sum of some attributes $\kappa_j(a)$ for $a \in \bigcup_{i=1}^{\alpha} T_i$. For each team T_i, each candidate $a \in T_i$ contributes to the score $\mathsf{som}_{\mathbf{h}}(T_i)$ in a certain way. Specifically, for each dimension $j \in \{1, \dots, d\}$, the candidate a is either among the top h_j candidates in T_i in that dimension, in which case it contributes $\kappa_j(a)$, or it is not, in which case it contributes 0[1]. The information of how the d attributes of $a \in T_i$ contribute to the score $\mathsf{som}_{\mathbf{h}}(T_i)$ can be depicted by a number $k \in \{0, \dots, 2^d - 1\}$ (or equivalently, a d-bit binary string) where $\mathsf{bin}_j(k) = 1$ if $\kappa_j(a)$ contributes to $\mathsf{som}_{\mathbf{h}}(T_i)$ and $\mathsf{bin}_j(k) = 0$ if $\kappa_j(a)$ does not contribute, for $j \in \{1, \dots, d\}$. We call k the *type* of the candidate a in the solution $T_1, \dots, T_\alpha$. Now every candidate in $\bigcup_{i=1}^{\alpha} T_i$ has its type, which is a number in $\{0, \dots, 2^d - 1\}$. For the unassigned candidates, i.e., the candidates in $C \setminus \bigcup_{i=1}^{\alpha} T_i$, we simply say their type is $\square$ (in the solution $T_1, \dots, T_\alpha$). In this way, we give every candidate in C a type in the solution, which is an element in $\mathit{\Gamma} = \{0, \dots, 2^d - 1\} \cup \{\square\}$. We then define the *type assignment* (or *assignment* for short) of the solution $T_1, \dots, T_\alpha$ as the function $\pi : C \to \mathit{\Gamma}$ that maps each candidate to its type in the solution.

We consider the following question: for a solution $T_1, \dots, T_\alpha \subseteq C$, if we were only given its type assignment $\pi : C \to \mathit{\Gamma}$ without the original teams $T_1, \dots, T_\alpha$, how much information about $T_1, \dots, T_\alpha$ can we recover from π? Observe first that we *can* easily recover the total score $\sum_{i=1}^{\alpha} \mathsf{som}_{\mathbf{h}}(T_i)$ of the solution, simply because the types of the candidates record how their attributes contribute to the total score. Specifically, if we define

$$\mathsf{scr}(\pi) = \sum_{a \in C,\ \pi(a) \neq \square} \langle \mathsf{bin}(\pi(a)), \kappa(a) \rangle = \sum_{a \in C,\ \pi(a) \neq \square} \left(\sum_{j=1}^{d} \mathsf{bin}_j(\pi(a)) \cdot \kappa_j(a) \right), \tag{2}$$

which we call the *score* of π, then it is clear that $\sum_{i=1}^{\alpha} \mathsf{som}_{\mathbf{h}}(T_i) = \mathsf{scr}(\pi)$. At the same time, however, we *cannot* recover the teams $T_1, \dots, T_\alpha$ from π, because it can happen that different solutions share the same type assignment (for example, there are situations where two candidates with the same attributes, but in different teams, could be swapped without changing their type, leading to a different solution with the same type assignment).

We say a solution $T_1, \dots, T_\alpha \subseteq C$ *realizes* a type assignment function $\pi : C \to \mathit{\Gamma}$ if π is the type assignment of $T_1, \dots, T_\alpha$. Thus, for a type assignment function $\pi : C \to \mathit{\Gamma}$, there could be zero, one, or more solutions that realize it, and all such solutions have the same total score. We say π is *realizable* if there exists at least one solution that realizes π. What we want is essentially a realizable $\pi : C \to \mathit{\Gamma}$ that maximizes $\mathsf{scr}(\pi)$.

Note that there are too many (type assignment) functions $\pi : C \to \mathit{\Gamma}$ to go over all of them; indeed, the number of such functions is $(2^d + 1)^n$. Furthermore, it turns out to be difficult to check whether a given π is realizable, and even if we know π is realizable, it is not clear how to find a witness solution $T_1, \dots, T_\alpha \subseteq C$ that realizes π. For this reason, our algorithm does not work on type assignment functions directly. Instead, we only guess some distinguishing features of the type assignment of an optimal solution. Perhaps the most natural distinguishing feature is "how many candidates are there of each type". We

[1] Here we assume that the "sum-of-top-h_j" function $\max^{h_j}$ in Equation 1 breaks ties in a certain way (e.g., take the attributes of the candidates with smaller indices first, etc.) so that the contributing attributes of each candidate in the team is uniquely defined.

formalize this as follows. The *configuration* of a function $\pi : C \to \mathit{\Gamma}$ is a 2^d-dimensional vector $\mathsf{conf}(\pi) = (c_0, \dots, c_{2^d-1}) \in \mathbb{N}^{2^d}$ where $c_k = |\pi^{-1}(\{k\})|$ for $k \in \{0, \dots, 2^d - 1\}$. In other words, the k-th entry c_k of the vector $\mathsf{conf}(\pi)$ records the number of candidates assigned to type k by π.

Clearly, not every vector in $\mathbb{N}^{2^d}$ can be the configuration of some realizable function. Next, we establish a simple *necessary* (but not sufficient) condition for a vector to be the configuration of some realizable function. Suppose $\mathbf{c} = (c_0, \dots, c_{2^d-1})$ is the configuration of a realization function $\pi : C \to \mathit{\Gamma}$ and let $T_1, \dots, T_\alpha \subseteq C$ be the solution that realizes π, i.e., π is the type assignment of $T_1, \dots, T_\alpha$. For $i \in \{1, \dots, \alpha\}$ and $k \in \{0, \dots, 2^d - 1\}$, let $v_{i,k}$ be the number of candidates in T_i which are mapped to k by π, i.e., $v_{i,k} = |\pi^{-1}(\{k\}) \cap T_i|$. Since π maps all candidates in $C \backslash (\bigcup_{i=1}^{\alpha} T_i)$ to $\Box$, we have $c_k = |\pi^{-1}(\{k\})| = \sum_{i=1}^{\alpha} v_{i,k}$ for all $k \in \{0, \dots, 2^d - 1\}$ and hence $\mathbf{c} = \sum_{i=1}^{\alpha} \mathbf{v}_i$ where $\mathbf{v}_i = (v_{i,0}, \dots, v_{i,2^d-1})$. Now what are the conditions that each $\mathbf{v}_i$ has to satisfy? First, since $|T_i| = \beta$ and π maps all candidates in T_i to $\{0, \dots, 2^d - 1\}$, the sum of all entries of $\mathbf{v}_i$ is equal to β, i.e., $\sum_{k=0}^{2^d-1} v_{i,k} = \beta$. Second, for each $j \in \{1, \dots, d\}$, the number of candidates in T_i which contribute in the j-th dimension is precisely h_j, and thus the sum of the entries of $\mathbf{v}_i$ corresponding to types k which contribute in the j-th dimension, i.e., $\mathsf{bin}_j(k) = 1$, is equal to h_j, i.e., $\sum_{k=0}^{2^d-1} v_{i,k} \cdot \mathsf{bin}_j(k) = h_j$. To summarize, in order to be the configuration of some realizable function, a vector $\mathbf{c}$ must be the sum of α vectors each of which satisfies the above two conditions. This is exactly the necessary condition we want. Formally, we give the following definition.

▶ **Definition 3** (legal vectors). *A vector $\mathbf{v} = (v_0, \dots, v_{2^d-1}) \in \mathbb{N}^{2^d}$ is **$(\boldsymbol{\beta}, \mathbf{h})$-legal** (or simply **legal** when β and $\mathbf{h}$ are all clear from the context) if $\sum_{k=0}^{2^d-1} v_k = \beta$ and $\sum_{k=0}^{2^d-1} v_k \cdot \mathsf{bin}_j(k) = h_j$ for all $j \in \{1, \dots, d\}$.*

▶ **Fact 4.** *If $\pi : C \to \mathit{\Gamma}$ is realizable, then $\mathsf{conf}(\pi)$ is the sum of α legal vectors.*

Note that the converse of the above fact is not true, i.e., it is possible that $\mathsf{conf}(\pi)$ is the sum of α legal vectors but π is not the type assignment of any solution. However, we have the following nice property.

▶ **Lemma 5.** *If $\pi : C \to \mathit{\Gamma}$ is a function such that $\mathsf{conf}(\pi)$ is the sum of α legal vectors, then $\mathsf{scr}(\pi) \le \mathsf{opt}$. Furthermore, given π and a decomposition $\mathsf{conf}(\pi) = \sum_{i=1}^{\alpha} \mathbf{v}_i$ into legal vectors, one can compute in $O(n + 2^d)$ time a solution $T_1, \dots, T_\alpha \subseteq C$ of the problem such that $\mathsf{scr}(\pi) \le \sum_{i=1}^{\alpha} \mathsf{som}_{\mathbf{h}}(T_i)$.*

Proof. Suppose $\mathsf{conf}(\pi) = (c_0, \dots, c_{2^d-1}) = \sum_{i=1}^{\alpha} \mathbf{v}_i$, where each $\mathbf{v}_i = (v_{i,0}, \dots, v_{i,2^d-1})$ is a legal vector. For $k \in \{0, \dots, 2^d - 1\}$, we arbitrarily partition the c_k candidates in $\pi^{-1}(\{k\})$ into α groups $G_{1,k}, \dots, G_{\alpha,k}$ such that $|G_{i,k}| = v_{i,k}$; this is possible because $c_k = \sum_{i=1}^{\alpha} v_{i,k}$. We then define $T_i = \bigcup_{k=0}^{2^d-1} G_{i,k}$ for $i \in \{1, \dots, \alpha\}$. It is clear that $T_1, \dots, T_\alpha$ are disjoint subsets of C with size β. Therefore, $\sum_{i=1}^{\alpha} \mathsf{som}_{\mathbf{h}}(T_i) \le \mathsf{opt}$. It suffices to show $\mathsf{scr}(\pi) \le \sum_{i=1}^{\alpha} \mathsf{som}_{\mathbf{h}}(T_i)$. Note that $\pi(a) \in \{0, \dots, 2^d - 1\}$ for all $a \in \bigcup_{i=1}^{\alpha} T_i$ and $\pi(a) = \Box$ for all $a \in C \backslash (\bigcup_{i=1}^{\alpha} T_i)$. So we have $\mathsf{scr}(\pi) = \sum_{i=1}^{\alpha} \sum_{a \in T_i} \sum_{j=1}^{d} \mathsf{bin}_j(\pi(a)) \cdot \kappa_j(a)$. Equivalently, $\mathsf{scr}(\pi) = \sum_{i=1}^{\alpha} \sum_{j=1}^{d} \sum_{a \in T_{i,j}} \kappa_j(a)$, where $T_{i,j} = \{a \in T_i : \mathsf{bin}_j(\pi(a)) = 1\}$. Since $\mathbf{v}_1, \dots, \mathbf{v}_\alpha$ are $(\beta, \mathbf{h})$-legal, we have $|T_{i,j}| = h_j$ for all $i \in \{1, \dots, \alpha\}$ and $j \in \{1, \dots, d\}$. Thus, $\sum_{a \in T_{i,j}} \kappa_j(a) \le \max^{h_j}\{\kappa_j(a) : a \in T_i\}$ (recall that $\max^{h_j} S$ denotes the sum of the largest h_j numbers in the multiset S). It follows that

$$\mathsf{scr}(\pi) = \sum_{i=1}^{\alpha} \sum_{j=1}^{d} \sum_{a \in T_{i,j}} \kappa_j(a) \le \sum_{i=1}^{\alpha} \sum_{j=1}^{d} \max^{h_j}\{\kappa_j(a) : a \in T_i\} = \sum_{i=1}^{\alpha} \mathsf{som}_{\mathbf{h}}(T_i).$$

Therefore, $\mathsf{scr}(\pi) \le \mathsf{opt}$. If we are given π and the legal vectors $\mathbf{v}_1, \dots, \mathbf{v}_\alpha$, then the teams $T_1, \dots, T_\alpha$ can clearly be constructed in $O(n + 2^d)$ time. ◀

With the above lemma in hand, it now suffices to compute a function $\pi^* : C \to \mathit{\Gamma}$ with the maximum $\mathsf{scr}(\pi^*)$ such that $\mathsf{conf}(\pi^*)$ is the sum of α legal vectors and a decomposition $\mathsf{conf}(\pi^*) = \sum_{i=1}^{\alpha} \mathbf{v}_i$ into legal vectors. Indeed, once we have the function π^* and the decomposition $\mathsf{conf}(\pi^*) = \sum_{i=1}^{\alpha} \mathbf{v}_i$, we can apply the above lemma to obtain a solution $T_1^*, \dots, T_\alpha^* \subseteq C$ satisfying $\mathsf{scr}(\pi^*) \leq \sum_{i=1}^{\alpha} \mathsf{som}_{\mathbf{h}}(T_i^*)$. Note that Fact 4 guarantees $\mathsf{scr}(\pi^*) \geq \mathsf{opt}$, which implies $\sum_{i=1}^{\alpha} \mathsf{som}_{\mathbf{h}}(T_i^*) \geq \mathsf{opt}$, i.e., $T_1^*, \dots, T_\alpha^*$ is an optimal solution.

Next, we show how to compute the function π^* and the decomposition efficiently. To this end, we formulate the problem as an integer linear programming (ILP) instance. For each candidate $a \in C$, we define $2^d + 1$ variables $u_0(a), \dots, u_{2^d-1}(a), u_\square(a)$. These variables are used to encode the information of π^*. Specifically, the variable $u_k(a)$ will indicate whether $\pi^*(a) = k$: $u_k(a) = 1$ if $\pi^*(a) = k$ and $u_k(a) = 0$ if $\pi^*(a) \neq k$. Therefore, the values of these variables are in $\{0, 1\}$ and must satisfy the constraints $\sum_{k \in \mathit{\Gamma}} u_k(a) = 1$ for all $a \in C$. Our objective function, which is $\mathsf{scr}(\pi^*)$, can be expressed as $\sum_{a \in C} \sum_{k=0}^{2^d-1} u_k(a) \cdot \langle \mathsf{bin}(k), \kappa(a) \rangle$, according to the formula of Equation 2. In addition, we need variables and constraints to guarantee that $\mathsf{conf}(\pi^*)$ is the sum of α legal vectors. Note that $\mathsf{conf}(\pi^*)$ can be expressed as $\sum_{a \in C} \mathbf{u}(a)$, where $\mathbf{u}(a) = (u_0(a), \dots, u_{2^d-1}(a))$. We introduce variables $v_{i,0}, \dots, v_{i,2^d-1}$ for all $i \in \{1, \dots, \alpha\}$. Each vector $\mathbf{v}_i = (v_{i,0}, \dots, v_{i,2^d-1})$ is supposed to be a legal vector. So we include the constraints $\sum_{k=0}^{2^d-1} v_{i,k} = \beta$ and $\sum_{k=0}^{2^d-1} v_{i,k} \cdot \mathsf{bin}_j(k) = h_j$ for all $j \in \{1, \dots, d\}$. Finally, we need to constraint $\sum_{a \in C} \mathbf{u}(a) = \sum_{i=1}^{\alpha} \mathbf{v}_i$ to ensure that $\mathsf{conf}(\pi^*)$ is the sum of $\mathbf{v}_1, \dots, \mathbf{v}_\alpha$. In sum, our ILP instance is

$$\begin{aligned} &\max \sum_{a \in C} \sum_{k=0}^{2^d-1} u_k(a) \cdot \langle \mathsf{bin}(k), \kappa(a) \rangle \\ \text{s.t.}\quad & \textstyle\sum_{k \in \mathit{\Gamma}} u_k(a) = 1 \text{ for all } a \in C, \\ & \textstyle\sum_{k=0}^{2^d-1} v_{i,k} = \beta \text{ for all } i \in \{1, \dots, \alpha\}, \\ & \textstyle\sum_{k=0}^{2^d-1} v_{i,k} \cdot \mathsf{bin}_j(k) = h_j \text{ for all } i \in \{1, \dots, \alpha\} \text{ and } j \in \{1, \dots, d\}, \\ & \textstyle\sum_{a \in C} \mathbf{u}(a) = \sum_{i=1}^{\alpha} \mathbf{v}_i, \\ & \mathbf{0} \leq \mathbf{u}(a) \leq \mathbf{1} \text{ for all } a \in C \text{ and } \mathbf{v}_i \geq \mathbf{0} \text{ for all } i \in \{1, \dots, \alpha\}. \end{aligned} \tag{3}$$

The above ILP instance has $(2^d + 1)n + 2^d\alpha$ variables, thus we cannot apply any general ILP solver to solve it in time polynomial in n. Fortunately, this ILP instance has some nice structural property which we can exploit. In order to describe the property, we need to first introduce the notion of N-fold ILP. In an N-fold ILP instance, the linear constraints on the variable vector $\mathbf{x}$ can be represented as $\mathbf{x}_{\text{low}} \leq \mathbf{x} \leq \mathbf{x}_{\text{high}}$ and $A\mathbf{x} = \mathbf{b}$ where

$$A = \begin{pmatrix} M_1 & M_2 & \cdots & M_N \\ M_1' & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & M_2' & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & M_N' \end{pmatrix}. \tag{4}$$

Let r be the maximum number of rows of the matrices $M_1, \dots, M_N$ and $M_1', \dots, M_N'$, and t be the maximum number of columns of the matrices $M_1', \dots, M_N'$. It was shown in [10] that the N-fold ILP instance can be solved in $\Delta^{O(r^3)}(Nt)^{O(1)}$ time, where $\Delta = \max\{2, \|A\|_\infty\}$.

We observe that our ILP instance in Equation 3 is in fact an N-fold ILP instance with $N = n + \alpha$, $r = 2^d$, $t = 2^d + 1$, and $\Delta = 2$. To this end, we classify our variables into $n + \alpha$ groups. For each $a \in C$, we have a group $G_a = \{u_k(a) : k \in \mathit{\Gamma}\}$ of $2^d + 1$ variables. For each $i \in \{1, \dots, \alpha\}$, we have a group $G'_i = \{v_{i,0}, \dots, v_{i,2^d-1}\}$ of 2^d variables. We obtain our variable vector $\mathbf{x}$ by permuting all $(2^d+1)n + 2^d\alpha$ variables such that the variables in each group are consecutive in the permutation. Now notice that the constraint $\sum_{k \in \Gamma} u_k(a) = 1$ is only for the variables in G_a, while the constraints $\sum_{k=0}^{2^d-1} v_{i,k} = \beta$ and $\sum_{k=0}^{2^d-1} v_{i,k} \cdot \mathsf{bin}_j(k) = h_j$ for $j \in \{1, \dots, d\}$ are only for the variables in G'_i. We call these constraints *local constraints*. Local constraints can be realized using the M'-matrices in Equation 4; the number of rows of these matrices is at most $d+1$ because we have one local constraint for each group G_a and $d+1$ local constraints for each group G'_i, and the number of columns of these matrices is at most 2^d+1 because each group has at most 2^d+1 variables. Finally, we have the "global" constraints $\sum_{a \in C} \mathbf{u}(a) = \sum_{i=1}^{\alpha} \mathbf{v}_i$. Since the dimension of the vectors $\mathbf{u}(a)$ and $\mathbf{v}_i$ is 2^d, the global constraints can be expressed as $M\mathbf{x} = \mathbf{0}$ for some 2^d-row matrix M, which can be in turn realized using matrices $M_1, \dots, M_N$ in Equation 4. To summarize, the constraints of our ILP instance of Equation 3 can be written as $A\mathbf{x} = \mathbf{b}$, where A is of the form of Equation 4 in which $N = n + \alpha$ and the maximum number of rows (resp., columns) of the matrices $M_1, \dots, M_N, M'_1, \dots, M'_N$ is 2^d (resp., $2^d + 1$). Also, as one can easily verified, the entries of A are all in $\{-1, 0, 1\}$, which implies $\|A\|_\infty \leq 1$ and $\Delta = 2$. Therefore, applying the algorithm of [10] solves our ILP instance in $2^{2^{O(d)}} n^{O(1)}$ time.

After solving the ILP instance of Equation 3, we obtain the desired function $\pi^* : C \to \Gamma$ by setting $\pi^*(a)$ to be the (unique) element $k \in \Gamma$ satisfying $u_k(a) = 1$, and a decomposition $\mathsf{conf}(\pi^*) = \sum_{i=1}^{\alpha} \mathbf{v}_i$ into legal vectors. As argued before, we can then use Lemma 5 to compute an optimal solution for the problem in $O(n)$ time. The overall running time of our algorithm is $2^{2^{O(d)}} n^{O(1)}$. This proves Theorem 1, which we restate below.

▶ **Theorem 1.** *There exists a $2^{2^{O(d)}} n^{O(1)}$-time algorithm for* Multi-team Formation.

Although the running time of our algorithm depends double exponentially on d, it is ETH-tight and hence unlikely to be substantially improved. The lower bound follows readily from the reduction in [20] and the ETH lower bound in [1] for 3-dimensional Matching.

▶ **Theorem 2.** *The existence of a $2^{2^{d/c}} n^{O(1)}$-time algorithm for* Multi-team Formation *with any constant $c > 12$ violates the Exponential Time Hypothesis (ETH).*

Proof. Let $c > 12$ be a constant. Schibler et al. [20] described a polynomial-time reduction from 3-dimensional Matching to Multi-team Formation with $n = O(m)$ and $d = 12 \log m + O(1)$, where m is the size of the 3-dimensional Matching instance. Therefore, a $2^{2^{d/c}} n^{O(1)}$-time algorithm for Multi-team Formation implies a $2^{m^{12/c}} m^{O(1)}$-time algorithm for 3-dimensional Matching. However, it was shown in [1] that any algorithm with running time $2^{o(m)}$ for 3-dimensional Matching violates the ETH. ◀

3 Conclusion and future work

In this paper, we considered Multi-team Formation under the natural sum-of-maxima scoring rule, and presented an algorithm that runs in $2^{2^{O(d)}} \cdot n^{O(1)}$ time, which is ETH-tight since a $2^{2^{d/c}} \cdot n^{O(1)}$-time algorithm, for any constant $c > 12$, would violate the ETH.

A direction for future work is approximation algorithms for Multi-team Formation. Exploiting the submodularity of the sum-of-maxima scoring function, one can easily formulate Multi-team Formation as a submodular maximization problem with two matroid constraints, which leads to a polynomial-time $(0.5 - \varepsilon)$-approximation algorithm for any constant $\varepsilon > 0$ using the algorithm of [13]. Whether one can achieve a better approximation in polynomial time is an interesting open question to be studied.

References

1 Nikhil Bansal, Tim Oosterwijk, Tjark Vredeveld, and Ruben Van Der Zwaan. Approximating vector scheduling: almost matching upper and lower bounds. *Algorithmica*, 76(4):1077–1096, 2016.

2 Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM J. Comput.*, 45(1):67–83, 2016.

3 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming. *arXiv preprint*, 2019. arXiv:1904.01361.

4 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

5 Erin L. Fitzpatrick and Ronald G. Askin. Forming effective worker teams with multi-functional skill requirements. *Computers & Industrial Engineering*, 48(3):593–608, 2005.

6 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. N-fold integer programming in cubic time. *Mathematical Programming*, 137(1-2):325–341, 2013.

7 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

8 Jon Kleinberg and Maithra Raghu. Team performance with test scores. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, EC '15, pages 511–528, 2015.

9 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21(5):493–503, 2018.

10 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

11 Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 467–476, 2009.

12 Patrick R. Laughlin and Andrea B. Hollingshead. A theory of collective induction. *Organizational Behavior and Human Decision Processes*, 61(1):94–107, 1995.

13 Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Mathematics of Operations Research*, 35(4):795–806, 2010.

14 Tomasz P. Michalak, Talal Rahwan, Edith Elkind, Michael J. Wooldridge, and Nicholas R. Jennings. A hybrid exact algorithm for complete set partitioning. *Artif. Intell.*, 230:14–50, 2016.

15 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978.

16 Scott Page. *The Difference: How the Power of Diversity Creates Better Groups, Firms, Schools, and Societies.* Princeton University Press, 2007.

17 Marcin Pilipczuk and Manuel Sorge. A double exponential lower bound for the distinct vectors problem. *CoRR*, abs/2002.01293, 2020. arXiv:2002.01293.

18 Habibur Rahman, Senjuti Basu Roy, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. Optimized group formation for solving collaborative tasks. *The VLDB Journal*, 28(1):1–23, February 2019.

19 Talal Rahwan, Tomasz P. Michalak, Michael J. Wooldridge, and Nicholas R. Jennings. Coalition structure generation: A survey. *Artif. Intell.*, 229:139–174, 2015.

20 Thomas Schibler, Ambuj Singh, and Subhash Suri. On multi-dimensional team formation. In *Proc. of the 31st Canadian Conference on Computational Geometry*, pages 146–152, 2019.

21 Travis C. Service and Julie A. Adams. Coalition formation for task allocation: theory and algorithms. *Autonomous Agents and Multi-Agent Systems*, 22(2):225–248, March 2011.

22 Marjorie E. Shaw. A comparison of individuals and small groups in the rational solution of complex problems. *The American Journal of Psychology*, 44(3):491–504, 1932.

23 Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artif. Intell.*, 101(1-2):165–200, 1998.

24 I. D. Steiner. *Group process and productivity.* New York: Academic Press, 1972.

25 Xinyu Wang, Zhou Zhao, and Wilfred Ng. A comparative study of team formation in social networks. In *Database Systems for Advanced Applications - 20th International Conference, DASFAA 2015*, pages 389–404. Springer, 2015.

Dominating Set in Weakly Closed Graphs is Fixed Parameter Tractable

Daniel Lokshtanov ✉
University of California Santa Barbara, CA, USA

Vaishali Surianarayanan ✉ 🏠
University of California Santa Barbara, CA, USA

Abstract

In the Dominating Set problem the input is a graph G and an integer k, the task is to determine whether there exists a vertex set S of size at most k so that every vertex not in S has at least one neighbor in S. We consider the parameterized complexity of the Dominating Set problem, parameterized by the solution size k, and the *weak closure* of the input graph G. Weak closure of graphs was recently introduced by Fox et al. [*SIAM J. Comp. 2020*] and captures sparseness and triadic closure properties found in real world graphs. A graph G is *weakly c-closed* if for every induced subgraph G' of G, there exists a vertex $v \in V(G')$ such that every vertex u in $V(G')$ which is non-adjacent to v has less than c common neighbors with v. The weak closure of G is the smallest integer γ such that G is weakly γ-closed. We give an algorithm for Dominating Set with running time $k^{O(\gamma^2 k^3)} n^{O(1)}$, resolving an open problem of Koana et al. [ISAAC 2020].

One of the ingredients of our algorithm is a proof that the VC-dimension of (the set system defined by the closed neighborhoods of the vertices of) a weakly γ-closed graph is upper bounded by 6γ. This result may find further applications in the study of weakly closed graphs.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Dominating Set, Weakly Closed Graphs, FPT, Domination Cores, VC-dimension

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.29

Funding The authors are supported by BSF award 2018302 and NSF award CCF-2008838.

Acknowledgements We thank Saket Saurabh for insightful feedback on the manuscript.

1 Introduction

A dominating set of a graph $G = (V, E)$ is a set $S \subseteq V$ of vertices of G such that every vertex in $V \backslash S$ is adjacent to at least one vertex in S. In the Dominating Set problem, the input is a graph G and a positive integer k and the task is to determine whether G has a dominating set of size at most k. Dominating Set is NP-complete and has been extensively studied within all established paradigms for coping with NP-hardness such as parameterized complexity, approximation algorithms and exact exponential time algorithms [9, 13, 19, 31]. In fact, it is hard to overstate the pivotal role that Dominating Set has played in the development of parameterized complexity; it was, together with Clique, one of the first examples of natural parameterized problems that were proved intractable [13] as well as FPT-inapproximable [6, 8, 18].

While, on the one hand, Dominating Set on general graphs has been a driver of parameterized intractability, on the other hand, the study of Dominating Set on restricted graph classes has been a treasure trove of algorithmic techniques. For instance, the subexponential time algorithms for Dominating Set on planar graphs [1, 7], and the linear kernel [2] on planar graphs led to the celebrated bidimensionality theory [11]. These algorithms and kernels have been extended to much wider classes of graphs, such as, (topological) minor free graphs [20], nowhere dense graphs [10, 14], d-degenerate graphs [3, 27], $K_{i,j}$-free graphs [27]

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 29; pp. 29:1–29:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and induced ladder-free graphs [17]. In this article we study the Dominating Set problem on c-closed graphs and weakly γ-closed graphs, which were recently introduced by Fox et al. [21].

▶ **Definition 1** ([21]). *A graph G is said to be* ***c-closed*** *if for every pair of non-adjacent vertices u and v in G, $|N_G(u) \cap N_G(v)| < c$. A graph G is said to be* ***weakly γ-closed*** *if for every induced subgraph G' of G there exists a vertex v in G' such that for every vertex u in G' not adjacent to v, $|N_{G'}(u) \cap N_{G'}(v)| < \gamma$. The* ***closure*** *of a graph G is the smallest c such that G is c-closed. The* ***weak closure*** *of a graph G is the smallest γ such that G is weakly γ-closed.*

The class of c-closed and weakly γ-closed graphs contains the class of graphs of maximum degree at most c and graphs with degeneracy at most γ, respectively. Additionally they capture the triadic closure principle, namely that two people who have many common friends in a social network are likely to be friends themselves. From an application viewpoint, the weak closure is typically found to be small for large real-world social network graphs [21, 23]. In addition, the parameters also have the appealing feature that they are computable in polynomial time [21].

Motivated by the salient features of (weakly) closed graphs, Koana et al. [24] initiated a systematic study of the parameterized complexity of computational problems on c-closed graphs, closely followed by Husic and Roughgarden [22]. Koana et al. [24] show that a number of problems, including Dominating Set, are FPT on closed graphs. In a follow up work Koana et al. [23] show that a number of problems remain FPT even on weakly closed graphs. Very recently, the same set of authors [25] provide polynomial kernels and kernel lower bounds for various problems including Connected Vertex Cover and Capacitated Vertex Cover on weakly closed graphs. They also obtain polynomial kernels for Dominating Set on weakly closed split graphs and weakly closed bipartite graphs. However, they were not able to obtain an FPT algorithm for Dominating Set on weakly closed graphs, leading them to pose the existence of such an algorithm as an open problem. Specifically, Koana et al. [23] asked whether the following parameterized problem is FPT or not.

Dominating Set in weakly γ-closed graph **Parameter:** γ, k
Input: Weakly γ-closed graph G and a non-negative integer k.
Question: Does there exist a set $X \subseteq V(G)$ of size at most k such that $N_G[X] = V(G)$.

In this work, we give an algorithm with running time $k^{O(\gamma^2 k^3)} n^{O(1)}$, resolving the problem in the affirmative. We now state our main result.

▶ **Theorem 2.** *There exists a deterministic algorithm that given as input a weakly γ-closed graph G and an integer k determines in time $k^{O(\gamma^2 k^3)} n^{O(1)}$ whether G has a dominating set of size at most k and outputs one if it exists.*

Methods. Our algorithm is based on domination cores, first defined by Dawar and Kreutzer [10] and then later employed in multiple settings [14, 15, 17]. A *k-domination core* of a graph is a set X of vertices of the graph such that every set of size at most k that dominates X dominates the whole graph. Observe that the set of all vertices of a graph is a domination core. It is well known (for example see [10] Lemma 4.1) that if one can efficiently compute a domination core whose size is upper bounded by a function of k, then we can obtain an FPT algorithm for Dominating Set. Thus our main technical contribution is an algorithm that given a graph produces a k-domination core of the graph of size $k^{O(\gamma k^2)}$.

We now give a very rough sketch of the proof for our main technical claim – every domination core W of size at least b, where $b = k^{O(\gamma k^2)}$ contains at least one vertex w such that $W \setminus \{w\}$ is also a domination core, and that such a vertex w can be found efficiently. In this exposition we focus only on the claim of existence of w. Suppose such a vertex w does not exist. Then, for every vertex $w \in W$ there must exist a set X_w of size at most k that dominates all of $W \setminus \{w\}$, but does not dominate w – otherwise $W \setminus \{w\}$ is still a domination core. We call a set W that has this property a *k-threshold set*[1] and prove that a weakly γ-closed graph can not contain a k-threshold set of size at least b.

The advantage of shifting our attention from k-domination cores to k-threshold sets is that k-threshold sets are closed under subsets – every subset of a k-threshold set is also a k-threshold set. This allows us to "dig for structure", that is, prove results of the form "if G has a sufficiently large k-threshold set W then W contains a large (as a function of k and $|W|$) k-threshold set W' with some additional property".

By invoking a (multi-color version of the) Ramsey Theorem [4] on an appropriately constructed auxiliary graph, we extract from W a sufficiently large and sufficiently symmetric threshold set $W' \subseteq W$. The existence of a large and symmetric threshold set W' in turn implies that G must contain as an induced subgraph one of three simple pattern graphs (such as a complete bipartite graph with $\gamma + 1$ vertices on both sides). Each one of these three pattern graphs can easily be shown not to be weakly γ-closed, contradicting that G was weakly γ-closed in the first place.

We remark that the actual proof proceeds in a different order of the exposition above. First, in Section 3 we define the pattern graphs that we will use and show that they are not weakly γ-closed. In Section 5 we prove that a purely existential upper bound on the size of k-threshold sets implies both an FPT algorithm to find a small k-domination core, and an FPT algorithm for Dominating Set. In Section 6 we obtain the aforementioned upper bound on the size of k-threshold sets in weakly γ-closed graphs by showing that a k-threshold set of size at least $b = k^{O(\gamma k^2)}$ implies that G must contain one of the forbidden pattern graphs from Section 3.

Efficiently computing a domination core W of size $k^{O(\gamma k^2)}$ immediately leads to a $2^{k^{O(\gamma k^2)}} n^{O(1)}$ time algorithm for Dominating Set on weakly γ-closed graphs. Indeed, to find a dominating set for G of size k (if one exists), it is sufficient to find a set S of size at most k that dominates all of W. This can be done by trying all possible partitions of W into k parts $P_1, \ldots, P_k$, and then determining whether there exists for every part P_i a single vertex $s_i \in V(G)$ that dominates P_i. This algorithm already resolves the open problem of Koana et al. [23] in the affirmative. At the same time the double exponential running time dependence on k is unsatisfactory.

We are able to improve the running time of our algorithm for Dominating Set to $k^{O(\gamma^2 k^3)} n^{O(1)}$ by proving an additional purely graph-theoretic result regarding the structure of weakly γ-closed graphs. A *set system* $(U, \mathcal{F})$ consists of a universe U along with a collection $\mathcal{F}$ of subsets of U. A subset containing $A \subseteq U$ is *shattered* by $\mathcal{F}$ if each subset of A can be expressed as the intersection of A with a set in $\mathcal{F}$. The *Vapnik-Chervonenkis* dimension (*VC-dimension*) of a set system is the cardinality of the largest subset A of U that is shattered by $\mathcal{F}$. The VC-dimension of a graph is defined as the VC-dimension of the set system induced by the closed neighbourhoods of its vertices. We prove in Section 4 that weakly γ-closed graphs have VC-dimension at most 6γ.

[1] Note that a k-threshold set is not necessarily a k-domination core, however every inclusion minimal k-domination core is a k-threshold set.

▶ **Theorem 3.** *Every weakly γ-closed graph has VC-dimension at most 6γ.*

Theorem 3 is tight up to the constant factor 6 (see Section 4 for a simple construction of a weakly γ-closed graph with VC-dimension γ).

Theorem 3 (together with our bound on the size of k-threshold sets) quite directly leads to a $k^{O(\gamma^2 k^3)} n^{O(1)}$ time algorithm for Dominating Set on weakly γ-closed graphs. Indeed, the double exponential running time of the previous algorithm came from the algorithm to determine whether there exists a set S of size at most k that dominates the entire domination core W. The size of the k-domination core W is assumed to be upper bounded by $k^{O(\gamma k^2)}$. Our improved algorithm to find S is remarkably simple: if two vertices u and v not in W have exactly the same set of neighbors in W, we remove u from the graph (since we can always pick v in its place). After this reduction, the Sauer-Shelah Lemma [28, 29] (See Lemma 8) implies that there are at most $k^{O(\gamma^2 k^2)}$ vertices left in G. Then a brute force algorithm that tries all possibilities for S takes time $k^{O(\gamma^2 k^3)} n^{O(1)}$.

We believe that Theorem 3 will find further uses in the design of algorithms for problems on weakly γ-closed graphs. For an example Theorem 3 also immediately implies that the improved approximation algorithm for Dominating Set on graphs of bounded VC-dimension [5, 16] applies to weakly γ-closed graphs (see Section 4 for details).

2 Notation and Preliminaries

In this section we give notations, and definitions that we use throughout the paper. Unless specified we will be using all general graph terminologies from the book of Diestel [12].

Given a graph G, we use $V(G)$ and $E(G)$ to denote the set of vertices and edges, respectively. We denote the open neighbourhood of a vertex v in G by $N_G(v) = \{u : u \in V(G), (u,v) \in E(G)\}$ and closed neighbourhood by $N_G[v] = \{v\} \cup N_G(v)$. Further, we denote the non-neighbourhood of v by $\overline{N_G[v]} = V(G) \setminus N[v]$. We extend this notation to a set $S \subseteq V(G)$ as well, that is $N_G(S) = \bigcup\limits_{v \in S} N_G(v)$, $N_G[S] = \bigcup\limits_{v \in S} N_G[v]$ and $\overline{N_G[S]} = V(G) \setminus N_G[S]$. Whenever the graph G is clear from the context, we will omit the subscript. A *dominating set* of G is a set of vertices $S \subseteq V(G)$ such that $N[S] = V(G)$. For any $X \subseteq V(G)$, we use the notation $G[X]$ to denote the subgraph induced by X in G.

We use the symbol $\uplus$ to denote the disjoint union operation on sets. Let l be a positive integer. We use the notation $[l]$ to denote the set $\{1, \dots, l\}$. A graph G having vertex set $V(G) = A \uplus B$ is called a *split graph* if A is a clique and B is an independent set. A graph G is *d-degenerate* if every subgraph G' of G has a vertex having degree at most d. We will need the notion of weak ordering of a weakly γ-closed graph. It is very similar to notion of degeneracy ordering for degenerate graphs [12].

▶ **Definition 4** ([21])**.** *A* ***weak ordering*** *O of a weakly γ-closed graph G is an ordering $O = \{v_1, \dots, v_n\}$ of $V(G)$ such that for each $v_i \in V(G)$ and for each $u \in \overline{N_{G_i}[v_i]}$, it holds that $|N_{G_i}(u) \cap N_{G_i}(v_i)| < \gamma$, where $G_i = G[\{v_i, \dots, v_n\}]$. A* ***forward neighbour*** *of v_i is a vertex adjacent to v_i in G_i.*

3 Obstructions to Weak Closure

In this section, we define a few simple pattern graphs and proceed to show that they (except *split half-graphs*, which are weakly 1-closed) are not weakly γ-closed. Many of our proofs are of the form "every weakly γ-closed graph G either has some desirable property or contains one of these patterns. The second case contradicts that G is weakly γ-closed, so we conclude that G has the desirable property".

▶ **Definition 5.**[2] *Given a positive integer n, let $A = \{a_1, \dots, a_n\}$, $B = \{b_1, \dots, b_n\}$ and $C = \{c_1, \dots, c_n\}$ be disjoint vertex sets. We define the following graphs:*

1. *A bipartite graph G with vertex set $V(G) = A \uplus B$ and bipartition A and B is called a **complete bipartite graph** of order n if $\forall i, j \in [n]$, $(a_i, b_j) \in E(G)$.*
2. *A graph G with vertex set $V(G) = A \uplus B$ is called a **semi split co-matching** of order n if A is a clique and $\forall i, j \in [n]$, $(a_i, b_j) \in E(G)$ iff $i \neq j$. The edges between B can be arbitrary.*
3. *A graph G with vertex set $V(G) = A \uplus B$ is called a **split half graph** of order n if G is a split graph with B being the independent set and $\forall i, j \in [n]$, $(a_i, b_j) \in E(G)$ iff $j > i$.*
4. *A graph G with vertex set $V(G) = A \uplus B \uplus C$ is called a **double split half graph** of order n if $G[A \cup B]$ and $G[B \cup C]$ are split half graphs with B being the independent set. That is $\forall i, j \in [n]$, $(a_i, b_j) \in E(G)$ iff $j > i$ and $(b_i, c_j) \in E(G)$ iff $j > i$. The edges between A and C can be arbitrary.*

▶ **Lemma 6.**[3] *If G is weakly γ-closed, then it does not contain any of the following graphs as an induced subgraph.*

(i) *Complete bipartite graph of order $n \geq \gamma$.*
(ii) *Semi split co-matching of order $n > \gamma$.*
(iii) *Double split half graphs of order $n \geq 3\gamma$.*

4 VC-dimension of Weakly Closed Graphs

In this section we prove Theorem 3, that is we show that the VC dimension of weakly γ-closed graphs is at most 6γ. Recall that the VC-dimension of a graph is defined as the VC-dimension of the set system induced by the closed neighbourhoods of its vertices.

Proof of Theorem 3. Suppose that the VC dimension of G is greater than 6γ. We will show that G is not weakly closed, thus contradicting our assumption. Since we assumed that the VC dimension is at least $6\gamma + 1$, there is a set $X \subseteq V(G)$ of size $6\gamma + 1$ that is shattered in G. Since X is shattered, for each $x \in X$, there exists a vertex y that dominates all vertices in X except x. We note that for each $x \in X$, there can be more than one such vertex but we need only one for our proof. We will call y the partner of x and x the partner of y. Observe that no two vertices in X can have the same partner. Let Y be the set of partners of all vertices in X. Also observe that every $x \in X$ dominates all vertices in Y except its partner and every $y \in Y$ dominates all vertices in X except its partner. We start by extracting a sufficiently large clique from X or Y.

▷ Claim 7. There exists a clique Z of size at least $\gamma + 1$ such that $Z \subseteq X$ or $Z \subseteq Y$.

Proof. Let X_1 be an arbitrary subset of X of size 3γ, and let Y_1 be an arbitrarily chosen set of 3γ vertices in Y that have no partner in X_1. If $|X_1 \cap Y_1| > \gamma$ then $Z = X_1 \cap Y_1$ is a clique that satisfies the conclusion of the lemma since every vertex in Y_1 dominates X_1.

We proceed with the case that $|X_1 \cap Y_1| \leq \gamma$. Define $X' = X_1 \backslash Y_1$ and $Y' = Y_1 \backslash X_1$ (i.e. remove common vertices from X_1 and Y_1). Note that $|X'| \geq 2\gamma$ and $|Y'| \geq 2\gamma$, that X' and Y' are disjoint, and that every vertex in X' is adjacent to every vertex in Y'. Let $O_{X' \cup Y'}$ be the order induced by a weak ordering O of G on $X' \cup Y'$. There must be $\gamma + 1$ vertices all from either X' or Y' among the first $2\gamma + 1$ vertices in $O_{X' \cup Y'}$. Let Z be the set of

[2] Refer to Figure 1 in Appendix A.
[3] Proof in Appendix A.

these $\gamma+1$ vertices all from X' or Y'. Since all vertices in X' are adjacent to all vertices in Y', every pair of vertices in Z must have at least γ common forward neighbours in the ordering O. Thus, since G is weakly γ-closed, Z must be a clique. ◁

Let Z be a clique as provided by Claim 7. Let P_Z be the set of partners of all vertices in Z. Observe that Z and P_Z are disjoint: for every $z \in Z$ its partner z' does not dominate z (by definition of partners) and therefore cannot be in Z, since Z is a clique. Next, we observe that every vertex z in Z is adjacent to every vertex in P_Z except its partner by the definition of X and Y and the fact that $Z \subseteq X$ or $Z \subseteq Y$. The induced subgraph $G[Z \cup P_Z]$ is a semi split co-matching (Definition 5) because Z is a clique, Z and P_Z are disjoint and every vertex z in Z is adjacent to every vertex in P_Z except its partner. This contradicts Lemma 6, concluding the proof. ◀

Theorem 3 is tight up to the constant factor 6, since there exists a weakly γ-closed graph having VC-dimension γ: Consider the bipartite graph G with $V(G) = A \uplus B$ where A has γ vertices and for each set $S \subseteq A$, B has one vertex whose neighbourhood is S. The graph G is weakly γ-closed and has VC-dimension at least γ since A is shattered by the closed neighborhood of vertices in B.

4.1 Set Cover and graphs of bounded VC-dimension

In the Set Cover problem, we are given a universe U, a family $\mathcal{F}$ of sets over U, and a positive integer k and the task is to determine whether there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k such that $\bigcup_{X \in \mathcal{F}'} X = U$. It is known [28, 29] that if the VC-dimension of a set system $(U, \mathcal{F})$ is bounded, then the size of the family $\mathcal{F}$ must be bounded.

▶ **Lemma 8** (Sauer-Shelah lemma [28, 29]). *If the VC-dimension of a set system $(U, \mathcal{F})$ is bounded by d, then $\mathcal{F}$ can consist of at most $\sum_{i=0}^{d} \binom{|U|}{i} = O(|U|^d)$ sets.*

We will exploit the fact that weakly closed graphs have bounded VC-dimension in the following way. Dominating Set on a graph of bounded VC-dimension corresponds to Set Cover on the set system $(U, \mathcal{F})$ where $U = V(G)$ and $\mathcal{F} = \{N[v] \ : \ v \in U\}$.

For a general set system $(U, \mathcal{F})$, there is a naive algorithm that goes over all families $\mathcal{F}'$ of size at most k in $\mathcal{F}$ and checks whether $\mathcal{F}'$ is a set cover in time $|\mathcal{F}|^k |U|^{O(1)}$. However if the VC-dimension of $(U, \mathcal{F})$ is bounded by d, then by Lemma 8, $|\mathcal{F}| = O(|U|^d)$ and therefore this algorithm solves Set Cover in $O(|U|^{kd})$ time.

▶ **Theorem 9.** *There exists a deterministic algorithm that given a Set Cover instance $(U, \mathcal{F}, k)$ such that the VC-dimension of $(U, \mathcal{F})$ is bounded by d determines in time $O(|U|^{kd})$ whether the instance has a set cover of size at most k and outputs one if it exists.*

We remark that this is not an FPT algorithm parameterized by k and d. However we will be invoking Theorem 9 with $|U|$ bounded by $2^{poly(k)}$ and d bounded by 6γ in our algorithm for Dominating Set.

An upper bound on the VC-dimension of G also leads to an improved approximation algorithm for Dominating Set. Indeed Brönnimann and Goodrich [5] give an $O(d \log(dk))$ approximation algorithm for set systems of VC-dimension d, where k is the size of the optimal solution. This, together with Theorem 3 directly yields an $O(\gamma \log(\gamma k))$-approximation for Dominating Set on weakly γ-closed graphs.

5 Dominating Set in Weakly Closed Graphs

Our algorithm is based on *domination cores*, which have been used for deriving several algorithms for the Dominating Set problem [14, 15, 17].

▶ **Definition 10.** *Given a graph G, an integer k, a set $S \subseteq V(G)$ is called a* ***k-domination core*** *of G if $\forall X \subseteq V(G)$ such that $|X| \leq k$ and $S \subseteq N[X]$, it holds that $N[X] = V(G)$.*

It is easy to see that the set of all vertices in a graph is a trivial domination core. We wish to prove that weakly γ-closed graphs contain k-domination cores whose size is upper bounded by a function of k and γ. This naturally leads our attention to inclusion minimal k-domination cores.

▶ **Definition 11.** *A k-domination core W is called a* ***minimal k-domination core*** *if $\forall w \in W$, $W \backslash \{w\}$ is not a k-domination core.*

We note that whenever k is clear from the context, we will omit k while referring to domination cores. In the following lemma, we provide a bound on the size of minimal domination cores in weakly γ-closed graphs.

▶ **Lemma 12.** *Every minimal k-domination core of a weakly γ-closed graph G has size at most b, where $b = k^{O(\gamma k^2)}$.*

Lemma 12 leads to the following intuitive algorithm - Start with the trivial domination core $D = V$ and as long as $|D| > b$ keep discarding a vertex x from D such that D remains a domination core (we will soon discuss how to algorithmically identify the vertex x, for now ignore this issue).

Finally use D to construct a Set Cover instance having universe D and family $\mathcal{F} = \{N[v] \cap D : v \in V(G)\}$. Since G is weakly γ-closed, by Theorem 3, G has VC-dimension at most 6γ. Thus, the Set Cover instance also has VC-dimension at most 6γ and so we use Theorem 9 to find a set cover of size at most k if it exists from which a dominating set for G can be easily recovered.

We now turn to the issue of identifying a vertex x to remove from D when $|D| > b$. To this end, we will use the following property of every minimal k-domination core W: for each $w \in W$, there is a set X_w of size at most k that dominates all of $W \backslash \{w\}$ but not w. Indeed, suppose there is a $w \in W$ for which no such X_w exists, and consider a set X of size at most k which dominates $W \backslash \{w\}$. Then X also dominates w (by the non-existence of X_w) and by extension all of G (since W is a k-domination core). But then $W \backslash \{w\}$ is also a domination core, contradicting minimality. We capture this property in the following definition.

▶ **Definition 13.** *A vertex set S is a* ***k-threshold set*** *if for every $v \in S$ there exists a set X_v of size at most k so that $N[X_v] \cap S = S \backslash \{v\}$.*

Also note that every subset S' of a k-threshold set S is also a k-threshold set because for every $v \in S'$, a set X_v of size at most k such that $N[X_v] \cap S = S \backslash \{v\}$ also satisfies $N[X_v] \cap S' = S' \backslash \{v\}$ as $S' \subseteq S$. We will use this property explicity in the next section. For now, the discussion leading up to Definition 13 immediately leads to the following observation.

▶ **Observation 14.** *Every minimal k-domination core of a graph G is also a k-threshold set of G.*

Since every minimal k-domination core is a k-threshold set, we will bound the size of k-threshold sets in weakly γ-closed graphs, proving Lemma 12 and leading to an algorithm.

▶ **Lemma 15.** *Every k-threshold set of a weakly γ-closed graph G has size at most b, where $b = k^{O(\gamma k^2)}$*

We now outline how Lemma 15 can be used to identify a vertex x to be removed from a domination core D having size more than b such that $D\backslash\{x\}$ still remains a domination core. No subset of D having size $b + 1$ can be a threshold set because of Lemma 15. Thus, we can pick an arbitrary subset X of D having size $b + 1$ and for each $x \in X$, test whether $X\backslash\{x\}$ has a dominating set of size at most k without dominating x. Since X is not a threshold set, we will find a vertex $x \in X$ for which such a dominating set does not exist. Thus, we can remove x from D and $D\backslash\{x\}$ will still remain a domination core.

We are now ready to patch up our ideas and provide the full algorithm to prove Theorem 2 assuming Lemma 15 is true. We dedicate the next section solely for the proof of Lemma 15.

Proof of Theorem 2 (assuming the statement of Lemma 15). We first provide the algorithm: Initialize $D = V(G)$. As long as $|D| > b$, arbitrarily pick a subset X of D having size $b + 1$. For each $x \in X$, construct a Set Cover instance $I_x = (U_x, \mathcal{F}_x, k)$ with universe $U_x = X\backslash\{x\}$ and family $\mathcal{F}_x = \{N[y] \cap X\backslash\{x\} : y \in \overline{N[x]}\}$. Solve I_x using Theorem 9. If I_x is a no instance, set $D = D\backslash\{x\}$ and proceed to start of the loop.

After the loop terminates, construct the Set Cover instance $I = (U, \mathcal{F}, k)$ where $U = D$ and $\mathcal{F} = \{X_v = N[v] \cap D : v \in V(G)\}$. Use Theorem 9 to find a set cover $\mathcal{S} \subseteq \mathcal{F}$ having size at most k if exists for I. Return no and terminate the algorithm if I is a no instance. If I is a yes instance, return the set $D' = \{v : X_v \in \mathcal{S}\}$.

▷ Claim 16. During each iteration of the loop, the algorithm finds a vertex x to remove from D.

Proof. Consider an arbitrary iteration of the loop. It is clear that $|D| > b$ since the algorithm enters the loop. Observe that no subset of D having size $b + 1$ can be a k-threshold set by Lemma 15. Let X be the subset of D picked by the algorithm in that iteration, it is clear that X is not a k-threshold set. Thus, by definition of a k-threshold set, there exists a vertex $x \in X$ for which $X\backslash\{x\}$ does not have a dominating set of size at most k that does not dominate x. It is also easy to see that $X\backslash\{x\}$ has a dominating set of size at most k not dominating x if and only if I_x has a set cover of size at most k. Thus, there is a vertex $x \in X$ for which I_x does not have a set cover of size at most k. Therefore, the algorithm would have removed at least one element from D in that iteration. ◁

▷ Claim 17. In each iteration of the algorithm, D is a domination core.

Proof. Since the set of all vertices of G is itself a trivial domination core, the algorithm starts with a domination core $D = V(G)$. Let X be the subset of D of size $b + 1$ picked by the algorithm in that iteration. Also let $x \in X$ be the vertex removed from D in that iteration. By the previous claim, such an x exists. Since the algorithm removed x from D, the set cover instance I_x must have been a no instance. It is easy to see that I_x is a no instance if and only if $X\backslash\{x\}$ does not have a dominating set of size at most k without dominating x. Thus, since every set of size at most k dominating $D\backslash\{x\}$ will dominate $X\backslash\{x\}$ which in turn will dominate x, $D\backslash\{x\}$ is a k-domination core.

Thus, in each iteration of the algorithm, D is a k-domination core. ◁

Now consider D in the last step of the algorithm. The algorithm reaches this step because of the first claim. It is easy to see that I is an yes instance if and only if D has a dominating set of size at most k. Since D is a domination core by the previous claim, this implies that G has dominating set of size at most k if and only if I is a yes instance. Thus, the algorithm returns a dominating set of G of size at most k if one exists, otherwise returns no. Namely, the recovered set D' is a dominating set of G.

For the runtime, the time taken to identify a vertex to remove from D when $|D| > b$ is $b^{O(\gamma k)}$ using Theorem 9 as $|U_x| = b$ and the VC-dimension of the set system $(U_x, \mathcal{F}_x)$ is bounded by 6γ by Theorem 3. This step is repeated at most $n - b$ times. The final step to find the dominating set again takes $b^{O(\gamma k)}$ time since in the last step D has size at most b. Thus, in total the algorithm takes $b^{O(\gamma k)} n^{O(1)}$ time which is $k^{O(\gamma^2 k^3)}$. ◀

6 Threshold Sets in Weakly Closed Graphs

In this section, we prove the crux of our algorithm, namely Lemma 15 which bounds the size of threshold sets in weakly γ-closed graphs. We first begin by stating that the graph induced by any k-threshold set of a weakly γ-closed graph is sparse.

▶ **Lemma 18.**[4] *Given a weakly γ-closed graph G and k-threshold set S of G, $G[S]$ is $(\gamma - 1)k$-degenerate.*

Since every d-degenerate graph on n vertices has an independent set of size at least $n/(d+1)$ [12], any large k-threshold set will also have a large independent set. This leads us to define the following notion.

▶ **Definition 19.** *A k-threshold set S of a graph G is called an **independent k-threshold set** of G if S is an independent set.*

Further, since every k-threshold set S of a weakly γ-closed graph has an independent set of size at least $\frac{|S|}{(\gamma-1)k+1}$ and since every subset of a k-threshold set is also a k-threshold set, we obtain the following result.

▶ **Lemma 20.** *Every k-threshold set S of a weakly γ-closed graph has an independent k-threshold set of size at least $\frac{|S|}{(\gamma-1)k+1}$.*

By the previous lemma, it is clear that to bound the size of threshold sets in weakly closed graphs, it is enough to bound the size of independent threshold sets. This fact along with Lemma 21 stated below combined prove Lemma 15[5].

▶ **Lemma 21.** *Every independent k-threshold set of a weakly γ-closed graph G has size at most $k^{O(\gamma k^2)}$.*

We prove Lemma 21 by contradiction. Assuming that G has a large independent k-threshold set, we first use results from Ramsey theory to extract a sufficiently large and highly symmetric independent 2-threshold set (this is never proved explitly in the argument). The highly structured independent 2-threshold set implies that G contains one of the obstructions from Lemma 6, contradicting that G is weakly γ-closed.

[4] Proof in Appendix B.
[5] Proof in Appendix C.

Proof of Lemma 21. Let W be an independent k-threshold set of a weakly γ-closed graph G having size greater than $(3^{15}k^2)^{(3^{16}\gamma k^2)}$. As a first step, we will use results from Ramsey theory to obtain three subsets of vertices of G having useful properties. We will then use these sets to show that G has one of the graphs listed in Lemma 6 as an induced subgraph. By Lemma 6, this will imply that G is not a weakly γ-closed graph, contradicting our assumption and thus completing the proof.

Since W is a k-threshold set of G, for every vertex $w \in W$ there exists a set $X_w \subseteq V(G)$ of size at most k that dominates all vertices in W except w. For each $w \in W$, order the vertices in X_w arbitrarily. Let $X_w = \{x_w^1, \ldots, x_w^{p_w}\}$ be the ordering. Also order the vertices in W arbitrarily. Let $W = \{w_1, \ldots, w_q\}$ be the ordering.

We now create an auxiliary edge-colored complete graph H with vertex set W. Each color will be a tuple[6] whose size and possible values will become clear in the next step where we assign colors to the edges.

For every pair $i, j \in [|W|]$ such that $i < j$, we color the edge (w_i, w_j) in H as follows:

1. One entry for the number r such that $x_{w_i}^r$ dominates w_j (if more than one such r exists, choose one arbitrarily)
2. One entry for the number s such that $x_{w_j}^s$ dominates w_i (if more than one such s exists, choose one arbitrarily)
3. For each pair[7] of vertices in the multi-set $\{w_i, w_j, x_{w_i}^r, x_{w_j}^s, x_{w_j}^r, x_{w_i}^s\}$ one entry from $\{0, 1, 2\}$ to denote whether those two vertices are (0) the same vertex (1) different and adjacent vertices or (2) different and non-adjacent vertices.

From the definition of H, it follows that the number of possible distinct edge-colors of H is at most $3^{15}k^2$. Let $B \subseteq W$ be a monochromatic clique of maximum size in H and let τ be the color of all the edges in the clique. We will now use the well known fact (from Ramsey theory [4]) that every edge-colored complete graph on n vertices colored with t colors has a monochromatic clique of size at least $\log_t(n)/t$ to lower bound the size of B. Since the number of possible distinct edge-colors of H is at most $3^{15}k^2$ and the size of W is greater than $(3^{15}k^2)^{(3^{16}\gamma k^2)}$, the size of B is at least 3γ.

Let $B = \{b_1, \ldots, b_l\}$ be the ordering of vertices of B in W. Let r and s be the two entries in τ that denote the numbers such that for every pair $i, j \in [l]$ having $i < j$, $x_{b_i}^r$ dominates b_j and $x_{b_j}^s$ dominates b_i. Let $A = \{x_{b_1}^r, \ldots, x_{b_l}^r\}$ and $C = \{x_{b_1}^s, \ldots, x_{b_l}^s\}$ be ordered multi-sets. For now, we will assume that A and C could be multi-sets but we will soon prove that it is not the case. We now capture some desired properties of A, B and C.

▷ Claim 22. The multi-sets $B = \{b_1, \ldots, b_l\}$, $A = \{x_{b_1}^r, \ldots, x_{b_l}^r\}$, and $C = \{x_{b_1}^s, \ldots, x_{b_l}^s\}$ satisfy the following properties:

1. B is an independent set in G.
2. A, B and C are sets.
3. $A \cap B = \emptyset$, $B \cap C = \emptyset$ and either $A \cap C = \emptyset$ or $A = C$.
4. $\forall i \in [l]$, $(b_i, x_{b_i}^r) \notin E(G)$ and $(b_i, x_{b_i}^s) \notin E(G)$.
5. $\forall i, j \in [l]$ such that $j > i$, $(x_{b_i}^r, b_j) \in E(G)$ and $(b_i, x_{b_j}^s) \in E(G)$.
6. A and C are each either an independent set or a clique in G.
7. $\forall i, j \in [l]$, such that $j < i$, $(x_{b_i}^r, b_j) \in E(G)$ or $\forall i, j \in [l]$, such that $j < i$ $(x_{b_i}^r, b_j) \notin E(G)$.
8. $\forall i, j \in [l]$, such that $j < i$, $(b_i, x_{b_j}^s) \in E(G)$ or $\forall i, j \in [l]$, such that $j < i$, $(b_i, x_{b_j}^s) \notin E(G)$.

[6] When comparing equality of two edge colors, we compare corresponding entries of the two tuples in the order they are defined. Thus the order of the entries in the tuples matter.

[7] We will not need all 15 pairs in our arguments. The colors are defined in this way to keep the description simple.

Proof. Since $B \subseteq W$ and W is a independent threshold set, it follows that B is an independent set (property 1). We now prove property 2. By definition, B is a subset of W which is a set. Now we prove that for each pair $i, j \in [l]$ such that $i < j$, $x^r_{b_i} \neq x^r_{b_j}$ and $x^s_{b_i} \neq x^s_{b_j}$. Since r and s are entries in the coloring τ, $x^r_{b_i}$ dominates b_j and $x^s_{b_j}$ dominates b_i. But by definition $x^r_{b_j}$ does not dominate b_j and $x^s_{b_i}$ does not dominate b_i. Therefore $x^r_{b_i} \neq x^r_{b_j}$ and $x^s_{b_i} \neq x^s_{b_j}$.

For property 3, we first show that $A \cap B = \emptyset$. We prove that $\forall i, j \in [l]$, $b_i \neq x^r_{b_j}$. If $i = j$, then $b_i \neq x^r_{b_j}$ because by definition $x^r_{b_i}$ belongs to X_i and thus does not dominate b_i. Let $b_i = x^r_{b_j}$ for some $i > j$, then in the coloring τ the entry corresponding to the pair of vertices b_i and $x^r_{b_j}$ must be 0 since they are the same. Thus, since all edges in clique B in H have color τ, it means that $x^r_{b_1} = b_2$ and $x^r_{b_1} = b_3$ but $b_2 \neq b_3$. Thus, $b_i \neq x^r_{b_j}$. Similarly, we can prove that $b_i \neq x^r_{b_j}$ in the case when $i < j$. The proof that $B \cap C = \emptyset$ is symmetric and therefore omitted.

Now, we show that either $A \cap C = \emptyset$ or $A = C$. If $r = s$, then $A = C$. If $r \neq s$, we will show that $\forall i, j \in [l]$, $x^r_{b_i} \neq x^s_{b_j}$. If $i = j$, by the definition of X_{b_i}, it follows that $x^r_{b_i} \neq x^s_{b_i}$. If $i \neq j$, without loss of generality let us consider the case when $i < j$ and a similar argument will hold for the case when $i > j$. If $x^r_{b_i} = x^s_{b_j}$, then by our coloring τ, $x^r_{b_1} = x^s_{b_2}$ and $x^r_{b_1} = x^s_{b_3}$. But $x^s_{b_2} \neq x^s_{b_3}$ by property 2. Thus, $x^r_{b_i} \neq x^s_{b_j}$.

Property 4 is true because $A \cap B = \emptyset$, $B \cap C = \emptyset$ and for each b_i in B, $x^r_{b_i}$ and $x^s_{b_i}$ are in X_i and thus do not dominate b_i. Property 5 follows because $A \cap B = \emptyset$, $B \cap C = \emptyset$ and the fact that r and s are entries in the coloring τ such that $\forall i, j \in [l]$, having $j > i$, $x^r_{b_i}$ dominates b_j and $x^s_{b_j}$ dominates b_i.

Since $A \cap B = \emptyset$ and $B \cap C = \emptyset$, $\forall i, j \in [l]$ such that $j < i$ the coloring τ has an entry with value either 1 or 2 corresponding to each pair in $\{(x^r_{b_i}, x^r_{b_j}), (x^s_{b_i}, x^s_{b_j}), (x^r_{b_i}, b_j), (b_i, x^s_{b_j})\}$. Since (1) denotes that the pair of vertices are adjacent and (2) denotes that the pair of vertices are non-adjacent, properties 6-8 are true. This completes the proof. ◁

We now use the sets (Claim 22 Property 2) A, B, and C to show that G has one of the graphs listed in Lemma 6 as an induced subgraph. For this, we will use the properties listed in Claim 22. We remark that we will directly refer to them as properties rather than referring to the claim each time. Recall that $l = |A| = |B| = |C|$. Firstly, we consider two cases based on whether $A = C$ or not.

Case (i) $A = C$**:** By property 3, A and B are disjoint. We divide this case further into two cases based on property 6 - A is either an independent set or a clique.

(a) A is a clique: Let $G' = G[A \cup B]$. Then, B is an independent set (by property 1) and $\forall i, j \in [l]$ $(x^r_i, b_i) \notin E(G')$ if $i = j$ (by property 4) and $(x^r_i, b_j) \in E(G')$ otherwise (by properties 5 and $A = C$). Thus G' is a semi split co-matching of order $l \geq 3\gamma$.

(b) A is an independent set: Let $A' = \{x^r_{b_1}, \ldots, x^r_{b_\gamma}\}$, $B' = \{b_{\gamma+1}, \ldots, b_{2\gamma}\}$, and $G' = G[A' \cup B']$. Observe that we can define sets A' and B' since $l \geq 3\gamma$. Again, by property 5, $\forall i \in \{1, \ldots, \gamma\}, j \in \{\gamma + 1, \ldots, 2\gamma\}$, $(x^r_{b_i}, b_j) \in E(G')$. Thus G' is a complete bipartite graph of order γ.

Case (ii) $A \neq C$**:** Since $A \neq C$, by property 3 the sets A, B, and C are disjoint. We divide this case further based on properties 6-8.

(a) A is an independent set: Let $A' = \{x^r_{b_1}, \ldots, x^r_{b_\gamma}\}$, $B' = \{b_{\gamma+1}, \ldots, b_{2\gamma}\}$ and $G' = G[A' \cup B']$. We can show that G' is a complete bipartite graph by the same argument as case (i.b).

(b) C is an independent set: Same argument as the previous case with sets $B' = \{b_1, \ldots, b_\gamma\}$, $C' = \{x^s_{b_{\gamma+1}}, \ldots, x^s_{b_{2\gamma}}\}$ and graph $G' = G[B' \cup C']$.

(c) A is a clique and $\forall i \in [l]$, $\forall j < i$, $x^r_{b_i}$ is adjacent to b_j: Similar to case (i.a), $G' = G[A \cup B]$ is a semi split co-matching of order $l \geq 3\gamma$.

(d) C is a clique and $\forall i \in [l]$, $\forall j < i$, b_i is adjacent to $x^s_{b_j}$: Same argument as previous case with $G' = G[B \cup C]$.

(e) A and C are cliques, $\forall i \in [l]$, $\forall j < i$, $x^r_{b_i}$ is not adjacent to b_j and $\forall i \in [l]$, $\forall j < i$, b_i is not adjacent to $x^s_{b_j}$: Let $G' = G[A \cup B \cup C]$. By the case we are in and property 5, it follows that G' is a double split half graph with B being the independent set (property 1).

Thus, in all cases $G[A \cup B \cup C]$ is not weakly γ-closed by Lemma 6, contradicting the assumption that G is weakly γ-closed, and completing the proof of the lemma. ◀

7 Conclusion and Barriers to Further Improvements

In this work we gave an algorithm for Dominating Set with running time $2^{O(\gamma^2 k^3)} n^{O(1)}$. This resolves affirmatively an open problem of Koana et al. [23] who asked whether the problem is fixed-parameter tractable when parameterized by k and the weak closure γ of the input graph. Our running time hides a large constant in the exponent. We made no effort to optimize this constant because, at this point, it is not even clear that the form $O(\gamma^2 k^3)$ of the exponent in the running time is near-optimal.

On the way to obtaining our main result, we proved that every minimal k-domination core of G has size at most $k^{O(\gamma k^2)}$. We also showed that the VC-dimension of a weakly γ-closed graph G is at most 6γ and used this result in our FPT algorithm for Dominating Set and to obtain an $O(\gamma \log(\gamma k))$-approximation for Dominating Set. The bound on VC-dimension might be interesting for other problems on weakly-closed graphs.

Our work leaves the following natural open problem: *does* Dominating Set *admit a kernel of size $k^{f(\gamma)}$ for some function f?* One natural approach would be to improve the bound in Lemma 12 by obtaining a polynomial upper bound for the size of minimal domination cores in weakly closed graphs. Unfortunately, this is not possible: for every positive integer k, there exists a weakly 1-closed graph with a minimal k-domination core of size 2^{k+1} (see Appendix D). Notice that the argument only shows an obstacle for using this approach for getting polynomial kernels and does not rule out the existence of polynomial kernels.

In light of the $O(\gamma \log(\gamma k))$ approximation algorithm from Section 4, it is natural to ask whether Dominating Set could admit for every fixed constant γ a constant factor approximation algorithm on weakly γ-closed graphs. It is known from [30] Theorem 2 that there exists a c such that a polynomial time $c\frac{\log n}{\log \log n}$-approximation algorithm for Dominating Set in $K_{3,3}$-free graphs would imply that $\mathsf{NP} \subseteq \mathsf{DTIME}(2^{n^{1-\varepsilon}})$ for some $0 < \varepsilon < 1/2$. The graphs constructed in the reduction[8] are also weakly 3-closed and hence we get the same result for weakly 3-closed graphs.

References

1 Jochen Alber, Hans L. Bodlaender, Henning Fernau, Ton Kloks, and Rolf Niedermeier. Fixed parameter algorithms for DOMINATING SET and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.

2 Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004.

[8] The reduction is from Set Cover on a set system in which the maximum intersection between any two sets in the family is 1 [30, 26].

3 Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009.

4 Noga Alon and Vojtěch Rödl. Asymptotically tight bounds for some multicolored ramsey numbers.

5 Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discret. Comput. Geom.*, 14(4):463–479, 1995.

6 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-eth to fpt-inapproximability: Clique, dominating set, and more. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 743–754. IEEE Computer Society, 2017.

7 Jianer Chen, Henning Fernau, Iyad A. Kanj, and Ge Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. *SIAM J. Comput.*, 37(4):1077–1106, 2007.

8 Yijia Chen and Bingkai Lin. The constant inapproximability of the parameterized dominating set problem. *SIAM J. Comput.*, 48(2):513–533, 2019.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

10 Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In Ravi Kannan and K. Narayan Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, volume 4 of *LIPIcs*, pages 157–168. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009.

11 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

12 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

13 Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science and Business Media, 2012.

14 Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

15 Eduard Eiben, Mithilesh Kumar, Amer E. Mouawad, Fahad Panolan, and Sebastian Siebertz. Lossy kernels for connected dominating set on sparse graphs. *SIAM J. Discret. Math.*, 33(3):1743–1771, 2019.

16 Guy Even, Dror Rawitz, and Shimon Shahar. Hitting sets when the vc-dimension is small. *Inf. Process. Lett.*, 95(2):358–362, 2005.

17 Grzegorz Fabianski, Michal Pilipczuk, Sebastian Siebertz, and Szymon Torunczyk. Progressive algorithms for domination and independence. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, pages 27:1–27:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

18 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.

19 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

20 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Kernels for (connected) dominating set on graphs with excluded topological minors. *ACM Trans. Algorithms*, 14(1):6:1–6:31, 2018.

21 Jacob Fox, Tim Roughgarden, C. Seshadhri, Fan Wei, and Nicole Wein. Finding cliques in social networks: A new distribution-free model. *SIAM J. Comput.*, 49(2):448–464, 2020.

22 Edin Husic and Tim Roughgarden. FPT algorithms for finding dense subgraphs in c-closed graphs. *CoRR*, abs/2007.09768, 2020. `arXiv:2007.09768`.

23 Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Computing dense and sparse subgraphs of weakly closed graphs. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 20:1–20:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

24 Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Exploiting c-closure in kernelization algorithms for graph problems. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 65:1–65:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

25 Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Essentially tight kernels for (weakly) closed graphs. *CoRR*, abs/2103.03914, 2021. `arXiv:2103.03914`.

26 V. S. Anil Kumar, Sunil Arya, and H. Ramesh. Hardness of set cover with intersection 1. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, volume 1853 of *Lecture Notes in Computer Science*, pages 624–635. Springer, 2000.

27 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Solving dominating set in larger classes of graphs: FPT algorithms and polynomial kernels. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 694–705. Springer, 2009.

28 Norbert Sauer. On the density of families of sets. *J. Comb. Theory, Ser. A*, 13(1):145–147, 1972.

29 Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

30 Sebastian Siebertz. Greedy domination on biclique-free graphs. *Inf. Process. Lett.*, 145:64–67, 2019.

31 Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2001.

A Proof of Lemma 6 (Obstructions to Weak Closure)

In order to prove Lemma 6, we will first prove that complete bipartite graphs, semi split co-matchings and double split half graphs (see Figure 1) having order more than γ, γ, and 3γ respectively are not weakly γ-closed.

Let G be a graph, if for a vertex v in G, there exists a non-neighbour u in G such that $|N(u) \cap N(v)| \geq \gamma$, we will refer to u as a *weak-pair* of v. Observe that if u is a weak-pair of v, then v is also a weak-pair of u. To prove that G is not weakly γ-closed, it is enough to show that every vertex in G has a weak-pair.

▶ **Lemma 23.** *If G is a complete bipartite graph of order $n \geq \gamma$, it is not weakly γ-closed.*

Proof. Let $V(G) = \{a_1, \ldots, a_n\} \uplus \{b_1, \ldots, b_n\}$. First, we show that $\forall i, j \in [n]$, having $i \neq j$, a_i is a weak-pair of a_j. It holds that $(a_i, a_j) \notin E(G)$ and $|N(a_i) \cap N(a_j)| \geq \gamma$ since it is a complete bipartite graph and $n \geq \gamma$. Similarly $\forall i, j \in n$, having $i \neq j$, b_i is a weak-pair of b_j. Thus G is not weakly γ-closed. ◀

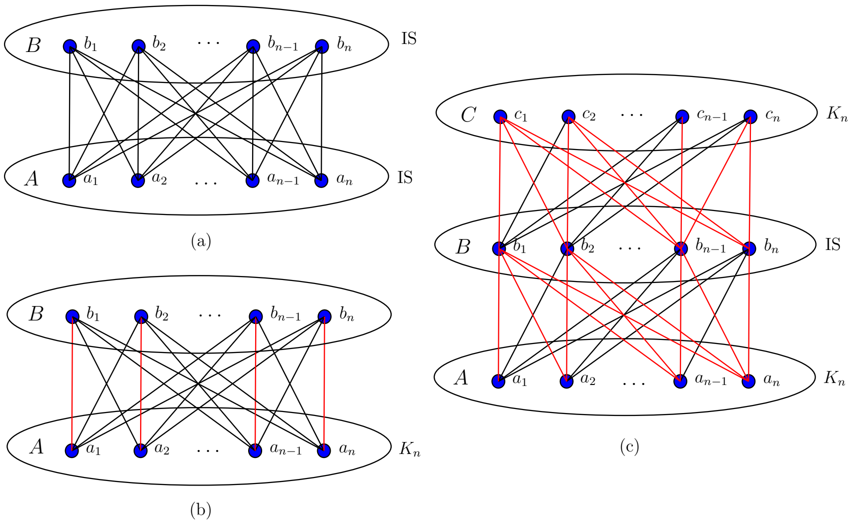


Figure 1 *Sufficiently large (a) complete bipartite graph, (b) semi split co-matching and (c) double split half graph are not weakly γ-closed. Edges are colored black and non-edges are colored red. Note that there may be arbitrary edges between the vertices in B in a semi split co-matching and between the two cliques (A and C) in a double split half graph. On the other hand split half graphs are weakly 1-closed.*

▶ **Lemma 24.** *If G is a semi split co-matching of order $n > \gamma$, it is not weakly γ-closed.*

Proof. Let $V(G) = \{a_1, \ldots, a_n\} \uplus \{b_1, \ldots, b_n\}$. We show that $\forall i \in [n]$, b_i is a weak-pair of a_i and thus a_i is a weak-pair of b_i. Since G is a semi-split co-matching, a_i is not adjacent to b_i and both a_i and b_i are adjacent to all a_j, $j \neq i$. Because $n > \gamma$, $|N(a_i) \cap N(b_i)| \geq \gamma$. Since all vertices in G have a weak pair, it is not weakly γ-closed. ◀

▶ **Lemma 25.** *If G is a double split half graph of order $n \geq 3\gamma$, it is not weakly γ-closed.*

Proof. Let $V(G) = \{a_1, \ldots, a_n\} \uplus \{b_1, \ldots, b_n\} \uplus \{c_1, \ldots, c_n\}$.

First, we will prove that $\forall i \in [n]$, b_i has a weak-pair. Since G is a double split half graph, observe that $\forall i \in [n]$, b_i is adjacent to all c_j, $j > i$ and to all a_j, $j < i$. Thus, since both $G[\{a_1, \ldots, a_n\}]$ and $G[\{c_1, \ldots, c_n\}]$ are cliques and $n \geq 3\gamma$, it follows that either $|N[b_i] \cap N[a_i]| \geq \gamma$ or $|N[b_i] \cap N[c_i]| \geq \gamma$. Hence, since b_i is not adjacent to both a_i and c_i, either a_i or c_i is a weak pair of b_i.

Second, we will prove that $\forall i \in [n]$, a_i has a weak-pair. We divide the proof into two cases: (a) $i > \gamma$ and (b) $i \leq \gamma$.

For case (a), we will show that b_i is a weak-pair of a_i. Since G is a double split half graph, a_i is not adjacent to b_i, b_i is incident to all a_j, $j < i$ and $G[\{a_1, \ldots, a_n\}]$ is a clique. Thus, as we are in the case when $i > \gamma$, it follows that $|N(a_i) \cap N(b_i)| \geq \gamma$. This proves that b_i is a weak-pair of a_i.

For case (b), we will show that either b_i or some c_j, $j > n - \gamma$ is a weak-pair of a_i. If a_i is not adjacent to some c_j, $j > n - \gamma$, then since G is a double split half graph, both a_i and c_j are adjacent to all b_k, $i < k < j$. Since, $n \geq 3\gamma$, $i \leq \gamma$ and $j > n - \gamma$, $|N(a_i) \cap N(c_j)| \geq \gamma$ and thus c_j is a weak-pair of a_i. If a_i is adjacent to all c_j, $j > n - \gamma$. Then again since G is a double split half graph, a_i is not adjacent to b_i and b_i is adjacent to all c_j, $j > n - \gamma$. Thus, it follows that $|N(a_i) \cap N(b_i)| \geq \gamma$ since $n \geq 3\gamma$. This proves that b_i is a weak-pair of a_i.

Finally, we can use a very similar argument to that used for a_is to prove that $\forall i \in [n]$, c_i has weak-pair. But here the two cases will be (a) $i \leq n - \gamma$ and (b) $i > n - \gamma$.

Therefore, since all vertices have a weak-pair, G is not weakly γ-closed. ◀

We give a short proof for lemma 6 using the previous lemmas.

Proof of Lemma 6. By the definition of weakly γ-closed graphs any graph having an induced subgraph that is not weakly γ-closed graph is also not weakly γ-closed. Thus, Lemma 6 follows from all the previous lemmas in this section. ◀

B Proof of Lemma 18

We now prove Lemma 18 which says that given a weakly γ-closed graph G and a k-threshold set S of G, $G[S]$ is $(\gamma - 1)k$-degenerate.

Proof of Lemma 18. Given a weak ordering O of a weakly γ-closed graph G, let the order induced by O on a subset S of vertices of G be denoted by O_S. To complete the proof, it is enough to prove the following claim.

▷ Claim 26. Given a weakly γ-closed graph G, weak ordering O of G and k-threshold set S of G, every vertex in S has forward degree at most $(\gamma - 1)k$ in O_S.

Proof. Suppose the claim was not true. Let u be the first vertex in O_S having more than $(\gamma - 1)k$ forward neighbours in O_S. Let F be the set of forward neighbours of u in O_S. Also, let X be a dominating set of $S \backslash \{u\}$ having size at most k and not dominating u. Since S is a k-threshold set of G, such a set X exists.

Firstly we prove that every vertex $v \in X$ that is not adjacent to u can dominate at most $\gamma - 1$ vertices in F since G is weakly γ-closed. If v is ahead of u in the ordering O, then since no non-neighbour of u can have more than $\gamma - 1$ forward common neighbours with u, v is adjacent to at most $\gamma - 1$ vertices in F. Similarly, if u is ahead of v in O, the same argument holds with respect to v.

Now, since $|F| > (\gamma - 1)k$ and $|X| \leq k$, by pigeon hole principle there is a vertex $v \in X$ that is adjacent to more than $\gamma - 1$ vertices in F. Therefore u must be equal to or adjacent to v as G is weakly γ-closed. Thus, we have reached a contradiction to the fact that X did not dominate u. This completes the proof. ◁

Let O be a weak ordering of G, then by the above claim, O_S is a degeneracy ordering of $G[S]$ with degeneracy $(\gamma - 1)k$. Thus $G[S]$ is a $(\gamma - 1)k$-degenerate graph. ◀

C Proof of Lemma 15

We now give a short proof for Lemma 15, that is we prove that the size of k-threshold sets in weakly γ-closed graphs is at most $k^{O(\gamma k^2)}$.

Proof of Lemma 15. Lemma 20 shows that every k-threshold set S of a weakly γ-closed graph must have an independent k-threshold set of size at least $\frac{|S|}{(\gamma-1)k+1}$. Lemma 21 shows that every independent k-threshold set of a weakly γ-closed graph has size at most $k^{O(\gamma k^2)}$. Combining these two results, we can infer that every k-threshold set of a weakly γ-closed graph must have size at most $k^{O(\gamma k^2)}$. ◀

D Minimal k-domination cores of size 2^k in weakly 1-closed graphs

Consider the graph G obtained by taking a complete binary tree T of depth $k+1$ and making every node adjacent to all its ancestors. The set S of all the nodes in level $k+1$ is a minimal k-domination core.

S is a k-domination core because any vertex adjacent to any vertex v in S is adjacent to all vertices adjacent to v. Thus since $N[S] = V(G)$, any set of size at most k dominating S will dominate $V(G)$ as well.

For every vertex $v \in S$, let A_v be the set of ancestors of v in T and let C_v be the set of all children of all the nodes in A_v in T. Then for each $v \in S$, the set $C_v \backslash (A_v \cup \{v\})$ is a dominating set of $S \backslash \{v\}$ of size k that does not dominate v. Therefore S is minimal.

It is natural to ask whether the example can be strengthened to give a c-closed graph with an exponential size minimal k-domination core. However, it is possible to upper bound the size of minimal k-domination cores in c-closed graphs by ck^{c+1}. We omit the proof of this statement, as it is out of scope for this paper.

Popular Matchings in the Hospital-Residents Problem with Two-Sided Lower Quotas

Meghana Nasre ✉
IIT Madras, Chennai, India

Prajakta Nimbhorkar ✉
Chennai Mathematical Institute, India
UMI ReLaX, Chennai, India

Keshav Ranjan ✉
IIT Madras, Chennai, India

Ankita Sarkar ✉
Dartmouth College, Hanover, NH, USA

Abstract

We consider the hospital-residents problem where both hospitals and residents can have lower quotas. The input is a bipartite graph $G = (\mathcal{R} \cup \mathcal{H}, E)$, each vertex in $\mathcal{R} \cup \mathcal{H}$ has a strict preference ordering over its neighbors. The sets $\mathcal{R}$ and $\mathcal{H}$ denote the sets of residents and hospitals respectively. Each hospital has an upper and a lower quota denoting the maximum and minimum number of residents that can be assigned to it. Residents have upper quota equal to one, however, there may be a requirement that some residents must not be left unassigned in the output matching. We call this as the residents' lower quota.

We show that whenever the set of matchings satisfying all the lower and upper quotas is non-empty, there always exists a matching that is popular among the matchings in this set. We give a polynomial-time algorithm to compute such a matching.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases Matching, Popularity, Lower quota, Preferences

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.30

Funding *Meghana Nasre*: Partially funded by SERB grant CRG/2019/004757.
Prajakta Nimbhorkar: Partially funded by Infosys Grant and SERB grant CRG/2019/004757.
Ankita Sarkar: Part of the work was done when the author was a Master's student at Chennai Mathematical Institute, India.

1 Introduction

The stable marriage problem and its many-to-one generalization, namely the hospital residents (HR) problem, have been extensively investigated in the literature. In this work, we consider a generalization of the HR problem where hospitals and residents both can specify *demand* constraints. More formally, the input to our problem is a bipartite graph $G = (\mathcal{R} \cup \mathcal{H}, E)$ where $\mathcal{R}$ denotes the set of residents, $\mathcal{H}$ denotes the set of hospitals, and an edge $(r, h) \in E$ denotes that r and h are mutually acceptable to each other. Every resident and hospital specify a strict ranking of acceptable elements to them, and this ranking is called the *preference list* of the agent. Every hospital h has two additional inputs associated with it - $q^+(h)$, the capacity or upper quota of h, and $q^-(h)$, the demand or lower quota of h. The upper quota denotes the maximum number of residents that can be matched to h, and the lower quota denotes the minimum number of residents that must be assigned to h in any feasible assignment. In practical scenarios, residents may also have demands. That is, some residents must be matched in a round of assignments of residents (medical interns)

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 30; pp. 30:1–30:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to hospitals. We model this by allowing residents to specify an integral lower quota as a part of the input. Thus, associated with every resident $r \in \mathcal{R}$ we have $q^-(r) \in \{0, 1\}$ and $q^+(r) = 1$. We denote this as the HR2LQ problem. When all residents have lower quota zero, it is denoted as the HRLQ problem, which is well investigated in the literature. A matching M in G is a subset of the edge set E. Our goal for the HR2LQ problem is to compute a *feasible* matching that is *optimal* with respect to the preferences specified by the agents.

▶ **Definition 1** (Feasible matching)**.** *A* feasible matching *M in $G = (\mathcal{R} \cup \mathcal{H}, E)$ is a subset of E such that $q^-(v) \leq |M(v)| \leq q^+(v)$ for each $v \in \mathcal{R} \cup \mathcal{H}$, where $M(v)$ is the set of neighbours of v assigned to v in M.*

Before we discuss the notion of optimality, we outline the challenges that lower quotas pose in the HRLQ problem, a special case of the HR2LQ problem. In the presence of two-sided preferences but no lower quotas, *stability* is a well-accepted notion of optimality. Stable matchings are characterized by the absence of a blocking pair.

▶ **Definition 2** (Blocking pair)**.** *Given a matching M, a pair $(r, h) \in E \setminus M$ is called a* blocking pair *with respect to M if either r is unmatched or r prefers h over its matched partner $M(r)$ and either h is under-subscribed ($|M(h)| < q^+(h)$) or h prefers r over at least one of the residents matched to it, that is, some resident in $M(h)$. A matching M is* stable *if there is no blocking pair with respect to M.*

A stable matching always exists; however, a stable *and* feasible matching need not exist in the presence of lower quotas for even the hospitals alone (HRLQ problem). Nasre and Nimbhorkar [20] used an alternate notion of optimality, namely *popularity* to circumvent the problem. Informally, a matching is *popular* if no *majority* of agents wish to deviate from the matching. In [20], the authors show that for every instance of the HRLQ problem, there exists a feasible matching that is *popular* amongst the set of feasible matchings, and such matching can be computed efficiently. In our work, we show that in the presence of lower quotas on both sides of the bipartition, that is, in the HR2LQ setting, a matching that is popular amongst the set of feasible matchings exists, and it can be efficiently computed.

The setting consisting of two-sided preferences and lower quotas has received a lot of attention, e.g. Huang [9] investigate it for stable matchings where there are classifications along with lower quotas. Fleiner and Kamiyama [6] consider stable matchings with matroid constraints and lower quotas. Mnich and Schlotter [19] investigate lower quotas on both sides in the restricted stable marriage setting (one-to-one). Popularity in the HR2LQ setting has not been investigated so far. The HR2LQ problem is well motivated by several practical applications. Hospital lower quotas are important for the smooth functioning of hospitals. Similarly, some residents may have to be matched because of their economic backgrounds or because they are unallocated from the previous year. Another setting where HR2LQ arise is the allocation of mentors to students. The mentors need a group of students to carry out discussion sessions, whereas it may be required that the students whose CGPA falls below a certain threshold must get a mentor. Another application of HR2LQ is in elective allocation. While allotting electives to students, it is natural to have a lower quota on electives denoting the minimum number of students required for the elective to be offered, and the students who are in their final semester must be assigned an elective.

Notion of popularity. The notion of popularity used here is the same as in [20]. It is based on votes cast by each vertex to compare two given matchings M and N. A resident r votes for M if r prefers $M(r)$ over $N(r)$, and vice versa. If $M(r) = N(r)$ then r is indifferent between the two matchings, and hence does not cast any vote. If r is unmatched in M, we define $M(r) = \perp$, and r prefers any hospital in its preference list over $\perp$. We denote $vote_r(M, N)$ to denote the vote of r between M and N. It takes values 1, -1 or 0 depending on whether r prefers $M(r)$ over $N(r)$ or vice versa, or is indifferent between them.

For a hospital h, there can be up to $q^+(h)$ residents in $M(h)$ and $N(h)$. If h is undersubscribed in M or N then we assume that the remaining positions of h are matched to $\perp$. In this way, we can always assume that $|M(h)| = |N(h)| = q^+(h)$. So the hospital can cast up to $q^+(h)$ votes. The hospital is indifferent between M and N as far as the residents in $M(h) \cap N(h)$ are concerned. For the remaining residents, h needs to decide a correspondence $\mathbf{corr}_h$ for comparing $M(h) \setminus N(h)$ to $N(h) \setminus M(h)$. Then

$$vote_h(M, N, \mathbf{corr}_h) = \sum_{r \in M(h) \setminus N(h)} vote_h(r, \mathbf{corr}_h(r, M, N))$$

Here $\mathbf{corr}_h(r, M, N)$ denotes the resident $r' \in N(h) \setminus M(h)$ corresponding to r, and $vote_h(r, \mathbf{corr}_h(r, M, N))$ is 1 if h prefers r over r', is -1 if h prefers r' over r, and 0 if $r = r'$.

Finally the number of votes that the matching M gets over N is given by

$$\Delta(M, N, \mathbf{corr}) = \sum_{r \in \mathcal{R}} vote_r(M, N) + \sum_{h \in \mathcal{H}} vote_h(M, N, \mathbf{corr}_h)$$

▶ **Definition 3** (Popular Matching [20]). *A matching M is more popular than N (denoted as $M \succ_{\mathbf{corr}} N$) under **corr** if $\Delta(M, N, \mathbf{corr}) > 0$. A matching M is popular if there is no matching N such that $N \succ_{\mathbf{corr}} M$ for any choice of **corr** from N to M.*

Related work and techniques. The notion of popularity was introduced by Gärdenfors [7] in 1975 as a majority assignment in the context of a full stable marriage problem. In 2005, Abraham et al. [1] discussed an efficient algorithm for computing popular matching in a bipartite graph where only one set of the partition has preferences. Since then popular matchings have been well-studied and a vast literature [2, 10, 12, 8, 4, 18, 13] on popular matchings is available.

Huang and Kavitha [10] and subsequently Kavitha [12] studied the popular matchings as an alternative to stability in the stable marriage setting (where there are no lower quotas). Their motivation was to obtain matchings larger in size than the stable matching, which are optimal with respect to the preferences. Subsequently, for matchings in bipartite graphs with two-sided preferences, popularity has been investigated in the many-to-one setting (HR problem) [21], many-to-many setting [3] and the many-to-one setting with hospital lower quotas (HRLQ problem) [20, 18].

Independent of our work, very recently, Kavitha [15] investigates popularity of a matching in a stable marriage instance when both sides have lower quotas – denoted as critical nodes in her work. Her approach is similar to ours and finds a maximum size popular matching amongst the set of critical matchings. A critical matching is one which matches as many critical nodes as possible. Thus, in her work, it is not required to have the guarantee that the input instance admits a feasible matching, which is required in our work. We remark that our algorithm is for the HR2LQ setting which allows lower-quotas (or equivalently critical nodes) on both sides and non-unit upper-quotas on one side of the bipartition. Our algorithm can be easily extended to compute a maximum size feasible popular matching.

We comment on the two related but seemingly different ways of computing a popular matching used in the literature. The first approach used in [12, 18, 3] is what we term as *the modified Gale and Shapley (GS)* approach. This involves the first round of proposals using the standard GS algorithm, followed by the second round of proposals where unmatched vertices are allowed to propose with *increased priority.* This simple and elegant idea has its origins in Kirlay's work [17] on computing constant factor approximation to maximum sized stable matching in the presence of ties in preferences.

The second approach used in [21, 4] is to *simulate* the modified GS algorithm via reducing the original instance to a stable matching instance G'. A stable matching in G' is mapped to a popular matching in G. We call this the *reduction approach.* For example, in [20], the authors use the reduction approach and convert the input HRLQ instance into an HR instance where the standard GS algorithm is used to compute a matching. The reduced instance consists of multiple copies of all the hospitals with a positive lower quota. A modified GS approach on an HRLQ instance would involve giving multiple higher priorities to *deficient* hospitals, i.e. the hospitals whose lower quotas are not met. The two different approaches are indeed equivalent in the HRLQ setting since the reduction essentially simulates the modified GS algorithm.

Extension to HR2LQ. A natural extension of the modified GS approach for HR2LQ would be to execute one round of GS algorithm with say hospitals proposing, then letting deficient hospitals (if any) propose with multiple higher priorities, and then letting deficient residents (if any) propose with multiple higher priorities. In other words, one side, let's say $\mathcal{H}$, start proposing using the standard GS algorithm to compute round 1 matching M_1, which is stable. If a hospital h remains deficient in M_1 ($|M_1(h)| < q^-(h)$), then h is allowed to propose with an increased priority. A resident will always accept the proposal from a higher priority hospital by rejecting a lower priority one. A hospital keeps proposing with increased priority if it exhausts its preference list and is left deficient at the current priority. The priority of a hospital is increased (by one at a time) until it becomes non-deficient. It can be shown that no hospital is deficient in the matching M_2 obtained at the end of round 2. But a resident may still be deficient and hence, a deficient resident r is allowed to propose (possibly with increased priority) until it becomes non-deficient. In this process, another resident r' may get *rejected* for the *first time* by a hospital h because h has got a proposal from a higher priority resident and $|M(h)| = q^+(h)$. Now at this point, r' has two choices either (a) start proposing from the beginning of its list or (b) propose the hospital just after h in its preference list and continue. If r' is such that $q^-(r') = 0$, then it continues proposing hospitals until either it gets matched to some hospital or has exhausted its preference list. But if $q^-(r') = 1$ and r' has exhausted its preference list without getting any match, then it starts proposing from the beginning of the list with increased priority. In the end, we get the round 3 matching M_3.

Note that in the HRLQ setting, the modified GS approach comprises only round 1 and round 2 and gives a popular matching. But the example in Figure 1 shows that this approach in the case of HR2LQ does not yield a popular matching using any of the two choices mentioned above in round 3.

The round 1 matching computed by $\mathcal{H}$-proposing GS algorithm is $M_1 = \{(h_1, \perp), (h_2, \perp), (h_3, r_3), (h_4, r_1), (h_5, r_2), (h_6, \perp), (h_7, r_5), (h_8, \perp)\}$. Note that the matching M_1 is not feasible as the hospitals h_1, h_2 and h_6 are deficient. In round 2, these hospitals increase their priority to 1 (from 0) and start proposing from the start of the list. In order to remove the deficiency, h_1 increases its priority to 2 whereas h_2 and h_6 got matched while being at priority level 1. The matching M_2 at the end of round 2 is $M_2 = \{(h_1^2, r_1), (h_2^1, r_2), (h_3, r_3), (h_4, \perp), (h_5, \perp), (h_6^1, r_5), (h_7, \perp), (h_8, \perp)\}$. In M_2, the resident r_2 is deficient and hence round 3 starts.

$[0,1]$	r_1 :	h_4	h_2	h_1	
$[0,1]$	r_2 :	h_5	h_2	h_6	h_1
$[1,1]$	r_3 :	h_3	h_2		
$[1,1]$	r_4 :	h_3			
$[0,1]$	r_5 :	h_7	h_6	h_8	

(a) Preference List of Residents with Quotas $[q^-(r), 1]$.

$[1,1]$	h_1 :	r_1	r_2	
$[1,1]$	h_2 :	r_1	r_2	r_3
$[0,1]$	h_3 :	r_3	r_4	
$[0,1]$	h_4 :	r_1		
$[0,1]$	h_5 :	r_2		
$[1,1]$	h_6 :	r_2	r_5	
$[0,1]$	h_7 :	r_5		
$[0,1]$	h_8 :	r_5		

(b) Preference List of Hospitals with Quotas $[q^-(h), q^+(h)]$.

Figure 1 Counter-example for the modified GS approach.

In round 3, r_3 and r_4 try to snatch h_3 from each other and they raise their priority level to 1. The hospital h_2 rejects r_2 when it receives a proposal from r_3^1 and this was the first rejection of r_2 in this round. So, following the first choice of round 3, r_2 proposes h_5 and got matched to it. This results in the matching $M_3' = \{(h_1^2, r_1), (h_2^1, r_3^1), (h_3, r_4^2), (h_4, \perp), (h_5, r_2), (h_6^1, r_5), (h_7, \perp), (h_8, \perp)\}$. When we remove the priority levels we get $M' = \{(h_1, r_1), (h_2, r_3), (h_3, r_4), (h_5, r_2), (h_6, r_5)\}$. If the procedure follows the second choice of round 3, then r_2 proposes to h_6 which in turn reject r_5 and then r_5 proposes to h_8 and get matched to it. Round 3 ends here and it results in the matching $M_3'' = \{(h_1^2, r_1), (h_2^1, r_3^1), (h_3, r_4^2), (h_4, \perp), (h_5, \perp), (h_6^1, r_2), (h_7, \perp), (h_8, r_5)\}$ which maps to $M'' = \{(h_1, r_1), (h_2, r_3), (h_3, r_4), (h_6, r_2), (h_8, r_5)\}$.

We remark that neither M' nor M'' is popular as there exists another matching $M = \{(h_1, r_1), (h_2, r_6), (h_3, r_4), (h_6, r_2), (h_7, r_5)\}$ which is more popular than both.

Although the modified GS approach does not work, a natural extension of the reduction approach works. Thus we present a reduction from the HR2LQ problem to the HR problem and show that a stable matching in the reduced HR instance can be translated to a popular matching in the original instance. Thus the reduction approach works but seems to have no straightforward analogous modified GS approach. Our correctness proof is inspired by the one in [3], which uses LP duality to prove the popularity of their matching. They exhibit a dual assignment as a certificate for the popularity of the matching. Dual certificate for proving the popularity has been used earlier in the literature [13, 11, 5, 14, 16].

However, the algorithm in [3] uses a modified GS approach and has no lower quotas. Also, in [3], only one side of the bipartition gets *one* higher priority. So the dual certificate consists of a $\{\pm 1, 0\}$ assignment to all the dual variables. In contrast, our reduction makes multiple copies of residents and hospitals. As a consequence, exhibiting the dual assignment is more involved and needs weights linear in the size of the input instance.

Formally, our result is stated below:

▶ **Theorem 4.** *In an HR2LQ instance that admits a feasible matching, there always exists a matching that is popular amongst all the feasible matchings. Moreover, such a popular matching can be computed in time polynomial in the size of the input instance.*

Organization of the paper. We describe our reduction in Section 2. The feasibility and popularity proofs appear in Section 3 and Section 4 respectively. Section 5 concludes the paper.

2 Reduction

Given an HR2LQ instance $G = (\mathcal{H} \cup \mathcal{R}, E)$ we construct an HR instance $G' = (\mathcal{H}' \cup \mathcal{R}', E')$ as follows. Let μ_R and μ_H denote respectively the sum of lower quotas of residents and hospitals in the instance G. That is, $\mu_R = \sum_{r \in R} q^-(r)$ and $\mu_H = \sum_{h \in H} q^-(h)$. Let $\mathcal{R}_{lq}$ and $\mathcal{H}_{lq}$ denote the set of residents and set of hospitals with positive lower quota. That is, $\mathcal{R}_{lq} = \{\ r \mid r \in \mathcal{R}\ and\ q^-(r) = 1\}$ and $\mathcal{H}_{lq} = \{\ h \mid h \in \mathcal{H}\ and\ q^-(h) > 0\}$. A resident in $\mathcal{R}_{lq}$ is called a *lower-quota* resident and a hospital in $\mathcal{H}_{lq}$ is called *lower-quota* hospital. Our instance G' has the following residents and hospitals:

1. **Resident Copies:** For every resident $r \in \mathcal{R}_{lq}$, we have $\mu_R + 1$ copies of r in G'. These copies are $r^0, r^1, \ldots, r^{\mu_R}$ where r^x is called the level-x copy of r. Note that all $r \in \mathcal{R}_{lq}$ have both upper and lower quota equal to one in the HR2LQ instance G. The capacities in G' for these copies are: level-0 copy has capacity equal to the upper quota of r and all other copies have capacity equal to the lower quota of r. That is,
$$q(r^x) \quad = \quad 1 \qquad \text{for } 0 \le x \le \mu_R$$
A resident $r \notin \mathcal{R}_{lq}$ in G has exactly one copy r^0 in G' and we call it the level-0 copy of the resident r. The capacity of the level-0 copy in G' is $q(r^0) = 1$. We denote this set of copies of the residents as $\mathcal{R}'_c$ and call them *true* residents.
2. **Hospital Copies:** For every hospital $h \in \mathcal{H}_{lq}$, we have $\mu_H + 1$ copies of h in G'. These copies are $h^0, h^1, \ldots, h^{\mu_H}$ where h^y is called the level-y copy of h. The capacities in G' for these copies are: level-0 copy has capacity equal to the upper quota of h and all other copies have capacity equal to the lower quota of h. That is,
$$\begin{aligned} q(h^y) &= q^+(h) \quad \text{if } y = 0 \\ &= q^-(h) \quad \text{if } 1 \le y \le \mu_H \end{aligned}$$
A hospital $h \notin \mathcal{H}_{lq}$ in G has exactly one copy h^0 in G' and we call it the level-0 copy of the hospital. The capacity of the level-0 copy in G' is $q(h^0) = q^+(h)$. We denote this set of copies of the hospitals as $\mathcal{H}'_c$ and call them *true* hospitals.
3. **Dummy Hospitals:** For every $r \in \mathcal{R}_{lq}$ we have μ_R many dummy hospitals. The role of dummy hospitals is to ensure that in a stable matching in G' at most, one true hospital is matched across several copies of a lower-quota resident. We denote the set of dummy hospitals corresponding to $r \in \mathcal{R}_{lq}$ as $\mathcal{D}_r$ which is defined as:
$$\mathcal{D}_r = \{\ d_r^y \quad | \quad 0 \le y < \mu_R\ \}$$
4. **Dummy Residents:** For every $h \in \mathcal{H}_{lq}$ we have μ_H sets of dummy residents. As with dummy hospitals, the role of the dummy residents is to ensure that in any stable matching in G', at most $q^+(h)$ many true residents are matched across several copies of the lower-quota hospital. We denote the level-x set of dummy residents corresponding to a lower-quota hospital $h \in \mathcal{H}_{lq}$ as $\mathcal{D}_h^x$. The set is defined as follows:
$$\mathcal{D}_h^x = \begin{cases} \{d_{h,1}^x, d_{h,2}^x, \ldots, d_{h,q^+(h)}^x\} & \text{for } x = 0 \\ \{d_{h,1}^x, d_{h,2}^x, \ldots, d_{h,q^-(h)}^x\} & \text{for } 1 \le x < \mu_H \end{cases}$$

We are now ready to define our resident set $\mathcal{R}'$ and hospital set $\mathcal{H}'$ in G'.

$$\mathcal{R}' = \mathcal{R}'_c \cup \bigcup_{\substack{h \in \mathcal{H}_{lq} \\ 0 \le x \le \mu_H - 1}} \mathcal{D}_h^x \qquad\qquad \mathcal{H}' = \mathcal{H}'_c \cup \bigcup_{r \in \mathcal{R}_{lq}} \mathcal{D}_r$$

We now define our preference lists for the residents and the hospitals in G'. For any vertex $v \in \mathcal{R} \cup \mathcal{H}$, let $\langle list_v \rangle$ denote the preference list of v in G. Let $\langle lqlist_v \rangle$ denote the preference list of v restricted to the lower-quota vertices in its preference list, where the relative ordering of the lower-quota vertices is preserved. Finally, for a particular level t, we denote by $\langle lqlist_v\rangle^t$ as the list of level-t copies of the lower-quota vertices in the preference list of v. For example in G if a resident r has its preference list as h_1, h_2, h_3, h_4 where h_2 and h_4 belong to $\mathcal{H}_{lq}$, then $\langle list_r\rangle = h_1, h_2, h_3, h_4$ and $\langle lqlist_r\rangle = h_2, h_4$. Furthermore say $t = 3$, then $\langle lqlist_r\rangle^3 = h_2^3, h_4^3$. We let the symbol $\circ$ denotes the concatenation of two preference lists.

1. **Preferences of true residents:** For a resident $r \notin \mathcal{R}_{lq}$, we have exactly one copy r^0 in G' whose preference list is obtained by concatenating the r's highest level lower-quota hospitals, followed by the next highest level lower-quota hospitals and so on finally followed by *all* level-0 hospitals in its preference list. Formally, the list for r^0 is defined below.

   $$r^0 \ : \ \langle lqlist_r\rangle^{\mu_H} \circ \langle lqlist_r\rangle^{\mu_H - 1} \circ \ldots \circ \langle lqlist_r\rangle^{1} \circ \langle list_r\rangle^{0}$$

   Recall that, for a resident $r \in \mathcal{R}_{lq}$ we have $\mu_R + 1$ many copies of r in G'. Broadly, the preference list of these copies are obtained by prefixing and suffixing dummies to the preference list of r^0 shown above. Hence we find it convenient to use the notation $\langle clonedlist_r\rangle$ to denote the following:

   $$\langle clonedlist_r\rangle \ = \ \langle lqlist_r\rangle^{\mu_H} \circ \langle lqlist_r\rangle^{\mu_H - 1} \circ \ldots \circ \langle lqlist_r\rangle^{1} \circ \langle list_r\rangle^{0}$$

   The preference lists of the $\mu_R + 1$ copies of r can be defined using the $\langle clonedlist_r\rangle$ as given below.

   $$\begin{array}{lcr}
   r^0 & : & \langle clonedlist_r\rangle \circ d_r^0 \\
   r^1 & : & d_r^0 \circ \langle clonedlist_r\rangle \circ d_r^1 \\
   & & \vdots \\
   r^{\mu_R - 1} & : & d_r^{\mu_R - 2} \circ \langle clonedlist_r\rangle \circ d_r^{\mu_R - 1} \\
   r^{\mu_R} & : & d_r^{\mu_R - 1} \circ \langle clonedlist_r\rangle
   \end{array}$$

2. **Preferences of true hospitals:** For a hospital $h \notin \mathcal{H}_{lq}$, we have exactly one copy h^0 in G' whose preference list is obtained by concatenating the hospital's highest level lower-quota residents, followed by the next highest level lower-quota residents and so on finally followed by *all* level-0 residents in its preference list. Formally, the list for h^0 is defined below.

   $$h^0 \ : \ \langle lqlist_h\rangle^{\mu_R} \circ \langle lqlist_h\rangle^{\mu_R - 1} \circ \ldots \circ \langle lqlist_h\rangle^{1} \circ \langle list_h\rangle^{0}$$

   As with lower-quota residents, we have $\mu_H + 1$ many copies of a lower-quota hospitals in G'. We will use the notation $\langle clonedlist_h\rangle$ to denote the following:

   $$\langle clonedlist_h\rangle \ = \ \langle lqlist_h\rangle^{\mu_R} \circ \langle lqlist_h\rangle^{\mu_R - 1} \circ \ldots \circ \langle lqlist_h\rangle^{1} \circ \langle list_h\rangle^{0}$$

   The preference lists of the $\mu_H + 1$ copies of h can be defined using the $\langle clonedlist_h\rangle$ as given below. Note that for a level-y copy h^y of h we have $q(h^y)$ many leading dummies, followed by the *cloned list* of h followed by $q(h^y)$ many trailing dummies. The leading dummies for h^y are the trailing dummies for h^{y-1} and the the trailing dummies for h^y are leading dummies for h^{y+1}. Recall that $q(h^0) = q^+(h)$ and $q(h^1) = q^-(h)$ and hence $q^-(h)$ trailing dummies of h^0 are the leading dummies of h^1. Thus, k in h^1's preference list denote the value $k = q^+(h) - q^-(h) + 1$.

   $$\begin{array}{lcr}
   h^0 & : & \langle clonedlist_h\rangle \circ d^0_{h,1}, \ldots, d^0_{h,q^+(h)} \\
   h^1 & : & d^0_{h,k}, \ldots, d^0_{h,q^+(h)} \circ \langle clonedlist_h\rangle \circ d^1_{h,1}, \ldots, d^1_{h,q^-(h)} \\
   h^2 & : & d^1_{h,1}, \ldots, d^1_{h,q^-(h)} \circ \langle clonedlist_h\rangle \circ d^2_{h,1}, \ldots, d^2_{h,q^-(h)} \\
   & & \vdots \\
   h^{\mu_H - 1} & : & d^{\mu_H - 2}_{h,1}, \ldots, d^{\mu_H - 2}_{h,q^-(h)} \circ \langle clonedlist_h\rangle \circ d^{\mu_H - 1}_{h,1}, \ldots, d^{\mu_H - 1}_{h,q^-(h)} \\
   h^{\mu_H} & : & d^{\mu_H - 1}_{h,1}, \ldots, d^{\mu_H - 1}_{h,q^-(h)} \circ \langle clonedlist_h\rangle
   \end{array}$$

3. **Preferences of dummy hospitals:** All dummy hospitals have a preference list of length two. The preference list of a dummy hospital d_r^y for a lower quota resident r is:
   $d_r^y : r^y, r^{y+1}$
4. **Preferences of dummy residents:** We have several sets of dummy residents corresponding to a lower-quota hospital $h \in \mathcal{H}_{lq}$. The dummy residents, except for the first $k-1$ many level-0 dummy residents ($k = q^+(h) - q^-(h) + 1$) have a preference list of length two. The preference lists of the dummy residents is given below.
   $$\begin{array}{llll} d_{h,i}^x & : & h^0 & x = 0, \quad i \in \{1, 2, \ldots, k-1\} \\ & : & h^0, h^1 & x = 0, \quad i \in \{k, \ldots, q^+(h)\} \\ & : & h^x, h^{x+1} & x \in \{1, 2, \ldots, \mu_H - 1\} \end{array}$$
   The preference lists of dummy residents and dummy hospitals ensures that a stable matching in G' naturally maps to a popular matching in the original instance G.

This completes the description of our reduced instance G'. We illustrate this reduction in Appendix A.1 using an example, where we convert the HR2LQ instance of Figure 1 to an HR instance. In the next section, we describe the outline of our algorithm and show that if the given HR2LQ instance admits a feasible matching, then the output of our algorithm is a feasible matching.

3 Our algorithm and its feasibility

Given our reduction from an HR2LQ instance G to an HR instance G', our algorithm to compute a popular matching M is straightforward. We simply run the standard Gale-Shapley algorithm on G' and compute a stable matching M_s. We will first prove some useful properties of the stable matching M_s which allows a natural way to obtain a matching M in G.

We call a vertex *under-subscribed* in a matching M if $|M(v)| < q^+(v)$. Note that for residents, under-subscribed is the same as unmatched.

▶ **Definition 5** (Active vertex). *A hospital h^y is* active *in M_s if h^y is matched to at least one true resident in M_s. Otherwise, we call h^y inactive in which case it is matched to all dummy residents. A resident r^x is* active *in M_s if r^x is matched to a true hospital in M_s, else r^x is inactive.*

▶ **Definition 6** (True edges). *An edge $e \in E'$ is called a* true *edge if both the end points are true vertices that is $e = (r^x, h^y)$ where $r^x \in \mathcal{R}'_c$ and $h^y \in \mathcal{H}'_c$.*

Next, we describe some crucial properties of a stable matching M_s in the HR instance G'. Our reduction, more specifically the placement of the set of dummy residents in preference lists, ensures that a hospital h is not matched to more than $q^+(h)$ many true residents across all the level copies $h^0, \ldots, h^{\mu_H}$ in M_s. As M_s is a stable matching, at most two *consecutive* level copies h^y and h^{y+1} of h can be matched to true residents in M_s, otherwise there exists a blocking pair w.r.t. M_s. All lower-level copies (less than y) are completely matched to the respective trailing dummy residents, and all higher-level copies (greater than $y+1$) are completely matched to the respective leading dummy residents. Moreover, only the highest level copy of a hospital can remain under-subscribed, and if it happens then, none of its lower-level copies is matched to a true resident. Similar properties hold for resident copies as well. That is, at most, one *level copy* of a resident is matched to a true hospital, and all other *level copies* are matched to dummy hospitals in any stable matching. Moreover, if a level-x copy of a resident is matched to some true hospital, then all its lower-level copies are matched to the respective trailing dummy hospital, and the higher level copies are matched to the respective leading dummy hospital. We formally list these properties of a stable matching of G' in Lemma 7. The proof appears in Appendix A.2.

▶ **Lemma 7.** *The stable matching M_s in G' satisfies the following properties:*

1. *For any resident $r \in \mathcal{R}$, M_s matches at most one true hospital across all the level copies of r in G'. For any $h \in \mathcal{H}$, M_s matches at most $q^+(h)$ true residents across all the level copies of h in G'.*
2. *The matching M_s leaves only the highest level copy of the vertex (resident or hospital) under-subscribed. In case of hospitals, for $h \in \mathcal{H}_{lq}$, this implies that only h^{μ_H} is possibly under-subscribed in M_s. For a non lower-quota hospital h, the level-0 copy which is the highest level copy may be under-subscribed in M_s. Similar claims hold true for residents.*
3. *If r^x is active in M_s then,*
   a. *Every resident r^i where $0 \leq i \leq x-1$ is inactive in M_s and matched to its trailing dummy hospital d_r^i.*
   b. *Every resident r^i where $x+1 \leq i \leq \mu_R$ is inactive in M_s and matched to its leading dummy hospital d_r^{i-1}.*
4. *If h^y is active in M_s then*
   a. *The hospital h^{y-1} must be matched to at least one dummy resident among its trailing dummies, that is, in the set $\mathcal{D}_h^{y-1}$.*
   b. *Every hospital h^j where $0 \leq j \leq y-2$ is inactive in M_s and fully-subscribed with its trailing dummies, that is residents in $\mathcal{D}_h^j$*
   c. *Every hospital h^j where $y+2 \leq j \leq \mu_H$ is inactive in M_s and fully-subscribed with its leading dummies, that is, residents in $\mathcal{D}_h^{j-1}$.*
5. *For any resident at most one of its level copy is active in M_s. For any hospital h at most two consecutive level copies are active in M_s.*
6. *If a level y, $y > 0$ copy of a hospital h is active in M_s, then h is matched to at most $q^-(h)$ true residents in M_s. If the highest level copy h^{μ_H} of a hospital h is under-subscribed in M_s then none of its level-j copies for $j < \mu_H$ are active in M_s.*

Lemma 8 below states that a stable matching in G' cannot contain an edge whose both the endpoints are *active* at *higher* levels. This allows us to define a simple *map function* that maps the stable matching M_s in G' to a feasible popular matching M in G.

▶ **Lemma 8.** *For every true edge (r^x, h^y) in G', if r^x and h^y are active in M_s, then at least one of x and y must be 0.*

Proof. For the sake of contradiction, let us assume that there is an edge (r^x, h^y) such that $x, y > 0$ and both are active in M_s. Since, r^x is matched to a true hospital, r^{x-1} must get matched to its last dummy hospital d_r^{x-1} by Part 3a in Lemma 7. So, r^{x-1} prefers h^{y-1} over $M_s(r^{x-1}) = d_r^{x-1}$. Similarly, since h^y is active in M_s, h^{y-1} must be matched to at least one of the trailing dummies in M_s say, $d \in \mathcal{D}_h^{y-1}$. Thus, h^{y-1} prefers r^{x-1} over one of its matched partner d. Hence, (r^{x-1}, h^{y-1}) is a blocking pair w.r.t. a stable matching M_s. ◀

From the above discussion, a *mapping function* that maps M_s of G' to M in G is straight forward. For each edge (r^x, h^0) or (r^0, h^y) in M_s, we include the edge (r, h) in M. We prove that the mapping M is feasible for the original HR2LQ instance G by combining the two claims given in Lemma 9. We give the proofs of these two feasibility claims in Appendix A.2.

▶ **Lemma 9.** *Let M be the map of the stable matching M_s in G', then following holds true:*

1. *If G admits a resident-feasible matching, then M is resident-feasible in G.*
2. *If G admits a hospital-feasible matching, then M is hospital-feasible for G.*

4 Popularity of our matching

Given a feasible matching N in the HR2LQ instance $G = (\mathcal{R} \cup \mathcal{H}, E)$ we construct a weighted bipartite graph $\tilde{G}_N$ along with a matching N^*. The weight of an edge in $\tilde{G}_N$ is the sum of the votes by the end vertices of that edge when compared to the matching N, and the matching N^* is a one-to-one matching corresponding to N. The construction of the matching N^* is in such a way that it matches all the *clones* of residents and hospitals in $\mathcal{R} \cup \mathcal{H}$. We call N^* an $(\mathcal{R} \cup \tilde{\mathcal{H}})$- *perfect matching* (where $\tilde{\mathcal{H}}$ is set of all the clones of hospitals). In other words, the weight of an edge represents the sum of the votes by end vertices and the $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching N^* has weight 0. Hence, to prove that N is popular, it suffices to show that the maximum weight $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching in $\tilde{G}_N$ has weight at most zero. For this, we write a maximum weight matching LP for $\tilde{G}_N$ and exhibit a feasible dual assignment with value zero. This is inspired from [3]; however, as mentioned earlier, our dual assignment is considerably involved given that we have multiple levels for both residents and hospitals.

4.1 The graph $\tilde{G}_N$ corresponding to a feasible matching N

Now we describe the construction of the weighted graph $\tilde{G}_N$ corresponding to any feasible matching N in G, and the one-to-one matching N^* using the matching N.

1. **Vertex set of $\tilde{G}_N$:** The graph $\tilde{G}_N$ has the vertex set as $\mathcal{R} \cup \tilde{\mathcal{H}} \cup \tilde{\mathcal{L}}$. The set $\mathcal{R}$ denotes the same set of residents as in G. The set $\tilde{\mathcal{H}}$ denotes *clones* of the original hospitals where every hospital in $\mathcal{H}$ has upper quota many clones. That is,
$$\tilde{\mathcal{H}} = \{\ h_j \ \mid \ h \in \mathcal{H} \ \text{ and } 1 \le j \le q^+(h)\}$$
every $h \in \mathcal{H}$ has $q^+(h)$ many clones in $\tilde{\mathcal{H}}$. Having these upper quota many clones allows us to convert the many-to-one matching N to a one-to-one matching N^*. Finally, the set $\tilde{\mathcal{L}} = \tilde{\mathcal{L}_r} \cup \tilde{\mathcal{L}_h}$ denotes the set of (dummy) last-resort vertices. For every non lower-quota resident r we have a last resort hospital $\ell_r \in \tilde{\mathcal{L}_r}$. For each vertex $h \in \mathcal{H}$ let $d(h) = q^+(h) - q^-(h)$ denote the difference between the upper quota and lower quota of h. Corresponding to h we have $d(h)$ many last-resort residents in $\tilde{\mathcal{L}_h}$. That is,
$$\begin{aligned} \tilde{\mathcal{L}_r} &= \{\ \ell_r \ \mid \ r \in \mathcal{R} \text{ and } q^-(r) = 0\} \\ \tilde{\mathcal{L}_h} &= \{\ \ell_{h_k} \ \mid \ h \in \mathcal{H} \ \text{ and } 1 \le k \le d(h)\} \end{aligned}$$
Thus a lower-quota resident and a hospital with a lower quota equal to its upper quota do not have any last-resort vertices corresponding to it. These last-resort vertices are used to convert N to an $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching in $\tilde{G}_N$. We call these vertices last-resorts to avoid confusion with the dummies used in the reduction in Section 2.
2. **The matching N^*:** Given the feasible (many-to-one) matching N, we construct an $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect one-to-one matching N^*. For every edge $(r, h) \in N$ we select an unselected clone of h say h_j and add the edge (r, h_j) to N^*. For a resident $r \in \mathcal{R}$ which is unmatched in N, we add the edge (r, ℓ_r) to N^*. For any $h \in \mathcal{H}$ which is under-subscribed in N, that is $|N(h)| < q^+(h)$, for every unmatched clone of h, say h_j we match it to a unique last-resort say ℓ_{h_j} and hence add the edge (h_j, ℓ_{h_j}) to N^*. Thus our matching N^* is a one-to-one and $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching.
3. **The unmatched edges E_U in $\tilde{G}_N$:** For every edge $(r, h) \in E \setminus N$, we add $q^+(h)$ many edges to E_U. That is, we add to the edge set the edges (r, h_j) for every clone h_j of h. We also have unmatched edges from clones of hospitals to the last-resorts corresponding to the hospitals. We have two cases depending on whether $|N(h)| > q^-(h)$ or $|N(h)| = q^-(h)$. This construction is important for our dual feasible setting in the next section.

- For a hospital h where $|N(h)| > q^-(h)$ we have a complete bipartite graph between the $q^+(h)$ many clones of h and all the last-resort vertices corresponding to h. Recall that we have $d(h) = q^+(h) - q^-(h)$ many last resorts corresponding to h. Thus we add to E_U edges of the form (ℓ_{h_k}, h_j) where $1 \leq k \leq d(h)$ and $1 \leq j \leq q^+(h)$.
- For a hospital h where $|N(h)| = q^-(h)$, we have a complete bipartite graph between the set of clones of h matched to last-resort vertices and all the last resort vertices corresponding to h. Thus we add to E_U edges of the form (ℓ_{h_k}, h_j) where $1 \leq k \leq d(h)$ and h_j is matched to a last-resort in the above construction.

4. **The edge set $\tilde{E}$ and their weights:** The edge set $\tilde{E} = N^* \cup E_U$. Every edge of N^* is assigned a weight 0 and every edge (r, h_j) of E_U is assigned a weight $=$ $vote_r(h, N^*(r)) + vote_h(r, N^*(h_j))$ where $r \in \mathcal{R}$ and $h_j \in \tilde{\mathcal{H}}$. Every edge of the form (r, ℓ_r) of E_U is assigned a weight $= vote_r(\ell_r, N^*(r))$ where $r \in \mathcal{R}$ and $\ell_r \in \tilde{\mathcal{L}}_r$. Similarly, every edge of the form (h_j, ℓ_{h_k}) of E_U is assigned a weight $= vote_h(\ell_{h_k}, N^*(h_j))$ where $h \in \tilde{\mathcal{H}}$ and $\ell_{h_k} \in \tilde{\mathcal{L}}_h$.

This completes the description of the weighted bipartite graph $\tilde{G}_N$. Now we use Theorem 10, which gives the sufficient condition for the matching N to be popular, to prove the popularity of the matching M computed by our algorithm. We will construct the matching M^* and the graph $\tilde{G}_M$ corresponding to the matching M. Our goal is to show that *every* $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching in $\tilde{G}_M$ has weight at most 0.

▶ **Theorem 10.** *Let N be a feasible matching in G such that every $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching in $\tilde{G}_N$ has weight at most 0 then N is popular.*

Proof. For any feasible matching T in G, we show a corresponding matching T^* in $\tilde{G}_N$ such that T^* is an $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching and $wt(T^*) = \Delta(T, N, \mathbf{corr})$, where $wt(T^*)$ denotes the sum of the weights of the edges in T^*. We construct T^* as described next. We find appropriate index $j \in \{1, ..., q^+(h)\}$ corresponding to each edge $(r, h) \in T$, where $(r, h_j) \in \tilde{E} \cap T^*$. For an unmatched resident r and an under-subscribed hospital h in T, we add (r, ℓ_r) and (h_k, ℓ_{h_j}) edges in T^* to make T^* an $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching.

(i) For each edge $e = (r, h) \in N \cap T$: if $(r, h_j) \in N^*$ then we add the edge (r, h_j) to T^*.

(ii) For every edge $(r, h) \in T \setminus N$, we need to decide the index j such that $(r, h_j) \in T^*$. While evaluating the votes, h uses the correspondence function $\mathbf{corr}_h$. (a) If $\mathbf{corr}_h(r, T, N) = r'$ then the matching N^* must contain an edge (r', h_j) for some j. We include the edge (r, h_j) in T^*. (b) If $\mathbf{corr}_h(r, T, N) = \perp$ then we include (r, h_j) in T^* for some j such that h_j is unmatched so far in T^* and is not adjacent to any of the corresponding last resorts. If there is no such h_j then we arbitrarily choose a clone h_i such that $(h_i, \ell_{h_i}) \in N^*$ and include (h_i, ℓ_{h_i}) in T^*.

(iii) For any vertex $v_k \in \mathcal{R} \cup \tilde{\mathcal{H}}$ that is left unmatched in the above step, we select an arbitrary but distinct j for $1 \leq j \leq q^+(v) - q^-(v)$ and include the edge (v_k, ℓ_{v_k}) in T^*.

Since T is a feasible matching in G, all the clones of every hospital $h \in \mathcal{H}$ which are not adjacent to last resort vertices must get matched to a resident $r \in \mathcal{R}$ in T^*. Other clones of h get matched to either a resident or to the last resort. The graph $\tilde{G}_N$ contains exactly $q^+(v) - q^-(v)$ many last resorts for each vertex v and hence all the copies of v which are not matched to a true vertex in step (i) or (ii) above must get matched to one of these last resorts in step (iii). So, all the vertices in $\mathcal{R} \cup \tilde{\mathcal{H}}$ are matched in T^*. It is also easy to see that T^* is a one-to-one matching in $\tilde{G}_N$.

Next, we compute the weight of T^* and show that it is $\Delta(T, N, \mathbf{corr})$. Since every $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching in $\tilde{G}_N$ has weight at most 0, $\Delta(T, N, \mathbf{corr}) = wt(T^*) \leq 0$. This says that there is no feasible matching T which is more popular than N. Hence N is popular matching among all the feasible matchings.

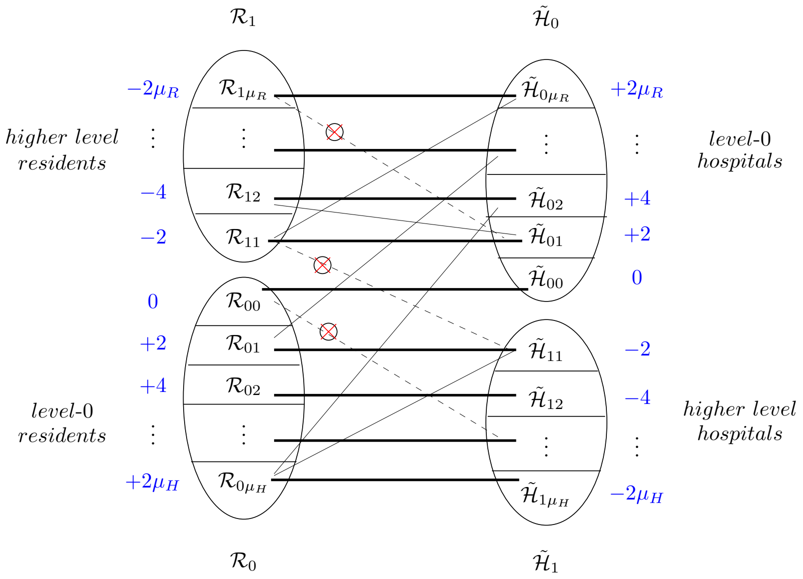


Figure 2 The graph $\tilde{G}_M$ corresponding to our feasible matching M. The bold edges represent the edges in M^*. The values outside the ellipse denote the dual setting and are useful in Section 4.4.

$$\begin{aligned}
wt(T^*) = \sum_{e \in T^*} wt(e) &= \sum_{(r,h_j) \in T^*} wt(r, h_j) + \sum_{(v_k, \ell_{v_k}) \in T^*} wt(v_k, \ell_{v_k}) \\
&= \sum_{(r,h_j) \in T^*} (vote_r(T^*(r), N^*(r)) + vote_{h_j}(T^*(h_j), N^*(h_j))) \\
&\quad + \sum_{(v_k, \ell_{v_k}) \in T^*} vote_{v_k}(\ell_{v_k}, N^*(v_k)) \\
&= \sum_{r \in \mathcal{R}} vote_r(T^*(r), N^*(r)) + \sum_{h \in \mathcal{H}} \sum_{i=1}^{q^+(h)} vote_h(T^*(h_i), N^*(h_i)) \\
&= \sum_{r \in \mathcal{R}} vote_r(T, N) + \sum_{h \in \mathcal{H}} vote_h(T, N, \mathbf{corr}_h) \\
&= \Delta(T, N, \mathbf{corr})
\end{aligned}$$

Thus it follows that $\Delta(T, N, \mathbf{corr}) = wt(T^*)$ which is at most 0 and hence N is popular. ◀

4.2 The graph $\tilde{G}_M$ corresponding to M obtained from M_s

For the HR2LQ instance G, consider the feasible matching M obtained from the stable matching M_s in the reduced HR instance G'. We now use the construction described in Section 4.1 to obtain the graph $\tilde{G}_M$ and the one-to-one $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching M^*. Since M was obtained from the stable matching M_s in G', an edge $(r, h) \in M$ corresponds to an edge $(r^x, h^y) \in M_s$ where r^x and h^y are level-x and level-y copies of the respective vertices. Further, by Lemma 8 we know that at least one of x or y is zero. We use this property crucially to partition the vertex set of $\tilde{G}_M$. We partition the vertices as described below. Figure 2 shows the high level partition of the vertex set as $\mathcal{R}_0 \cup \mathcal{R}_1 \cup \tilde{\mathcal{H}}_0 \cup \tilde{\mathcal{H}}_1$. Each partition is further refined, for instance $\mathcal{R}_0$ is partitioned as $\mathcal{R}_{00} \cup \mathcal{R}_{01} \cup \ldots \cup \mathcal{R}_{0\mu_H}$. Recall the vertex set of $\tilde{G}_M$ which is given by $\mathcal{R} \cup \tilde{\mathcal{H}} \cup \tilde{\mathcal{L}_h} \cup \tilde{\mathcal{L}_r}$. We partition the residents (including last-resort residents) $\mathcal{R} \cup \tilde{\mathcal{L}_h}$ as $\mathcal{R}_0 \cup \mathcal{R}_1$ and the hospitals (including last-resort hospitals) $\tilde{\mathcal{H}} \cup \tilde{\mathcal{L}_r}$ as $\tilde{\mathcal{H}}_0 \cup \tilde{\mathcal{H}}_1$. Note that the edges of M^* are obtained from the edges of M and the matching M is obtained from the stable matching M_s in G'. We use the edges of M_s, in particular, the levels of the end points of the matched edges, to partition the vertices of $\tilde{G}_M$. The vertices of $\tilde{G}_M$ includes the upper quota many clones for every hospital and the last-resort vertices.

Partition of vertices of $\tilde{G}_M$. Here, we define the sets $\mathcal{R}_{0x}, \mathcal{R}_{1x}, \mathcal{H}_{0y}, \mathcal{H}_{1y}$ based on the edges of M_s.

- Let (r^x, h^y) be an edge in M_s. We consider three cases based on the values of x and y. Note that the case $x > 0, y > 0$ does not arise due to Lemma 8.
  1. If $x = 0$ and $y > 0$, then add r to $\mathcal{R}_{0y}$ and add $M^*(r)$ to $\tilde{\mathcal{H}}_{1y}$. We would like to emphasize that we are using $M^*(r)$ and *not* $M_s(r)$. Since M^* is a one-to-one $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching, $M^*(v)$ is a uniquely defined vertex of $\tilde{G}_M$ for any vertex v of $\tilde{G}_M$.
  2. If $x > 0$ and $y = 0$ then add r to $\mathcal{R}_{1x}$ and add $M^*(r)$ to $\tilde{\mathcal{H}}_{0x}$.
  3. If $x = 0$ and $y = 0$ then add r to $\mathcal{R}_{00}$ and add $M^*(r)$ to $\tilde{\mathcal{H}}_{00}$.
- For any resident $r \in \mathcal{R}$ that is unmatched in M_s, we add r to $\mathcal{R}_{00}$ and $M^*(r)$ to $\mathcal{H}_{00}$. Note that $M^*(r)$ is a last-resort hospital.
- For any hospital $h \in \mathcal{H}$ that is under-subscribed in M_s, let h_j be a clone of h which is matched to a last-resort in M^*. We add h_j to $\mathcal{H}_{00}$ and $M^*(h_j)$ to $\mathcal{R}_{00}$.
- Finally, any last-resort resident not yet added to any partition is added to $\mathcal{R}_{00}$. Similarly any last-resort hospital not yet added to any partition is added to $\mathcal{H}_{00}$.

We note that the set $\mathcal{R}_0 = \bigcup_{x=1}^{\mu_H} \mathcal{R}_{0x}$. The sets $\mathcal{R}_1, \mathcal{H}_0, \mathcal{H}_1$ are defined similarly. Figure 2 shows the graph $\tilde{G}_M$. It is convenient to have the sets $\mathcal{R}_0$ and $\mathcal{H}_1$ drawn on the lower part and the sets $\mathcal{R}_1$ and $\mathcal{H}_0$ drawn in the upper part. Furthermore, inside $\mathcal{R}_0$ we have the sets $\mathcal{R}_{01}, \mathcal{R}_{02}, \ldots, \mathcal{R}_{0\mu_H}$ arranged from top to bottom. Similarly, we arrange the sets inside $\mathcal{R}_1, \mathcal{H}_0, \mathcal{H}_1$ as shown in Figure 2. We now state the properties of the edges of the graph viewed via the lens of the partition of the vertices in Lemma 11 and Lemma 12. For an edge (u, v) where $u \in \mathcal{R}_{ax}$ and $v \in \tilde{\mathcal{H}}_{by}$ we say the edge is of the form $\mathcal{R}_{ax} \times \tilde{\mathcal{H}}_{by}$.

▶ **Lemma 11.** *The graph $\tilde{G}_M$ does not contain an edge of the form:*

1. $\mathcal{R}_1 \times \tilde{\mathcal{H}}_1$. *That is, there is no edge in $\tilde{G}_M$ from the top right set of residents to the lower left set of hospitals.*
2. $\mathcal{R}_{1x} \times \tilde{\mathcal{H}}_{0y}$ *where $y < x - 1$. That is, in the top set of residents and hospitals, there is no steep downward edge in $\tilde{G}_M$.*
3. $\mathcal{R}_{0x} \times \tilde{\mathcal{H}}_{1y}$ *for $y > x + 1$. That is, in the bottom set of residents and hospitals there is no steep downward edge in $\tilde{G}_M$.*

Proof.

- *Proof of 1*: Proof is immediate from Lemma 8. Because, if there exists an edge (r, h) in G such that $r \in \mathcal{R}_1$ and $h_j \in \tilde{\mathcal{H}}_1$ then in G', there must exist an edge (r^x, h^y) with $x, y > 0$. But then the edge (r^x, h^y) blocks M_s as they both prefer each other over some of their matched partners.
- *Proof of 2*: We prove this by contradiction. Suppose there exist an edge $e = (r, h_j)$ such that $r \in \mathcal{R}_{1x}$ and $h_j \in \tilde{\mathcal{H}}_{0y}$ for $y \leq x - 2$. This means r^x is matched to some h'^0 and one of the matched partner of h^0 is some resident r'^y. Now, consider the resident r^{x-1}. This resident must be matched to its trailing dummy hospital and hence prefers h^0 over its

matched partner. Any hospital prefers higher level resident over any lower level resident and hence h^0 prefers r^{x-1} over r'^y. Thus, (r^{x-1}, h^0) blocks the stable matching M_s. This is a contradiction.

- *Proof of 3*: Proof is the same as the proof of 2 above. ◀

In Figure 2, the edges not present in $\tilde{G}_M$ are the dashed edges marked with a red cross inside a circle. We now state the properties of the weights on the edges of $\tilde{G}_M$. Note that the weight of an edge of $\tilde{G}_M$ denotes the sum of the votes of the end-points when compared to the matching M. Thus for $e \in \tilde{E}$, we have $-2 \leq wt(e) \leq 2$.

▶ **Lemma 12.** *Let $e = (r, h_j)$ be any edge in $\tilde{G}_M$ such that $r \in \mathcal{R}$ and $h_j \in \tilde{\mathcal{H}}$. Then,*

1. *If $e \in \mathcal{R}_{1x} \times \tilde{\mathcal{H}}_{0(x-1)}$ then $wt(e) = -2$.*
2. *If $e \in \mathcal{R}_{1x} \times \tilde{\mathcal{H}}_{0x}$ then $wt(e) \in \{-2, 0\}$*
3. *If $e \in \mathcal{R}_{1x} \times \tilde{\mathcal{H}}_{0y}$ for $y > x$ then $wt(e) \leq 2$.*
4. *If $e \in \mathcal{R}_0 \times \tilde{\mathcal{H}}_0$ then $wt(e) \leq 2$. Moreover, if $e \in \mathcal{R}_{00} \times \tilde{\mathcal{H}}_{00}$ then $wt(e) \leq 0$.*
5. *If $e \in \mathcal{R}_{0x} \times \tilde{\mathcal{H}}_{1y}$ for $y < x$ then $wt(e) \leq 2$.*
6. *If $e \in \mathcal{R}_{0x} \times \tilde{\mathcal{H}}_{1x}$ then $wt(e) = 0$ or -2.*
7. *If $e \in \mathcal{R}_{0x} \times \tilde{\mathcal{H}}_{1(x+1)}$ then $wt(e) = -2$.*

Proof.

- *Proof of 1*: The weight $wt(e)$ for an edge $e = (r, h_j)$ is defined as $wt(r, h_j) = vote_r(h_j, M^*(r)) + vote_{h_j}(r, M^*(h_j))$. So, our goal here is to show that $vote_r(h_j, M^*(r)) = vote_{h_j}(r, M^*(h_j)) = -1$. Assume for the sake of contradiction that $vote_r(h_j, M^*(r)) \neq -1$ and $vote_{h_j}(r, M^*(h_j)) \neq -1$. Hence, we have three other possibilities [(+1,+1), (+1,-1) and (-1,+1)] and all these three are covered in below two cases: (a) $vote_r(h_j, M^*(r)) = 1$ and (b) $vote_r(h_j, M^*(r)) = -1$. Suppose $r \in \mathcal{R}_{1x}$ and $h_j \in \tilde{\mathcal{H}}_{0(x-1)}$. Consider the same edge in the reduced graph G'. This is an edge between r^x and h^0, and h^0 is matched with some resident $r'^{(x-1)}$. First, we show that $vote_r(h_j, M^*(r)) \neq 1$ which implies $vote_r(h_j, M^*(r)) = -1$. If $vote_r(h_j, M^*(r)) = 1$ then (r^x, h^0) is a blocking edge w.r.t. the stable matching M_s in G' because h^0 always prefers higher level resident r^x over the lower level resident $r'^{(x-1)}$, one of its matched partner. Now, as $vote_r(h_j, M^*(r)) = -1$, the only other possibility left is $vote_r(h_j, M^*(r)) = -1$ and $vote_{h_j}(r, M^*(h_j)) = +1$ but then consider the vertex r^{x-1} in G', this vertex must be matched to its trailing dummy and hence prefers h^0 more. As h prefers r over its matched partner r', h^0 must prefer r^{x-1} over $r'^{(x-1)}$. Again, we have a blocking edge (r^{x-1}, h^0) w.r.t. the stable matching M_s in G'. Hence, $wt(e) = -2$.
- *Proof of 2*: If $e \in \mathcal{R}_{1x} \times \tilde{\mathcal{H}}_{0x}$ then $wt(e)$ cannot be $+2$, otherwise the same edge in G' will be a blocking edge w.r.t. M_s. Hence, $wt(e)$ can only be -2 or 0.
- *Proof of 3*: The maximum possible weight for any e is 2.
- *Proof of 4*: The largest possible weight $wt(e)$ is $+2$. If $e \in \mathcal{R}_{00} \times \tilde{\mathcal{H}}_{00}$ and $wt(e) = 2$ then the same edge in G' blocks M_s.

Proofs of 5, 6 and 7 follow from the earlier claims. ◀

4.3 Linear Program and its Dual

Given the weighted graph $\tilde{G}_M$ we use the standard linear program (LP) to compute a maximum weight $(\mathcal{R} \cup \tilde{\mathcal{H}})$-perfect matching in $\tilde{G}_M$. Recall that every edge e has a weight associated with it which denotes the sum of the votes of the endpoints of the edge with respect to the matching M. The LP and its dual (dual-LP) are given below. For the (primal) LP we have a variable x_e for every edge in $\tilde{E}$. We let $\delta(v)$ denote the set of edges incident on the vertex v in the graph $\tilde{G}_M$.

$$\textbf{LP:} \qquad \max \sum_{e \in \tilde{E}} wt(e) \cdot x_e$$

subject to:

$$\begin{aligned} \sum_{e \in \delta(v)} x_e &= 1 && \forall v \in \mathcal{R} \cup \tilde{\mathcal{H}} \\ \sum_{e \in \delta(\ell_{h_k})} x_e &\leq 1 && \forall \ell_{h_k} \in \tilde{\mathcal{L}}_h \\ x_e &\geq 0 && \forall e \in \tilde{E} \end{aligned}$$

We obtain the dual of the above LP by associating a variable α_v for every $v \in \mathcal{R} \cup \tilde{\mathcal{H}} \cup \tilde{\mathcal{L}}$.

$$\textbf{dual-LP:} \qquad \min \sum_{r \in \mathcal{R}} \alpha_r + \sum_{h_j \in \tilde{\mathcal{H}}} \alpha_{h_j} + \sum_{\ell_{h_k} \in \tilde{\mathcal{L}}_h} \alpha_{\ell_{h_k}}$$

subject to:

$$\begin{aligned} \alpha_r + \alpha_{h_j} &\geq wt(r, h_j) && \forall (r, h_j) \in \tilde{E} \text{ where } r \in \mathcal{R},\ h_j \in \tilde{\mathcal{H}} && (1) \\ \alpha_{\ell_{h_k}} + \alpha_{h_j} &\geq wt(\ell_{h_k}, h_j) && \forall (\ell_{h_k}, h_j) \in \tilde{E} \text{ where } \ell_{h_k} \in \tilde{\mathcal{L}}_h, h_j \in \tilde{\mathcal{H}} && (2) \\ \alpha_r &\geq wt(r, \ell_r) && \forall r \in \mathcal{R} \text{ and } q^-(r) = 0 && (3) \\ \alpha_{\ell_{h_k}} &\geq 0 && \forall \ell_{h_k} \in \tilde{\mathcal{L}}_h && (4) \end{aligned}$$

4.4 Dual Assignment and its correctness

In this section we present an assignment of values to the dual variables of the **dual-LP** and prove that it is feasible as well as the sum of the dual values is zero. The dual assignment is shown in Figure 2 in blue.

- Set $\alpha_r = +2x$ for all $r \in \mathcal{R}_{0x}$ where $0 \leq x \leq \mu_H$.
- Set $\alpha_{h_j} = -2y$ for all $h_j \in \tilde{\mathcal{H}}_{1y}$ where $1 \leq y \leq \mu_H$.
- Set $\alpha_r = -2x$ for all $r \in \mathcal{R}_{1x}$ where $1 \leq x \leq \mu_R$.
- Set $\alpha_{h_j} = +2y$ for all $h_j \in \tilde{\mathcal{H}}_{0q}$ where $0 \leq y \leq \mu_R$.
- Set $\alpha_{\ell_{h_k}} = 0$ for the last resorts corresponding to a hospital $h \in \mathcal{H}$ are 0.

▶ **Lemma 13.** *The above dual assignment is feasible, and the sum of the dual values is zero.*

Proof. We prove that the dual assignment satisfies all (1), (2), (3), (4) of the dual LP. Eq (4) holds because the last resorts corresponding to a hospital are assigned α-values 0. It is clear from the partition of the vertices of $\mathcal{R} \cup \tilde{\mathcal{H}}$ that all the non-lower quota residents are only in $\mathcal{R}_0$. The α-values for all such residents are non-negative. Moreover, the $wt(r, \ell_r)$ for a non lower-quota resident is at most 0. This implies that all the non-lower-quota residents satisfy the Eq (3). Recall that there is no last resort corresponding to a lower-quota resident.

Next, we show that our dual assignment satisfies Eq (2). Since, $wt(\ell_{h_k}, h_j)$ is at most 0, it is sufficient to show that the LHS of the second inequality $\alpha_{\ell_{h_k}} + \alpha_{h_j}$ is at least 0. From the partition of the vertices into subsets, it is easy to see that no *non lower-quota copy* of a hospital is in $\tilde{\mathcal{H}}_1$ and, hence no vertex in $\tilde{\mathcal{H}}_1$ is connected to the corresponding last resorts. This implies that all the copies (of hospitals) which are connected to corresponding last resorts are in $\tilde{\mathcal{H}}_0$ and they are assigned α-value at least 0. Since $\alpha_{\ell_{h_k}} = 0$, the LHS of the second inequality is at least 0.

Now, to show that the first inequality of the dual LP holds true for the above assignments, we use the Lemma 11 and Lemma 12.

- Lemma 11 excludes all the edges which can not be there in $\tilde{G}_M$.
- Lemma 12(1) states that for every edge $e \in \mathcal{R}_{1x} \times \tilde{\mathcal{H}}_{0(x-1)}$, we have $wt(e) = -2$. As per our dual assignment $\alpha_r + \alpha_{h_j} = -2x + 2(x-1) = -2 \geq wt(e)$.
- Lemma 12(2) states that for every edge $e \in \mathcal{R}_{1x} \times \tilde{\mathcal{H}}_{0x}$, we have $wt(e) = -2$ or 0. As per our dual assignment $\alpha_r + \alpha_{h_j} = -2x + 2x = 0 \geq wt(e)$.
- Lemma 12(3) states that for every edge $e \in \mathcal{R}_{1x} \times \tilde{\mathcal{H}}_{0y}$ such that $y > x$, we have $wt(e)$ is at most 2. As per our dual assignment $\alpha_r + \alpha_{h_j} = -2x + 2y \geq 2 \geq wt(e)$.
- Lemma 12(4) states that (a) for every edge $e \in \mathcal{R}_{00} \times \tilde{\mathcal{H}}_{00}$, we have $wt(e) \leq 0$. As per our dual assignment $\alpha_r + \alpha_{h_j} = 0 \geq wt(e)$. (b) for all other edges $e \in \mathcal{R}_0 \times \tilde{\mathcal{H}}_0$, we have $wt(e) \leq 2$. Our dual assignment ensures that $\alpha_r + \alpha_{h_j} \geq 2 \geq wt(e)$.
- Lemma 12(5) states that for every edge $e \in \mathcal{R}_{0x} \times \tilde{\mathcal{H}}_{1y}$ for $y < x$, we have $wt(e) \leq 2$. Our dual assignment ensures that $\alpha_r + \alpha_{h_j} = 2x - 2y \geq 2 \geq wt(e)$.
- Lemma 12(6) states that for every edge $e \in \mathcal{R}_{0x} \times \tilde{\mathcal{H}}_{1x}$, we have $wt(e) \leq 0$. As per our dual assignment $\alpha_r + \alpha_{h_j} = 2x - 2x = 0 \geq wt(e)$.
- Lemma 12(7) states that for every edge $e \in \mathcal{R}_{0x} \times \tilde{\mathcal{H}}_{1(x+1)}$, we have $wt(e) = -2$. In this case also, $\alpha_r + \alpha_{h_j} = 2x - 2(x+1) = -2 \geq wt(e)$.

Hence, all the edges of $\tilde{E}$ satisfy inequality (1). As per our assignment, all the matched edges (r, h_j) has $\alpha_r + \alpha_{h_j} = 0$, and $\alpha_r = 0$ for all the residents r matched to last resorts and $\alpha_{h_j} = 0$ for all the clones h_j of hospital h such that h_j are matched to last resorts. Hence it follows that $\sum_{v \in \mathcal{R} \cup \tilde{\mathcal{H}}} \alpha_v = 0$. ◀

Lemma 13 and the weak duality theorem together implies that the optimal value of the primal LP is at most 0. That is, every matching in $\tilde{G}_M$ that matches all vertices in $\mathcal{R} \cup \tilde{\mathcal{H}}$ has weight at most 0. Thus, by using Theorem 10, we establish the main result of this paper stated in Theorem 4.

5 Discussion

In this paper, we addressed the problem of computing a popular, feasible matching in the many-to-one setting when both sides are having lower quotas. This approach can suitably be modified to compute a maximum size popular feasible matching. We comment on the natural generalizations of our problem as follows:

- **Many-to-many setting with one sided lower-quotas:** We call this setting as the student course allocation problem with lower quotas for courses (SCLQ problem). The simple modification of our reduction presented in the paper works for the SCLQ setting.
- **Many-to-many setting with two-sided lower-quotas:** We denote this as the SC2LQ problem. For the SC2LQ setting, there are non-trivial challenges in extending our reduction. These are posed by the presence of capacities as well as lower quotas on both the sides of the bipartition. We remark that these difficulties do not arise in the SCLQ setting where only one side has lower quotas as well as in [3] there are no lower quotas on either side of the bipartition.

References

1 David J Abraham, Robert W Irving, Telikepalli Kavitha, and Kurt Mehlhorn. Popular matchings. *SIAM Journal on Computing*, 37(4):1030–1045, 2007.

2 Péter Biró, Robert W Irving, and David F Manlove. Popular matchings in the marriage and roommates problems. In *International Conference on Algorithms and Complexity*, pages 97–108. Springer, 2010.

3 Florian Brandl and Telikepalli Kavitha. Two problems in max-size popular matchings. *Algorithmica*, 81(7):2738–2764, 2019.

4 Ágnes Cseh and Telikepalli Kavitha. Popular edges and dominant matchings. *Mathematical Programming*, 172(1):209–229, 2018.

5 Yuri Faenza and Telikepalli Kavitha. Quasi-popular matchings, optimality, and extended formulations. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 325–344. SIAM, 2020.

6 Tamás Fleiner and Naoyuki Kamiyama. A matroid approach to stable matchings with lower quotas. *Math. Oper. Res.*, 41(2):734–744, 2016. doi:10.1287/moor.2015.0751.

7 Peter Gärdenfors. Match making: assignments based on bilateral preferences. *Behavioral Science*, 20(3):166–173, 1975.

8 Mizuki Hirakawa, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita. On the structure of popular matchings in the stable marriage problem-who can join a popular matching. In *the 3rd International Workshop on Matching under Preferences (MATCH-UP)*, 2015.

9 Chien-Chung Huang. Classified stable matching. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, pages 1235–1253, 2010. doi:10.1137/1.9781611973075.99.

10 Chien-Chung Huang and Telikepalli Kavitha. Popular matchings in the stable marriage problem. *Information and Computation*, 222:180–194, 2013.

11 Chien-Chung Huang and Telikepalli Kavitha. Popularity, mixed matchings, and self-duality. *Mathematics of Operations Research*, 2021.

12 Telikepalli Kavitha. A size-popularity tradeoff in the stable marriage problem. *SIAM Journal on Computing*, 43(1):52–71, 2014.

13 Telikepalli Kavitha. Popular half-integral matchings. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

14 Telikepalli Kavitha. Popular matchings with one-sided bias. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

15 Telikepalli Kavitha. Matchings, critical nodes, and popular solutions. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021 (To Appear)*, 2021.

16 Telikepalli Kavitha. Maximum Matchings and Popularity. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198, pages 85:1–85:21, 2021. doi:10.4230/LIPIcs.ICALP.2021.85.

17 Zoltán Király. Better and simpler approximation algorithms for the stable marriage problem. *Algorithmica*, 60(1):3–20, 2011.

18 A. M. Krishnapriya, Meghana Nasre, Prajakta Nimbhorkar, and Amit Rawat. How good are popular matchings? In *17th International Symposium on Experimental Algorithms, SEA 2018*, pages 9:1–9:14, 2018. doi:10.4230/LIPIcs.SEA.2018.9.

19 Matthias Mnich and Ildikó Schlotter. Stable matchings with covering constraints: A complete computational trichotomy. *Algorithmica*, 82(5):1136–1188, 2020.

20 Meghana Nasre and Prajakta Nimbhorkar. Popular matchings with lower quotas. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017*, pages 44:1–44:15, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.44.

21 Meghana Nasre and Amit Rawat. Popularity in the generalized hospital residents setting. In *International Computer Science Symposium in Russia*, pages 245–259. Springer, 2017.

A Appendix

A.1 Illustration of Reduction Method

Following the reduction given in Section 2 we convert the HR2LQ instance of Figure 1 to an HR instance. The resulting reduced instance is shown in Figure 3. The map of the hospital proposing stable matching in the reduced instance is $M = \{(h_1, r_1), (h_2, r_3), (h_3, r_4), (h_6, r_2), (h_7, r_5)\}$ which is popular. The level structure and dual assignment is shown in Figure 4.

$$\begin{array}{lllllllllllllll}
r_1^0 : & h_2^3 & h_1^3 & h_2^2 & h_1^2 & h_2^1 & h_1^1 & h_4^0 & h_2^0 & h_1^0 \\
r_2^0 : & h_2^3 & h_6^3 & h_1^3 & h_2^2 & h_6^2 & h_1^2 & h_2^1 & h_6^1 & h_1^1 & h_5^0 & h_2^0 & h_6^0 & h_1^0
\end{array}$$

$$\begin{array}{llllllll}
r_3^0 : & h_2^3 & h_2^2 & h_2^1 & h_3^0 & h_2^1 & d_{r_3}^0 \\
r_3^1 : & d_{r_3}^0 & h_2^3 & h_2^2 & h_2^1 & h_3^0 & h_2^1 & d_{r_3}^1 \\
r_3^2 : & d_{r_3}^1 & h_2^3 & h_2^2 & h_2^1 & h_3^0 & h_2^1
\end{array}$$

$$\begin{array}{lllllll}
r_4^0 : & h_3^3 & h_3^2 & h_3^1 & h_3^0 & d_{r_4}^0 \\
r_4^1 : & d_{r_4}^0 & h_3^3 & h_3^2 & h_3^1 & h_3^0 & d_{r_4}^1 \\
r_4^2 : & d_{r_4}^1 & h_3^3 & h_3^2 & h_3^1 & h_3^0
\end{array}$$

$$r_5^0 : \; h_6^3 \;\; h_6^2 \;\; h_6^1 \;\; h_7^0 \;\; h_6^0 \;\; h_8^0$$

$$\begin{array}{lll}
d_{h_{1,1}}^0 : & h_1^0 & h_1^1 \\
d_{h_{1,1}}^1 : & h_1^1 & h_1^2 \\
d_{h_{1,1}}^2 : & h_1^2 & h_1^3 \\
d_{h_{2,1}}^0 : & h_2^0 & h_2^1 \\
d_{h_{2,1}}^1 : & h_2^1 & h_2^2 \\
d_{h_{2,1}}^2 : & h_2^2 & h_2^3 \\
d_{h_{6,1}}^0 : & h_6^0 & h_6^1 \\
d_{h_{6,1}}^1 : & h_6^1 & h_6^2 \\
d_{h_{6,1}}^2 : & h_6^2 & h_6^3
\end{array}$$

(a) Preference List of Residents with Quota=1.

$$\begin{array}{lllll}
h_1^0 : & r_1^0 & r_2^0 & d_{h_{1,1}}^0 \\
h_1^1 : & d_{h_{1,1}}^0 & r_1^0 & r_2^0 & d_{h_{1,1}}^1 \\
h_1^2 : & d_{h_{1,1}}^1 & r_1^0 & r_2^0 & d_{h_{1,1}}^2 \\
h_1^3 : & d_{h_{1,1}}^2 & r_1^0 & r_2^0
\end{array}$$

$$\begin{array}{llllllll}
h_2^0 : & r_3^2 & r_3^1 & r_1^0 & r_2^0 & r_3^0 & d_{h_{2,1}}^0 \\
h_2^1 : & d_{h_{2,1}}^0 & r_3^2 & r_3^1 & r_1^0 & r_2^0 & r_3^0 & d_{h_{2,1}}^1 \\
h_2^2 : & d_{h_{2,1}}^1 & r_3^2 & r_3^1 & r_1^0 & r_2^0 & r_3^0 & d_{h_{2,1}}^2 \\
h_2^3 : & d_{h_{2,1}}^2 & r_3^2 & r_3^1 & r_1^0 & r_2^0 & r_3^0
\end{array}$$

$$\begin{array}{lllllll}
h_3^0 : & r_3^2 & r_4^2 & r_3^1 & r_4^1 & r_3^0 & r_4^0 \\
h_4^0 : & r_1^0 \\
h_5^0 : & r_2^0
\end{array}$$

$$\begin{array}{lllll}
h_6^0 : & r_2^0 & r_5^0 & d_{h_{6,1}}^0 \\
h_6^1 : & d_{h_{6,1}}^0 & r_2^0 & r_5^0 & d_{h_{6,1}}^1 \\
h_6^2 : & d_{h_{6,1}}^1 & r_2^0 & r_5^0 & d_{h_{6,1}}^2 \\
h_6^3 : & d_{h_{6,1}}^2 & r_2^0 & r_5^0
\end{array}$$

$$\begin{array}{ll}
h_7^0 : & r_5^0 \\
h_8^0 : & r_5^0
\end{array}$$

$$\begin{array}{lll}
d_{r_3}^0 : & r_3^0 & r_3^1 \\
d_{r_3}^1 : & r_3^1 & r_3^2 \\
d_{r_4}^0 : & r_4^0 & r_4^1 \\
d_{r_4}^1 : & r_4^1 & r_4^2
\end{array}$$

(b) Preference List of Hospitals with Quota=1.

Figure 3 Reduced HR instance corresponding to the counter example given in Figure 1.

A.2 Proofs from Section 3

Proof of Lemma 7.

- *Proof of 1:* If $r \notin \mathcal{R}_{lq}$ then r has only one copy in G' with quota 1 and the result holds trivially. So without loss of generality let us assume that $r \in \mathcal{R}_{lq}$ and hence G' has $\mu_R + 1$ copies of r. The total number of dummy hospitals corresponding to r is μ_R. Each dummy hospital d_r^i contains only two true residents r^i and r^{i+1}. Also, each d_r^i is the top choice

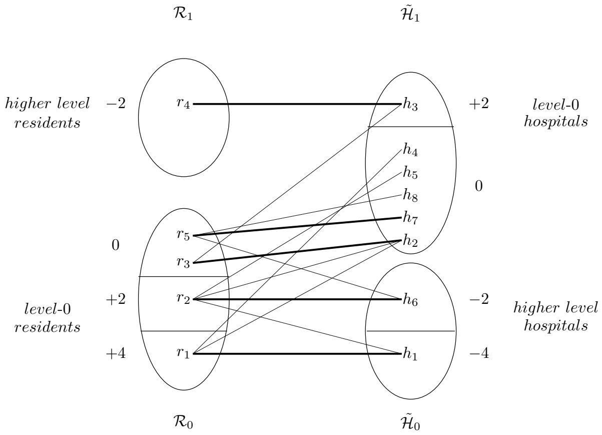


Figure 4 The graph $\tilde{G}_M$ corresponding to the example.

of r^{i+1} for $0 \leq i \leq \mu_R$. So, none of the dummy hospital can remain unmatched. This implies that at most 1 copy of r can get matched to a true hospital.

If $h \notin \mathcal{H}_{lq}$ then h has only one copy h^0 in G' with quota $q^+(h)$ and the result holds trivially. So without loss of generality, let us assume that $h \in \mathcal{H}_{lq}$ and hence G' has $\mu_H + 1$ copies of h. The total number of dummy residents for h in G' is $\alpha = q^+(h) + q^-(h) \cdot (\mu_H - 1)$, and the total capacity of all the copies of h in G' is $\beta = q^+(h) + q^-(h) \cdot \mu_H$. Consider the set of dummy residents $\mathcal{D}_h^0 \cup \ldots \cup \mathcal{D}_h^{\mu_H - 1}$ corresponding to a lower quota hospital h. For any $y < \mu_H$, except $y = 1$, $\mathcal{D}_h^{y-1}$ are the most preferred $q(h^y)$ dummy residents of h^y. Thus, these dummy residents can never remain unmatched in a stable matching. The dummy residents that can possibly remain unmatched are the subset of $\{d^0_{h,1}, \ldots, d^0_{h,q^+(h)-q^-(h)}\}$ as these are the only dummy residents that are not the top choice of any copy of the hospital h. Hence the number of dummy residents that can remain unmatched in any stable matching of G' is at most $\gamma = q^+(h) - q^-(h)$. This implies that the total number of true residents matched to h in M_s is at most $\beta - \alpha + \gamma = q^+(h)$.

- *Proof of 2:* If $h \notin \mathcal{H}_{lq}$ then it has only one copy h^0 with quota $q^+(h)$. So let us assume that $h \in \mathcal{H}_{lq}$. For each copy h^y, where $y < \mu_H$, there are exactly $q(h^y)$ dummy residents of level-y as their top choice. Thus, h^y cannot remain under-subscribed in any stable matching M_s of G', otherwise these dummy resident(s) form blocking pair(s) with h^y. This implies that only h^{μ_H} can possibly be left under-subscribed in M_s.

  If $r \notin \mathcal{R}_{lq}$, then its highest level copy r^0 remains unmatched. So let us assume that $r \in \mathcal{R}_{lq}$. We note that none of the μ_R many dummy hospitals $d^0_r, d^1_r, \ldots, d^{\mu_R - 1}_r$ corresponding to r can be left unmatched in any stable matching. Otherwise , the unmatched dummy hospital d^i_r forms a blocking pair with r^{i+1}. So, at most, one copy of r can potentially be left unmatched. Now, if a copy r^i for $0 \leq i \leq \mu_R - 1$ is left unmatched then r^i with its last dummy d^i_r forms a blocking pair w.r.t. M_s.
- *Proof of 3:* There are $\mu_R + 1$ copies of a resident $r \in \mathcal{R}_{lq}$ and μ_R dummy hospitals corresponding to it. From the proof of 1 above, we know that none of the dummy hospitals corresponding to r can remain unmatched. The preference list of a dummy hospital d^i_r contains r^i and r^{i+1}, and r^x is active in M_s. This implies that the only possible way to match r^i in a stable matching is to match it with d^i_r, the corresponding trailing dummy, where $0 \leq i \leq x - 1$. Similarly, the only possible way to match r^i in a stable matching is to match it with d^{i+1}_r, the corresponding leading dummy, where $x + 1 \leq i \leq \mu_R$.

- *Proof of 4a:* For the sake of contradiction, assume that h^{y-1} is not matched to any resident $d \in D_h^{y-1}$ and still h^y is active. Note that there are exactly $q(h^y)$ many dummy residents in the preference list of h^y from the set $\mathcal{D}_h^{y-1}$. Also, h^y prefers all such dummy residents over any true resident. Each dummy resident from the $(y-1)$-th set, $\mathcal{D}_h^{y-1}$ has only h^{y-1} and h^y in its preference list. It means there is a dummy resident $d_h^j \in D_h^{y-1}$ which is unmatched in M_s. But then (h^y, d_h^j) forms a blocking pair w.r.t. M_s.

  Proof of 4b: If h^y is active and h^j is matched to a true resident r for some $0 \leq j \leq y-2$, then (r, h^{y-1}) is a blocking pair w.r.t. M_s. This is because, as proved above, h^{y-1} must be matched to at least one resident in $\mathcal{D}_h^{y-1}$ and h^{y-1} prefers any true resident over any dummy resident in $\mathcal{D}_h^{y-1}$.

  Proof of 4c: If h^y is active then h^j cannot be active for $y+2 \leq j \leq \mu_H$, otherwise, h^{j-1} must be matched to a resident from $\mathcal{D}_h^{j-1}$. In this case, each true resident which is matched to h^y in M_s forms a blocking pair with h^{j-1}, contradicting the stability of M_s. Now we claim that each such h^j is fully-subscribed with its leading dummies. This is because if h^y is active and h^j is matched to any trailing dummy $d \in \mathcal{D}_h^j$ then a resident $r' \in M_s(h^y)$ forms a blocking pair with h^j.
- *Proof of 5:* The claim for a resident immediately follows from Part 1 above. So, let us prove it for a hospital.

  For the sake of contradiction, let us assume that $h \in \mathcal{R}_{lq}$ is a lower quota hospital such that h^{x_1} and h^{x_2} are active where $x_2 < x_1 - 1$. Also, assume that h^{x_1} and h^{x_2} are matched to r_1 and r_2 respectively. Then, h^{x_1-1} must be matched to at least one dummy residents from $\mathcal{D}_h^{x_1-1}$. But, Then, (r_2, h^{x_1-1}) forms a blocking pair w.r.t. M_s.
- *Proof of 6:* This follows from the fact that for any h, all the dummy residents of all the copies get matched in a stable matching in G', except possibly the $q^+(h) - q^-(h)$ trailing dummies of h^0. This is because all of them are the top choice of some h^i. The other part is true because otherwise a true resident matched to the level-j copy and h^{μ_H} form a blocking pair with respect to M_s. ◀

Proof of Lemma 9. In Lemma 7, we proved that each resident is matched to at most one hospital and each hospital h is matched to at most $q^+(h)$ many residents in G. Here, first we show that each $r \in \mathcal{R}_{lq}$ gets matched to at least one hospital and, then we show that each $h \in \mathcal{H}_{lq}$ gets matched to at least $q^-(h)$ many residents.

Let us assume for the sake of contradiction that M is not resident-feasible, and hence $M(r) = \perp$ for an $r \in \mathcal{R}_{lq}$, but there exists a feasible matching N in G where r is matched. Consider the decomposition of $M \oplus N$ into (possibly non-simple) alternating paths and cycles. The decomposition we use is the same as the one used in [21, 20]. As r is unmatched in M but matched in N there must exist an alternating path ρ in $M \oplus N$ ending at r. Moreover, the highest level copy r^{μ_R} must remain unmatched in M_s by Part 2 of Lemma 7.

Case 1: The other end-point of ρ is a resident r_k: Let $\rho = \langle r, h_1, r_1, h_2, r_2, \ldots, h_k, r_k \rangle$, where $(h_i, r_i) \in M$ and rest of the edges are in N. We show that such a path cannot exist and hence M must be feasible. The length of this path is even and hence r_k remains unmatched in N. It implies that r_k is a non-lower quota resident. Since we do not have multiple copies of a non-lower quota resident r, r_k^0 is matched to a non-dummy copy of a hospital h_k in M_s. Since r^{μ_R} is unmatched in M_s, all the residents $r \in M_s(h_1)$, and hence r_1, must also be the highest level copy. If not, then $(r^{\mu_R}, h_1^0$ blocks M_s because r^{μ_R} is unmatched and any copy of h_1 prefers r^{μ_R} over lower level copy of any resident. The copy of r_2 which is matched to h_2 must be either $r_2^{\mu_R}$ or $r_2^{\mu_R-1}$, otherwise, $(h_2^0, r_1^{\mu_R-1})$ blocks M_s. Continuing in this way, we see that the matched copy of r_3 must be in $\{r_3^{\mu_R}, r_3^{\mu_R-1}, r_3^{\mu_R-2}\}$, and matched copy of r_k must be in $\{r_k^{\mu_R}, r_k^{\mu_R-1}, \ldots, r_k^{\mu_R-(k-1)}\}$.

This implies that the path ρ goes downwards to level 0 but by at most one level for each resident on ρ. Since r_k is the 0^{th} level copy, ρ must contain at least $\mu_R + 1$ residents with non-zero lower quota. But there are only μ_R residents with lower quota 0. So such a path ρ cannot exist.

Case 2: The other end of ρ is a hospital h: Let $\rho = \langle r, h_1, r_1, h_2, r_2, \ldots, h_k, r_k, h\rangle$, where $(h_i, r_i) \in M$ and rest of the edges are in N. It is clear that $|M(h)| < |N(h)| \leq q^+(h)$, that is, h is under-subscribed in M. Consider the first r_p on ρ such that r_p^0 is active in M_s. Such an r_p must exist as proved in Claim 14 below. So, consider the sub-path $\rho' = r, h_1, r_1, \ldots, r_p$ of ρ. Using the same argument as in the previous case, ρ' must contain at least $\mu_R + 1$ lower-quota residents. Therefore ρ', and consequently ρ cannot exist.

▷ Claim 14. An alternating path ρ as considered in Case 2 above contains a resident r_p such that r_p^0 is active in M_s.

Proof of Claim 14. We consider the following two cases: (a) when h is only active at level 0 and, (b) when h is active at higher levels. In the first case, as h is under-subscribed in M_s, r_k^0 must be active in M_s. This is because if r_k^i is active for any $i > 0$ then (r_k^0, h^0) blocks M_s. In the second case, from Lemma 8, r_k^j cannot be active for any $j > 0$. So r_k^0 must be active. ◁

Now, we prove Part 2 of Lemma 9 by contradiction. Let us assume that M is not hospital-feasible but there exists a matching N which is hospital-feasible. That is, there exists $h \in \mathcal{H}_{lq}$ such that $|M(h)| < q^-(h) \leq |N(h)|$. Consider the decomposition of $M \oplus N$ into (possibly non-simple) alternating paths and cycles.

As $|M(h)| < |N(h)|$ there must exists a path ρ in $M \oplus N$ ending at h. Since h is deficient in M, the highest level copy h^{μ_H} must remain under-subscribed in M_s by Part 2 of Lemma 7.

Case 1: The other end of ρ is a hospital h_k: Let $\rho = \langle h, r_1, h_1, \ldots, r_{k-1}, h_{k-1}, r_k, h_k\rangle$, where $(r_i, h_i) \in M$ and rest of the edges are in N. We show that such a path cannot exist and hence M must be feasible. The length of this path ρ is even and $|M(h_k)| > |N(h_k)| \geq q^-(h_k)$. This implies that the higher level copies (level p for $p > 0$) of h_k are not active (Part 4 of Lemma 7). As h^{μ_H} remains under-subscribed in M_s, the copy of hospital h_1 which is matched to r_1 must be $h_1^{\mu_H}$, otherwise, (r_1^0, h^{μ_H}) forms a blocking pair w.r.t. M_s. From Part 4 of Lemma 7, $h_1^{\mu_H - p}$ for $p > 1$ cannot be active in M_s. Similarly, the copies $h_2^{\mu_H - p}$ for $p > 2$ of h_2 cannot be active in M_s. And, the copies $h_k^{\mu_H - p}$ for $p > k$ cannot be active in M_s. In other words, the only active copy of h is h^{μ_H}, the active copies of h_1 are in $\{h_1^{\mu_H}, h_1^{\mu_H - 1}\}$, the active copies of h_2 are in $\{h_2^{\mu_H}, h_2^{\mu_H - 1}, h_2^{\mu_H - 2}\}$ and so on. This implies that we may go downwards in this way but by at most one level for each hospital on ρ. As the only active copy of h_k is h_k^0 and hence, ρ must contain at least $\mu_H + 1$ copies of lower quota hospitals but the sum of all the lower quotas of hospitals is only μ_H. This is a contradiction.

Case 2: Other end of ρ is a resident r: Let $\rho = \langle h, r_1, h_1, r_2, h_2, \ldots, r_k, h_k, r\rangle$, where $(r_i, h_i) \in M$ and rest of the edges are in N. We know that the only active copy of h is h^{μ_H}. Here, r is unmatched in M. If $q^-(r) = 0$ then r^0 remains unmatched in M_s and hence, h_k cannot be active at level above 0. In this case, the same argument as in the previous case suffice to prove that such a path ρ cannot exist. The other case, when $q^-(r) = 1$, is not possible because of Part 1, which says that M is resident feasible but r is unmatched. ◀

Property B: Two-Coloring Non-Uniform Hypergraphs

Jaikumar Radhakrishnan ✉
School of Technology and Computer Science,
Tata Institute of Fundamental Research, Mumbai, India

Aravind Srinivasan ✉ 🏠
Department of Computer Science and UMIACS,
University of Maryland at College Park, MD, USA

Abstract

The following is a classical question of Erdős (*Nordisk Matematisk Tidskrift,* 1963) and of Erdős and Lovász (*Colloquia Mathematica Societatis János Bolyai,* vol. 10, 1975). Given a hypergraph $\mathcal{F}$ with minimum edge-size k, what is the largest function $g(k)$ such that if the expected number of monochromatic edges in $\mathcal{F}$ is at most $g(k)$ when the vertices of $\mathcal{F}$ are colored *red* and *blue* randomly and independently, then we are guaranteed that $\mathcal{F}$ is two-colorable? Duraj, Gutowski and Kozik (*ICALP 2018*) have shown that $g(k) \geq \Omega(\log k)$. On the other hand, if $\mathcal{F}$ is k-uniform, the lower bound on $g(k)$ is much higher: $g(k) \geq \Omega(\sqrt{k/\log k})$ (Radhakrishnan and Srinivasan, *Rand. Struct. Alg.*, 2000). In order to bridge this gap, we define a family of locally-almost-uniform hypergraphs, for which we show, via the randomized algorithm of Cherkashin and Kozik (*Rand. Struct. Alg.*, 2015), that $g(k)$ can be much higher than $\Omega(\log k)$, e.g., $2^{\Omega(\sqrt{\log k})}$ under suitable conditions.

2012 ACM Subject Classification Mathematics of computing → Combinatoric problems

Keywords and phrases Hypergraph coloring, Propery B

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.31

Funding *Jaikumar Radhakrishnan*: Supported by the Department of Atomic Energy, Government of India, under project no. RTI4001.
Aravind Srinivasan: Supported in part by NSF grants CCF-1422569 and CCF-1749864.

Acknowledgements We thank the reviewers of the current and previous versions of this paper for their constructive suggestions that helped us correct some errors and improve the presentation.

1 Introduction

A classical question of Erdős and of Erdős and Lovász is as follows [7, 9]. Given a hypergraph $\mathcal{F}$, let us define its minimum edge-size k as the (asymptotic) parameter of interest. Note that if the vertices of $\mathcal{F}$ are colored *red* and *blue* randomly and independently, then the expected number of of monochromatic edges in $\mathcal{F}$ is

$$M(\mathcal{F}) \doteq \sum_{f \in \mathcal{F}} 2^{1-|f|}.$$

(We view $\mathcal{F}$ as a collection of hyper-edges, hence the notation "$f \in \mathcal{F}$". Also, the constant multiplier "2" in the "$2^{1-|f|}$" is often not very important in our context, and we will typically study the sum $\sum_{f \in \mathcal{F}} 2^{-|f|}$.) Then, what is the largest function $g(k)$ such that if $M(\mathcal{F}) \leq g(k)$, then we are guaranteed that $\mathcal{F}$ is two-colorable? $\mathcal{F}$ was defined to have *Property B* by Erdős when it is two-colorable – in honor of F. Bernstein, who had considered the problem earlier [3] – and such questions on sufficient conditions for $\mathcal{F}$ to possess Property B have been studied quite a bit since the 1970s. In this work, we show improved sufficient conditions when $\mathcal{F}$ is *locally-almost-uniform* – specifically, λ-*approximately-uniform* as defined later – for a large range of the local-uniformity λ.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 31; pp. 31:1–31:8
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We start by reviewing prior work. A simple union bound shows that any $g(k) < 1$ will suffice, but the question is whether $g(k)$ can tend to infinity as k grows, and how fast g can grow. Beck [2] was the first to show that g can be allowed to grow with k, by proving that $g(k) \geq \Omega(\log^* k)$. The next improvement came recently, when Duraj, Gutowski and Kozik showed that $g(k) \geq \Omega(\log k)$ [6]. (If $\mathcal{F}$ is simple, i.e., if any two distinct edges intersect in at most one vertex, then $g(k) \geq \Omega(\sqrt{k})$ [11].) Much better lower bounds on $g(k)$ are known if $\mathcal{F}$ is k-uniform: $g(k) \geq \Omega(\sqrt{k/\log k})$ here [10] – see Cherkashin and Kozik [5] for a simpler proof of this result. On the other hand, $g(k) \leq O(k^2)$, even for k-uniform hypergraphs (Erdős [8]). It has been conjectured for long that $g(k)$ could be $\Theta(k)$, at least for uniform hypergraphs [9]. This remains a tantalizing open problem. Hypergraph two-coloring has also been studied for models of random hypergraphs, and from the viewpoint of inapproximability (see, e.g., [1, 4]).

Note the large gap between the lower bound of $\Omega(\log k)$ [6] and the lower bound of $\Omega(\sqrt{k/\log k})$ that holds for k-uniform hypergraphs [10, 5]. How can we bridge this gap? One approach is to study "how much uniformity" we need in order to get good lower bounds on $g(k)$: to this end, we define a family of locally-almost-uniform hypergraphs, for which we show that $g(k)$ can be much higher than $\Omega(\log k)$. Our randomized algorithm is the same as that of Cherkashin and Kozik [5], but our analysis is different. For $\lambda \geq 1$, we say that $\mathcal{F}$ is λ-approximately-uniform if

$$\max_{f,f' \in \mathcal{F}:\ |f \cap f'|=1} \frac{|f'|}{|f|} \leq \lambda.$$

That is, we require the local-almost-uniformity property that any two edges that intersect in exactly one vertex, have their size-ratio bounded by λ. (Note that this is asking less than requiring that any two intersecting edges have their size-ratio bounded by λ.)

Let $\exp(x)$ denote e^x. The following definition and lemma will be useful in the context of Theorem 3.

▶ **Definition 1.** ***(Parameter*** γ***)*** *For $\lambda, \alpha, k \geq 1$ such that $\alpha, \lambda \leq k$, we define $\gamma(k, \alpha, \lambda) \geq 0$ by its square:*

$$\gamma(k,\lambda,\alpha)^2 \doteq \min_{\lambda' \in [1,\lambda]} 2((\lambda'-1)k+1)\left[\left(1+\frac{\alpha}{k}\right)^{(\lambda'-1)k+1} - \left(1-\frac{\alpha}{\lambda' k}\right)^{(\lambda'-1)k+1)}\right]^{-1}.$$

Note that $\gamma(k, 1, \alpha) = \sqrt{k/\alpha}$; for larger values of λ, we will use the following lower bound on $\gamma(k, \alpha, \lambda)^2$ when we apply our main theorem to specific cases.

▶ **Lemma 2.** $\gamma(k,\lambda,\alpha)^2 \geq \left(\frac{k}{\alpha}\right)\exp(-\alpha\lambda)$.

Proof. We provide a proof in the appendix. ◀

We next present our main theorem followed by some of its consequences, following which we develop its proof.

▶ **Theorem 3.** *Let k be a positive integer and let $\lambda, \alpha \geq 1$ such that $2\alpha^2\lambda \leq k$. Let $\gamma(k, \lambda, \alpha)$ be as in Definition 1. Let $\mathcal{F}$ be any λ-approximately-uniform hypergraph with $k = \min_{f \in \mathcal{F}} |f|$. Then, $\mathcal{F}$ is two-colorable if*

$$\sum_{f \in \mathcal{F}} 2^{-|f|} \leq \frac{1}{4}\min\{\exp(\alpha), \gamma(k,\lambda,\alpha)\};$$

furthermore, such a coloring can be obtained in randomized polynomial time via the algorithm of Cherkashin and Kozik [5].

Note that the above theorem yields the bound obtained by the authors [10] for uniform hypergraphs. Indeed, if we set $\lambda = 1$ and $\alpha = \frac{1}{2}\ln\frac{k}{\ln k}$ (recall that $\gamma(k,1,\alpha) = \sqrt{k/\alpha}$), the above theorem implies that every k-uniform hypergraph with at most $(1/4)\sqrt{k/\ln k} \times 2^k$ edges is two-colorable – the constant $1/4$ can be improved slightly. This is not surprising, for the randomized algorithm we use to establish Theorem 3 reduces to the algorithm of Cherkashin and Kozik [5], and therefore yields the same bound with the same constant for k-uniform hypergraphs. (A minor subtlety is that $\lambda = 1$ does not imply k-uniformity: we can have $\lambda = 1$ and still allow two edges that intersect at two vertices or more, to have different sizes. However, such pairs of edges can often be ignored, as shown by the analyses of [10, 5] for uniform hypergraphs.)

For larger λ, we may use Lemma 2 to conclude that the hypergraph is two-colorable provided (for some choice of $\alpha \geq 1$)

$$\sum_{f\in\mathcal{F}} 2^{-|f|} \leq \frac{1}{4}\min\{\exp(\alpha), \sqrt{k/\alpha}\exp(-\alpha\lambda/2)\}.$$

Some illustrative examples (assume k is large):

- if $\lambda = 10$ and we set $\alpha = (\ln(k/\ln k))/12$, then we conclude that such a 10-approximately uniform hypergraph is two-colorable whenever
$$\sum_{f\in\mathcal{F}} 2^{-|f|} \leq \left(\frac{1}{4}\right)\left(\frac{k}{\ln k}\right)^{1/12};$$
- if $\lambda = \sqrt{\ln k}$ and we set $\alpha = \sqrt{\ln k}/2$, then we conclude that a λ-approximately-uniform hypergraph is two-colorable whenever
$$\sum_{f\in\mathcal{F}} 2^{-|f|} \leq \frac{1}{4}\exp(\sqrt{\ln k}/2);$$
- in general, we get nontrivial results for all $\lambda = o(\ln k)$.

2 Proof of the main result

Fix a hypergraph $\mathcal{F}$ satisfying the assumptions of Theorem 3. Let us use the following two-step randomized strategy due to Cherkashin and Kozik [5] that starts with all vertices uncolored.

Step 1: To each vertex v of the hypergraph independently assign a uniformly-random delay $\eta(v)$ from $[0,1]$.

Step 2: One by one, color the vertices using colors $\{$*blue*, *red*$\}$ in increasing order of their delays. Color a vertex v, when its turn comes, *red* if there exists some edge e containing v such that v is the last vertex to be colored in e, and such that all other vertices in e have already been colored *blue*; else color v *blue*.

As in Cherkashin and Kozik [5], we have the following observation: if an edge is left monochromatic at the end, then all its vertices must be colored *red*. In particular, suppose the vertices in f were colored *red* in the order $v_1, v_2, \ldots, v_r$, then there must be an edge e such that

$$|e \cap f| = 1, \text{ and } v_1 \text{ is the last vertex to be colored in } e.$$

In such a case, we say that f *blames* e.

Specifically, f blames e iff the following three conditions hold:

- $|e \cap f| = 1$ with $e \cap f = \{v\}$, say;
- $\eta(v) > \eta(u)$ for all other vertices u in e;
- $\eta(v) < \eta(w)$ for all other vertices w in f.

Thus, the probability that the above algorithm fails to two-color the hypergraph is at most

$$\Pr[\exists e, f : e \text{ blames } f]. \tag{1}$$

It is tempting to note that

$$\Pr[e \text{ blames } f] = \frac{(|e|-1)! \cdot (|f|-1)!}{(|e|+|f|-1)!}$$

and apply the union bound

$$\Pr[\exists e, f : e \text{ blames } f] \leq \sum_{(e,f)} \Pr[e \text{ blames } f].$$

However, a sum such as in the RHS of this union bound can be too large. Cherkashin and Kozik [5] suggest a nuanced approach in a similar situation; we adapt their approach to obtain a better bound for (1). The following claim is the main technical contribution of this work. Recall that $|e|/|f|,\ |f|/|e| \leq \lambda$ whenever e blames f.

▷ Claim 4.

$$\Pr[\exists e, f : e \text{ blames } f] \leq 2\sum_{e} 2^{-|e|} \exp(-\alpha) + 4 \sum_{(e,f):|e\cap f|=1} 2^{-|e|-|f|}\gamma(k,\lambda,\alpha)^{-2}.$$

Let us assume this claim and complete the proof of our theorem.

Proof of Theorem 3. By the assumption in our theorem, we have

$$\sum_{f\in\mathcal{F}} 2^{-|f|} \leq \frac{1}{4}\min\{\exp(\alpha), \gamma(k,\lambda,\alpha)\}.$$

So,

$$2\sum_{e} 2^{-|e|} \exp(-\alpha) \leq \frac{1}{2},$$

and

$$4 \sum_{(e,f):|e\cap f|=1} 2^{-|e|-|f|}\gamma(k,\lambda,\alpha)^{-2} \leq 4\left(\sum_{e} 2^{-|e|}\gamma(k,\lambda,\alpha)^{-1}\right)^{2} \leq \frac{1}{4}.$$

It follows from Claim 4 that the algorithm of Cherkashin and Kozik, outlined above, properly two-colors the hypergraph with probability at least $\frac{1}{4}$. The theorem follows from this. ◀

We now establish Claim 4.

Proof of Claim 4: For each edge e, let

$$\delta(e) \doteq \alpha/(2|e|).$$

Fix e and f with $e \cap f = \{v\}$, and define events

$$\begin{aligned}\mathcal{E}_1(e) &\equiv \forall u \in e : \eta(u) < \frac{1}{2} - \delta(e);\\ \mathcal{E}_2(f) &\equiv \forall w \in f : \eta(w) > \frac{1}{2} + \delta(f);\\ \mathcal{E}_3(e,f) &\equiv (f \text{ blames e}) \text{ and } \neg\mathcal{E}_1(e) \text{ and } \neg\mathcal{E}_2(f).\end{aligned}$$

Then, we have

$$(f \text{ blames } e) \subseteq \mathcal{E}_1(e) \cup \mathcal{E}_2(f) \cup \mathcal{E}_3(e,f). \tag{2}$$

We bound the probability of each of these three events separately. For the first two events we have

$$\begin{aligned}\Pr[\mathcal{E}_1(e)] &= \left(\frac{1}{2} - \delta(e)\right)^{|e|} \leq 2^{-|e|}\exp(-2\delta(e)|e|) \leq 2^{-|e|}\exp(-\alpha)\,;\\ \Pr[\mathcal{E}_2(f)] &= \left(\frac{1}{2} - \delta(f)\right)^{|f|} \leq 2^{-|f|}\exp(-2\delta(f)|f|) \leq 2^{-|f|}\exp(-\alpha).\end{aligned}$$

To bound the probability of the third event, namely $\mathcal{E}_3(e,f)$, note that if both $\neg\mathcal{E}_1(e)$ and $\neg\mathcal{E}_2(f)$ hold, then $\eta(v) \in [\frac{1}{2} - \delta(e), \frac{1}{2} + \delta(f)]$. We condition on $\eta(v) = \frac{1}{2} + x$ and integrate to obtain

$$\begin{aligned}\Pr[\mathcal{E}_3(e,f)] &\leq \int_{-\delta(e)}^{\delta(f)} \left(\frac{1}{2} + x\right)^{|e|-1} \left(\frac{1}{2} - x\right)^{|f|-1} \mathrm{d}x\\ &= 2^{-|e|-|f|+2} \int_{-\delta(e)}^{\delta(f)} (1+2x)^{|e|-1}(1-2x)^{|f|-1}\,\mathrm{d}x\\ &\leq 2^{-|e|-|f|+2} \cdot \beta(e,f),\end{aligned}$$

where

$$\beta(e,f) = \begin{cases} \int_{-\delta(f)}^{\delta(e)} (1+2x)^{|f|-|e|}\,\mathrm{d}x & \text{if } |e| \leq |f|;\\ \int_{-\delta(e)}^{\delta(f)} (1+2x)^{|e|-|f|}\,\mathrm{d}x & \text{if } |e| > |f|.\end{cases} \tag{3}$$

(Both bounds for $\beta(e,f)$ follow from the fact that $(1+2x)\cdot(1-2x) \leq 1$.)

We next show the following.

▷ Claim 5. For all large k, we have

$$\beta(e,f) \leq \max_{\lambda' \in [0,\lambda]} \frac{1/2}{(\lambda'-1)k+1}\left[\left(1+\frac{\alpha}{k}\right)^{(\lambda'-1)k+1} - \left(1 - \frac{\alpha}{\lambda' k}\right)^{(\lambda'-1)k+1}\right] \tag{4}$$

Note that the right hand side of Equation (4) is precisely $\gamma(k,\lambda,\alpha)^{-2}$. Claim 4 then follows from bound (2). ◁

Proof of Claim 5. Suppose $|f| \geq |e|$ (the case $|f| < |e|$ is similar). Set $|f| = \lambda'|e|$; note that $1 \leq \lambda' \leq \lambda$. Then,

$$\beta(e,f) = \frac{1/2}{(\lambda'-1)|e|+1}\left[\left(1+\frac{\alpha}{|e|}\right)^{(\lambda'-1)|e|+1} - \left(1-\frac{\alpha}{\lambda'|e|}\right)^{(\lambda'-1)|e|+1}\right]$$

Let

$$h_{\lambda'}(x) \doteq \frac{1}{(\lambda'-1)x+1}\left[\left(1+\frac{\alpha}{x}\right)^{(\lambda'-1)x+1} - \left(1-\frac{\alpha}{\lambda' x}\right)^{(\lambda'-1)x+1}\right]$$

To establish our claim, we will show that $h_{\lambda'}(x)$ is a decreasing function for x in the range $[k,\infty)$, so it is maximum for $x = k$. Indeed, the derivative $h'_{\lambda'}(x)$ is the sum of three terms:

$$-\frac{(\lambda'-1)}{((\lambda'-1)x+1)^2}\left[\left(1+\frac{\alpha}{x}\right)^{(\lambda'-1)x+1} - \left(1-\frac{\alpha}{\lambda' x}\right)^{(\lambda'-1)x+1}\right], \tag{5}$$

$$\frac{1}{(\lambda'-1)x+1}\left[\left(1+\frac{\alpha}{x}\right)^{(\lambda'-1)x+1}\left(\ln(1+\frac{\alpha}{x})(\lambda'-1) - ((\lambda'-1)x+1)\frac{\alpha}{x(\alpha+x)}\right)\right], \tag{6}$$

and

$$-\left(\frac{1}{(\lambda'-1)x+1}\right)\left(1-\frac{\alpha}{\lambda' x}\right)^{(\lambda'-1)x+1} \times \left(\ln(1-\frac{\alpha}{\lambda' x})(\lambda'-1) + ((\lambda'-1)x+1)\frac{\alpha}{x(\lambda' x-\alpha)}\right). \tag{7}$$

We wish to show that this derivative is negative for $x \in [k,\infty)$. First, we show that term (7) is negative, but verifying that the three factors in parentheses are positive. The first two factors are clearly positive. We rearrange the last factor as

$$(\lambda'-1)\left(\ln(1-\frac{\alpha}{\lambda' x}) + \frac{\alpha}{\lambda' x-\alpha}\right) + \frac{\alpha}{x(\lambda' x-\alpha)}, \tag{8}$$

and using $\ln(1-\frac{\alpha}{\lambda' x}) = -\ln(1+\frac{\alpha}{\lambda' x-\alpha}) \geq \frac{-\alpha}{\lambda' x-\alpha}$, verify that this factor is positive as well.

We now deal with the first two terms. The first term (5) is at most

$$-\frac{(\lambda'-1)}{((\lambda'-1)x+1)^2}\left(1+\frac{\alpha}{x}\right)^{(\lambda'-1)x+1}\left[1-\left(1-\frac{\alpha}{\lambda' x}\right)^{(\lambda'-1)x+1}\right], \tag{9}$$

Using the inequality $\ln(1+\frac{\alpha}{x}) \leq \frac{\alpha}{x}$, we see that the second term (6) is at most

$$\frac{1}{(\lambda'-1)x+1}\left(1+\frac{\alpha}{x}\right)^{(\lambda'-1)x+1}\left((\lambda'-1)\frac{\alpha^2}{x(x+\alpha)} - \frac{\alpha}{x(x+\alpha)}\right) \tag{10}$$

$$\leq \frac{\lambda'-1}{(\lambda'-1)x+1}\left(1+\frac{\alpha}{x}\right)^{(\lambda'-1)x+1}\left(\frac{\alpha^2}{x(x+\alpha)}\right). \tag{11}$$

Thus, since the third term (7) is negative, the bounds (9) and (11) on the first two terms, (5) and (6), imply that the derivative $h'_{\lambda'}(x)$ is at most $\frac{\lambda'-1}{(\lambda'-1)x+1}\left(1+\frac{\alpha}{x}\right)^{(\lambda'-1)x+1}$ times

$$-\frac{1}{(\lambda'-1)x+1} + \frac{\alpha^2}{x(x+\alpha)} + \frac{1}{(\lambda'-1)x+1}\left(1-\frac{\alpha}{\lambda' x}\right)^{(\lambda'-1)x+1}$$

$$= -\frac{x(x+\alpha)(1-\left(1-\frac{\alpha}{\lambda' x}\right)^{(\lambda'-1)x+1}) - \alpha^2((\lambda'-1)x+1)}{((\lambda'-1)x+1)x(x+\alpha)}. \tag{12}$$

We will verify that the numerator of the fraction in Equation (12) is positive. First note that by the weighted AM-GM inequality $(1-\epsilon)^z(1+\epsilon z) \leq 1^{1+z} = 1$ (for $0 \leq \epsilon < 1$ and $z \geq 0$); thus, $1-(1-\epsilon)^z \geq \epsilon z/(1+\epsilon z)$. In the following let $z \doteq (\lambda'-1)x+1$ and $\epsilon \doteq \alpha/(\lambda' x)$; then, the numerator in Equation (12) is at least

$$\begin{aligned} x(x+\alpha)\frac{\alpha z/(\lambda' x)}{1+\alpha z/(\lambda' x)} - \alpha^2 z &= \frac{\alpha z}{\lambda' x + \alpha z}\left[x(x+\alpha) - \alpha(\lambda' x + \alpha z)\right] \\ &= \frac{\alpha z}{\lambda' x + \alpha z}\left[x(x - \alpha(\lambda' + \alpha\lambda' - \alpha)) + \alpha(x-\alpha)\right] \\ &= \frac{\alpha z}{\lambda' x + \alpha z}\left[x^2 - x(\alpha\lambda + \alpha^2\lambda - \alpha^2) + \alpha(x-\alpha)\right] \\ &\geq \frac{\alpha z}{\lambda' x + \alpha z}\left[x(x - 2\alpha^2\lambda) + x\alpha^2 + \alpha(x-\alpha)\right]. \end{aligned}$$

Our assumption $2\alpha^2\lambda \leq k \leq x$ implies that the last quantity, and hence the numerator of the fraction in Equation (12), is positive. Thus, $h_{\lambda'}(x)$ is decreasing over $[k,\infty)$ and

$$\beta(e,f) \leq \frac{1}{2}\max_{\lambda' \in [1,\lambda]} h_{\lambda'}(k).$$

Claim 5 follows from this. ◁

References

1 Dimitris Achlioptas, Jeong Han Kim, Michael Krivelevich, and Prasad Tetali. Two-coloring random hypergraphs. *Random Struct. Algorithms*, 20(2):249–259, 2002. doi:10.1002/rsa.997.

2 J. Beck. On 3-chromatic hypergraphs. *Discrete Mathematics*, 24:127–137, 1978.

3 F. Bernstein. Zur Theorie der trigonometrische Reihen. *Leipz. Ber.*, 60:325–328, 1908.

4 Amey Bhangale. Np-hardness of coloring 2-colorable hypergraph with poly-logarithmically many colors. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 15:1–15:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.15.

5 Danila D. Cherkashin and Jakub Kozik. A note on random greedy coloring of uniform hypergraphs. *Random Struct. Algorithms*, 47(3):407–413, 2015. doi:10.1002/rsa.20556.

6 Lech Duraj, Grzegorz Gutowski, and Jakub Kozik. A note on two-colorability of nonuniform hypergraphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 46:1–46:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.46.

7 P. Erdős. On a combinatorial problem, I. *Nordisk Matematisk Tidskrift*, 11:5–10, 1963.

8 P. Erdős. On a combinatorial problem, II. *Acta Mathematica of the Hungarian Academy of Sciences*, 15:445–447, 1964.

9 P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday)*, volume II, pages 609–627. North-Holland, Amsterdam, 1975. Volume 10 of Colloquia Mathematica Societatis János Bolyai.

10 Jaikumar Radhakrishnan and Aravind Srinivasan. Improved bounds and algorithms for hypergraph 2-coloring. *Random Struct. Algorithms*, 16(1):4–32, 2000. doi:10.1002/(SICI)1098-2418(200001)16:1\%3C4::AID-RSA2\%3E3.0.CO;2-2.

11 Dmitry A. Shabanov. Around erdős-lovász problem on colorings of non-uniform hypergraphs. *Discret. Math.*, 338(11):1976–1981, 2015. doi:10.1016/j.disc.2015.04.017.

A Proof of Lemma 2

Proof. Recall that

$$\gamma(k,\lambda,\alpha)^2 \doteq \min_{\lambda'\in[1,\lambda]} 2((\lambda'-1)k+1)\left[\left(1+\frac{\alpha}{k}\right)^{(\lambda'-1)k+1} - \left(1-\frac{\alpha}{\lambda' k}\right)^{(\lambda'-1)k+1)}\right]^{-1}.$$

We will show that reciprocal of the expression under the minimum is at most $(\frac{\alpha}{k})\exp(\alpha\lambda)$. It will follow that $\gamma(k,\lambda,\alpha)^2 \geq (\frac{k}{\alpha})\exp(-\alpha\lambda)$.

Let $z \doteq (\lambda'-1)k+1$. We will consider two cases, based on whether or not $2z\alpha \geq k$. First, suppose $2z\alpha \geq k$, that is, $z \geq k/(2\alpha)$. Then, we have the desired upper bound

$$\frac{1}{2z}\left[\left(1+\frac{\alpha}{k}\right)^z - \left(1-\frac{\alpha}{\lambda' k}\right)^z\right] \leq \frac{1}{2z}\left(1+\frac{\alpha}{k}\right)^z \leq \left(\frac{\alpha}{k}\right)\exp(\alpha z/k) \leq \left(\frac{\alpha}{k}\right)\exp(\alpha\lambda).$$

Next assume $2z\alpha < k$. We will use the following inequalities for bounding expressions of the form $(1+x)^\ell$. For $\ell \geq 1$ and $\ell|x| < 1$ (note x may be negative), we have $1+\ell x \leq (1+x)^\ell \leq 1/(1-\ell x)$. Then

$$\begin{aligned}
\frac{1}{2z}\left[\left(1+\frac{\alpha}{k}\right)^z - \left(1-\frac{\alpha}{\lambda' k}\right)^z\right] &\leq \frac{1}{2z}\left[\frac{1}{1-z\alpha/k} - \left(1-\frac{z\alpha}{\lambda' k}\right)\right] \quad (\text{note } z\alpha/k < \tfrac{1}{2})\\
&= \frac{1}{2z}\left(\frac{1-(1-z\alpha/k)(1-z\alpha/(\lambda' k))}{1-z\alpha/k}\right)\\
&\leq \frac{\alpha+\alpha/\lambda' - z\alpha^2/(\lambda' k)}{2(k-z\alpha)}\\
&\leq \frac{\alpha+\alpha/\lambda' - z\alpha^2/(\lambda' k)}{k} \quad (\text{since } 2z\alpha \leq k), \lambda' \geq 1)\\
&\leq \frac{2\alpha}{k}\\
&\leq \left(\frac{\alpha}{k}\right)\exp(\alpha\lambda). \quad (\text{since } \alpha,\lambda \geq 1)
\end{aligned}$$

◀

Harmonic Algorithms for Packing d-Dimensional Cuboids into Bins

Eklavya Sharma ✉
Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

Abstract

We explore approximation algorithms for the d-dimensional geometric bin packing problem (dBP). Caprara [8] gave a *harmonic-based* algorithm for dBP having an asymptotic approximation ratio (AAR) of T_∞^{d-1} (where $T_\infty \approx 1.691$). However, their algorithm doesn't allow items to be rotated. This is in contrast to some common applications of dBP, like packing boxes into shipping containers. We give approximation algorithms for dBP when items can be orthogonally rotated about all or a subset of axes. We first give a fast and simple harmonic-based algorithm having AAR T_∞^d. We next give a more sophisticated harmonic-based algorithm, which we call $\texttt{HGaP}_k$, having AAR $T_\infty^{d-1}(1+\varepsilon)$. This gives an AAR of roughly $2.860+\varepsilon$ for 3BP with rotations, which improves upon the best-known AAR of 4.5. In addition, we study the *multiple-choice* bin packing problem that generalizes the rotational case. Here we are given n sets of d-dimensional cuboidal items and we have to choose exactly one item from each set and then pack the chosen items. Our algorithms also work for the multiple-choice bin packing problem. We also give fast and simple approximation algorithms for the multiple-choice versions of dD strip packing and dD geometric knapsack.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases Geometric bin packing

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.32

Related Version *ArXiv*: `https://arxiv.org/abs/2011.10963`

Acknowledgements I want to thank my advisor, Prof. Arindam Khan, for his valuable comments, and Arka Ray for helpful suggestions.

1 Introduction

Packing of rectangular and cuboidal items is a fundamental problem in computer science, mathematics, and operations research. Packing problems find numerous applications in practice, e.g., packing of concrete 3D items during storage or transportation [7], cutting prescribed 2D pieces from cloth or metal sheet while minimizing the waste [16], etc. In this paper, we study packing of d-dimensional (dD) cuboidal items (for $d \geq 2$).

Let I be a set of n dD cuboidal items, where each item has length at most one in each dimension. A feasible packing of items into a dD cuboid is a packing where items are placed inside the cuboid parallel to the axes without any overlapping. A dD *unit cube* is a dD cuboid of length one in each dimension. In the dD bin packing problem (dBP), we have to compute a feasible packing of I (without rotating the items) into the minimum number of bins that are dD unit cubes. Let $\operatorname{opt}_{d\mathrm{BP}}(I)$ be the minimum number of bins needed to pack I.

dBP is NP-hard, as it generalizes the classic bin packing problem [9]. Thus, we study approximation algorithms. For dBP, the worst-case approximation ratio usually occurs only for *small* pathological instances. Thus, the standard performance measure is the asymptotic approximation ratio (AAR). For an algorithm $\mathcal{A}$, AAR is defined as:

$$\lim_{m\to\infty} \sup_{I\in\mathtt{I}:\,\operatorname{opt}(I)=m} \frac{\mathcal{A}(I)}{\operatorname{opt}(I)},$$

where $\mathtt{I}$ is the set of all problem instances. $\mathcal{A}(I)$ and $\operatorname{opt}(I)$ are the number of bins used by $\mathcal{A}$ and the optimal algorithm, respectively, on I.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 32; pp. 32:1–32:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Coffman et al. [10] initiated the study of approximation algorithms for rectangle packing. They studied algorithms such as First-Fit Decreasing Height (FFDH) and Next-Fit Decreasing Height (NFDH). In his seminal paper, Caprara [8] devised a polynomial-time algorithm for dBP called $\texttt{HDH}_k$ (Harmonic Decreasing Height), where $k \in \mathbb{Z}$ is a parameter to the algorithm. $\texttt{HDH}_k$ has AAR equal to T_k^{d-1}, where T_k is a decreasing function of k and $T_\infty := \lim_{k\to\infty} T_k \approx 1.691$. $\texttt{HDH}_k$ is based on the harmonic algorithm [24] for 1BP.

A limitation of $\texttt{HDH}_k$ is that it does not allow rotation of items. This is in contrast to some real-world problems, like packing boxes into shipping containers ($d = 3$), where items can often be rotated orthogonally, i.e., 90° rotation around all or a subset of axes [1, 31]. Orientation constraints may sometimes limit the vertical orientation of a box to one dimension ("This side up") or to two (of three) dimensions (e.g., long but low and narrow box should not be placed on its smallest surface). These constraints are introduced to deter goods and packaging from being damaged and to ensure the stability of the load. One of our primary contributions is presenting variants of $\texttt{HDH}_k$ that work for generalizations of dBP that capture the notion of orthogonal rotation of items.

1.1 Prior Work

For 2BP, Bansal et al. [3] obtained AAR of $T_\infty + \varepsilon$ even for the case with rotations, using a more sophisticated algorithm that used properties of harmonic rounding. Then there has been a series of improvements [3, 19] culminating with the present best AAR of 1.406 [5], for both the cases with and without orthogonal rotations. Bansal et al. [6] showed that dBP is APX-hard for $d \geq 2$, and gave an asymptotic PTAS for dBP when all items are dD squares.

Closely related to dBP is the dD strip packing problem (dSP), where we have to pack I (without rotating the items) into a dD cuboid (called a strip) that has length one in the first $d - 1$ dimensions and the minimum possible length (called height) in the d^{th} dimension.

For 2SP, an asymptotic PTAS was given by Kenyon and Rémila [22]. Jansen and van Stee [21] extended this to the case with orthogonal rotations. For 3SP, when rotations are not allowed, Bansal et al. [4] gave a harmonic-based algorithm achieving AAR of $T_\infty + \varepsilon$. Recently, this has been improved to $1.5 + \varepsilon$ [20]. Miyazawa and Wakabayashi [25] studied 3SP and 3BP when rotations are allowed, and gave algorithms with AAR 2.64 and 4.89, respectively. Epstein and van Stee [13] gave an improved AAR of 2.25 and 4.5 for 3SP and 3BP with rotations, respectively. The $\texttt{HDH}_k$ algorithm also works for dSP and has an AAR of T_k^{d-1}. For online dBP, there are harmonic-based T_∞^d-asymptotic-approximation algorithms [12, 11], which are optimal for $O(1)$ memory algorithms.

1.2 Multiple-Choice Packing

We will now define the dD multiple-choice bin packing problem (dMCBP). This generalizes dBP and captures the notion of orthogonal rotation of items. This perspective will be helpful in designing algorithms for the rotational case. In dMCBP, we're given a set $\mathcal{I} = \{I_1, I_2, \dots, I_n\}$, where for each j, I_j is a set of items, henceforth called an *itemset*. We have to pick exactly one item from each itemset and pack those items into the minimum number of bins. See Figure 1 for an example of 2MCBP.

We can model rotations using multiple-choice packing: Given a set I of items, for each item $i \in I$, create an itemset I_i that contains all allowed orientations of i. Then the optimal solution to $\mathcal{I} := \{I_i : i \in I\}$ will tell us how to rotate and pack items in I.

Some algorithms for 2D bin packing with rotations assume that the bin is square [3, 19, 5]. This assumption holds without loss of generality when rotations are forbidden, because we can scale the items. But if rotations are allowed, this won't work because items i_1 and i_2

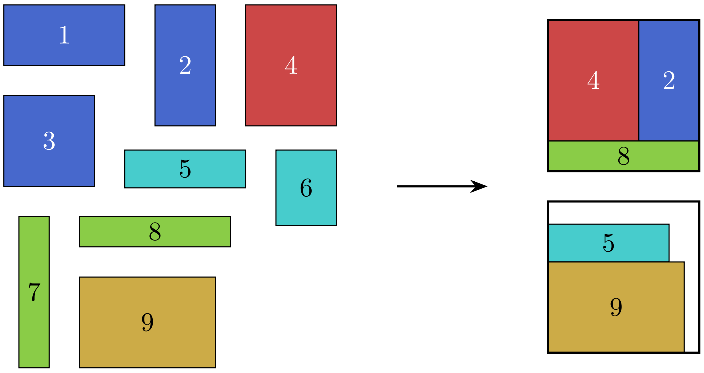


Figure 1 2MCBP example: packing the input $\mathcal{I} = \{\{1, 2, 3\}, \{4\}, \{5, 6\}, \{7, 8\}, \{9\}\}$ into two bins. Here items of the same color belong to the same itemset.

that are rotations of each other may stop being rotations of each other after they are scaled. Multiple-choice packing algorithms can be used in this case. For each item $i \in I$, we will create an itemset I_i that contains scaled orientations of i.

Multiple-choice packing problems have been studied before. Lawler gave an FPTAS for the multiple-choice knapsack problem [23]. Patt-Shamir and Rawitz gave an algorithm for multiple-choice vector bin packing having AAR $O(\log d)$ and a PTAS for multiple-choice vector knapsack [27]. Similar notions have been studied in the scheduling of malleable or moldable jobs [32, 18].

1.3 Our Contributions

After the introduction of the harmonic algorithm for online 1BP by Lee and Lee [24], many variants have found widespread use in multidimensional packing problems (both offline and online) [8, 3, 4, 2, 12, 11, 17, 28, 29]. They are also simple, fast, and easy to implement. For example, among algorithms for 3SP, 2BP and 3BP with practical running time, harmonic-based algorithms provide the best AAR.

In our work, we extend harmonic-based algorithms to dMCBP. dMCBP subsumes the rotational case for geometric bin packing, and we believe dMCBP is an important natural generalization of geometric bin packing that may be of independent interest.

In Section 3, we describe ideas from $\texttt{HDH}_k$ [8] that help us devise harmonic-based algorithms for dMCBP. In Section 4, we show an $O(Nd + nd \log n)$-time algorithm for dMCBP, called $\texttt{fullh}_k$, having an AAR of T_k^d, where n is the number of itemsets and N is the total number of items across all the n itemsets. $\texttt{fullh}_k$ is a fast and simple algorithm that works in two stages: In the first stage, we select the *smallest* item from each itemset (we will precisely define *smallest* in Section 4). In the second stage, we pack the selected items into bins using a variant of the $\texttt{HDH}_k$ algorithm.

In Section 5, we show an algorithm for dMCBP, called $\texttt{HGaP}_k$, having an AAR of $T_k^{d-1}(1+\varepsilon)$ and having a running time of $N^{O(1/\varepsilon^2)} n^{(1/\varepsilon)^{O(1/\varepsilon)}} + O(Nd + nd \log n)$. For $d \geq 3$, this matches the present best AAR for the case where rotations are forbidden. Also, for large k, this gives an AAR of roughly $T_\infty^2 \approx 2.860$ for 3D bin packing when orthogonal rotations are allowed, which is an improvement over the previous best AAR of 4.5 [13], an improvement after fourteen years.

Our techniques can be extended to some other packing problems, like strip packing and geometric knapsack. In Appendix C of the full version of our paper [30], we define the dD multiple-choice strip packing problem (dMCSP) and extend Caprara's $\texttt{HDH}_k$ algorithm [8] to

dMCSP. The algorithm has AAR T_k^{d-1} and runs in time $O(Nd + nd\log n)$, where n is the number of itemsets and N is the total number of items across all itemsets. In Appendix D of [30], we define the dD multiple-choice knapsack problem (dMCKS), and for any $0 < \varepsilon < 1$, we show an $O(Nd+N\log N+Nn/\varepsilon+nd\log n)$-time algorithm that is $(1-\varepsilon)3^{-d}$-approximate.

2 Preliminaries

Let $[n] := \{1, 2, \dots, n\}$. For a set X, define $\operatorname{sum}(X) := \sum_{x\in X} x$. For an n-dimensional vector $\mathbf{v}$, define $\operatorname{sum}(\mathbf{v}) := \sum_{i=1}^{n} \mathbf{v}_i$. For a set $X \subseteq I$ of items and any function $f : I \mapsto \mathbb{R}$, $f(X)$ is defined to be $\sum_{i\in X} f(i)$, unless stated otherwise.

The length of a dD item i in the j^{th} dimension is denoted by $\ell_j(i)$. Define $\operatorname{vol}(i) := \prod_{j=1}^{d} \ell_j(i)$. For a dD cuboid i, call the first $d-1$ dimensions *base dimensions* and call the d^{th} dimension *height*. For a set I of items, $|I|$ is the number of items in I. Let $|P|$ denote the number of bins used by a packing P of items into bins.

▶ **Lemma 1.** *Consider the inequality $x_1 + x_2 + \ldots + x_n \le s$, where for each $j \in [n]$, $x_j \in \mathbb{Z}_{\ge 0}$. Let N be the number of solutions to this inequality. Then $N = \binom{s+n}{n} \le (s+1)^n$.*

Proof. The proof of $N = \binom{s+n}{n}$ is a standard result in combinatorics.

To prove $N \le (s+1)^n$, note that we can choose each $x_j \in \{0, 1, \dots, s\}$ independently. ◀

2.1 Multiple-Choice Packing

Let $\mathcal{I}$ be a set of itemsets. Define $\operatorname{flat}(\mathcal{I})$ to be the union of all itemsets in $\mathcal{I}$.

Let K be a set of items that contains exactly one item from each itemset in $\mathcal{I}$. Formally, for each itemset $I \in \mathcal{I}$, $|K \cap I| = 1$. Then K is called an *assortment* of $\mathcal{I}$. Let $\Psi(\mathcal{I})$ denote the set of all assortments of $\mathcal{I}$. In dMCBP, given an input instance $\mathcal{I}$, we have to select an assortment $K \in \Psi(\mathcal{I})$ and output a bin packing of K, such that the number of bins used is minimized. Therefore, $\operatorname{opt}_{d\text{MCBP}}(\mathcal{I}) = \min_{K\in\Psi(\mathcal{I})} \operatorname{opt}_{d\text{BP}}(K)$.

3 Important Ideas from the $\texttt{HDH}_k$ Algorithm

In this section, we will describe some important ideas behind the $\texttt{HDH}_k$ algorithm for dBP by Caprara [8]. These ideas are the building blocks for our algorithms for dMCBP.

3.1 Weighting Functions

Fekete and Schepers [14] present a useful approach for obtaining lower bounds on the optimal solution to bin packing problems. Their approach is based on *weighting functions*.

▶ **Definition 2.** *$g : [0, 1] \mapsto [0, 1]$ is a weighting function iff for all $m \in \mathbb{Z}_{>0}$ and $x \in [0, 1]^m$,*

$$\sum_{i=1}^{m} x_i \le 1 \implies \sum_{i=1}^{m} g(x_i) \le 1$$

(Weighting functions are also called dual feasible functions *(DFFs)).*

▶ **Theorem 3.** *Let I be a set of dD items that can be packed into a bin. Let $g_1, g_2, \dots, g_d$ be weighting functions. For $i \in I$, define $g(i)$ as the item whose length is $g_j(\ell_j(i))$ in the j^{th} dimension, for each $j \in [d]$. Then $\{g(i) : i \in I\}$ can be packed into a dD bin (without rotating the items).*

Theorem 3 is proved in Appendix E of [30].

3.2 The Harmonic Function

To obtain a lower-bound on $\operatorname{opt}_{d\mathrm{BP}}(I)$ using Theorem 3, Caprara [8] defined a function f_k. For an integer constant $k \ge 3$, $f_k : [0,1] \mapsto [0,1]$ is defined as

$$f_k(x) := \begin{cases} \frac{1}{q} & x \in \left(\frac{1}{q+1}, \frac{1}{q}\right] \quad \text{for } q \in [k-1] \\ \frac{k}{k-2}x & x \le \frac{1}{k} \end{cases}.$$

f_k was originally defined and studied by Lee and Lee [24] for their online algorithm for 1BP, except that they used $k/(k-1)$ instead of $k/(k-2)$. Define $\operatorname{type}_k : [0,1] \mapsto [k]$ as

$$\operatorname{type}_k(x) := \begin{cases} q & x \in \left(\frac{1}{q+1}, \frac{1}{q}\right] \quad \text{for } q \in [k-1] \\ k & x \le \frac{1}{k} \end{cases}.$$

Define T_k to be the smallest positive constant such that $H_k(x) := f_k(x)/T_k$ is a weighting function. We call H_k the *harmonic weighting function*. We can efficiently compute T_k as a function of k using ideas from [24]. Table 1 lists the values of T_k for the first few k. It can also be proven that T_k is a decreasing function of k and $T_\infty := \lim_{k\to\infty} T_k \approx 1.6910302$.

Table 1 Values of T_k.

k	3	4	5	6	7	∞
T_k	3	2	$11/6 = 1.8\overline{3}$	$7/4 = 1.75$	$26/15 = 1.7\overline{3}$	≈ 1.6910302

For a dD cuboid i, define $f_k(i)$ to be the cuboid whose length is $f_k(\ell_j(i))$ in the j^{th} dimension, for each $j \in [d]$. For a set I of dD cuboids, let $f_k(I) := \{f_k(i) : i \in I\}$. Similarly define $H_k(i)$ and $H_k(I)$. Define $\operatorname{type}(i)$ to be a d-dimensional vector whose j^{th} component is $\operatorname{type}_k(\ell_j(i))$. Note that there can be at most k^d different values of $\operatorname{type}(i)$. Sometimes, for the sake of convenience, we may express $\operatorname{type}(i)$ as an integer in $[k^d]$.

▶ **Theorem 4.** *For a set of I of dD items,* $\operatorname{vol}(f_k(I)) \le T_k^d \operatorname{opt}_{d\mathrm{BP}}(I)$.

Proof. Let $m := \operatorname{opt}_{d\mathrm{BP}}(I)$. Let J_j be the items in the j^{th} bin in the optimal bin packing of I. By Theorem 3 and because H_k is a weighting function, $H_k(J_j)$ fits in a bin. Therefore,

$$\operatorname{vol}(f_k(I)) = \sum_{j=1}^{m} T_k^d \operatorname{vol}(H_k(J_j)) \le \sum_{j=1}^{m} T_k^d = T_k^d \operatorname{opt}_{d\mathrm{BP}}(I). \qquad ◀$$

3.3 The HDH-unit-pack$_k$ Subroutine

From the HDH$_k$ algorithm by Caprara [8], we extracted out a useful subroutine, which we call HDH-unit-pack$_k$, that satisfies the following useful property:

▶ **Property 5.** *The algorithm* HDH-unit-pack$_k^{[t]}(I)$ *takes a* sequence I *of dD items such that all items have type t and* $\operatorname{vol}(f_k(I - \{\operatorname{last}(I)\})) < 1$ *(here* $\operatorname{last}(I)$ *is the last item in sequence I). It returns a packing of I into a single dD bin in $O(nd \log n)$ time, where $n := |I|$.*

We use HDH-unit-pack$_k$ as a black-box subroutine in our algorithms, i.e., HDH-unit-pack$_k$ can be replaced by any algorithm that satisfies Property 5. See Appendix B of [30] for a complete description of HDH-unit-pack$_k$ and proof that it satisfies Property 5.

4 Fast and Simple Algorithm for dMCBP ($\texttt{fullh}_k$)

We will now describe an algorithm for dBP called the *full-harmonic algorithm* ($\texttt{fullh}_k$). We will then extend it to dMCBP. The $\texttt{fullh}_k$ algorithm works by first partitioning the items based on their type vector (type vector is defined in Section 3.2). Then for each partition, it repeatedly picks the smallest prefix J such that $\operatorname{vol}(f_k(J)) \geq 1$ and packs J into a dD bin using $\texttt{HDH-unit-pack}_k$. See Algorithm 1 for a more precise description of $\texttt{fullh}_k$. Note that $\texttt{fullh}_k(I)$ has a running time of $O(|I| d \log |I|)$.

Algorithm 1 $\texttt{fullh}_k(I)$: Returns a bin packing of dD items I.

```
Let $P$ be an empty list.
for each type $t$ do
    $I^{[t]} = \{i \in I : \operatorname{type}(i) = t\}$.
    while $|I^{[t]}| > 0$ do
        Find $J$, the smallest prefix of $I^{[t]}$ such that $J = I^{[t]}$ or $\operatorname{vol}(f_k(J))) \geq 1$.
        $B = \texttt{HDH-unit-pack}_k^{[t]}(J)$.                    // $B$ is a packing of $J$ into a $d$D bin.
        Append $B$ to the list $P$.
        Remove $J$ from $I^{[t]}$.
    end while
end for
return the list $P$ of bins.
```

▶ **Theorem 6.** *The number of bins used by* $\texttt{fullh}_k(I)$ *is less than* $Q + \operatorname{vol}(f_k(I))$*, where* Q *is the number of distinct* types *of items (so* $Q \leq k^d$*).*

Proof. Let $I^{[t]}$ be the items in I of type t. Suppose $\texttt{fullh}_k(I)$ uses $m^{[t]}$ bins to pack $I^{[t]}$. For each type t, the first $m^{[t]} - 1$ bins have $\operatorname{vol} \cdot f_k$ at least 1, so $\operatorname{vol}(f_k(I^{[t]})) > m^{[t]} - 1$. Therefore, total number of bins used is $\sum_{t=1}^{Q} m^{[t]} < \sum_{t=1}^{Q} (1 + \operatorname{vol}(f_k(I^{[t]}))) = Q + \operatorname{vol}(f_k(I))$. ◀

By Theorems 4 and 6, $\texttt{fullh}_k(I)$ uses less than $Q + T_k^d \operatorname{opt}_{d\text{BP}}(I)$ bins.

▶ **Theorem 7.** *Let* $\mathcal{I}$ *be a dMCBP instance. Let* $\widehat{K} := \{\operatorname{argmin}_{i \in I} \operatorname{vol}(f_k(i)) : I \in \mathcal{I}\}$*, i.e.,* $\widehat{K}$ *is the assortment obtained by picking from each itemset the item* i *having the minimum value of* $\operatorname{vol}(f_k(i))$*. Then the number of bins used by* $\texttt{fullh}_k(\widehat{K})$ *is less than* $Q + T_k^d \operatorname{opt}_{d\text{MCBP}}(\mathcal{I})$*, where* Q *is the number of distinct* types *of items in* $\operatorname{flat}(\mathcal{I})$ *(so* $Q \leq k^d$*).*

Proof. Let K^* be the assortment in an optimal packing of $\mathcal{I}$. So, $\operatorname{vol}(f_k(\widehat{K})) \leq \operatorname{vol}(f_k(K^*))$. By Theorems 4 and 6, the number of bins used by $\texttt{fullh}_k(\widehat{K})$ is less than

$$Q + \operatorname{vol}(f_k(\widehat{K})) \leq Q + \operatorname{vol}(f_k(K^*)) \leq Q + T_k^d \operatorname{opt}_{d\text{BP}}(K^*) = Q + T_k^d \operatorname{opt}_{d\text{MCBP}}(\mathcal{I}). \quad ◀$$

We can compute $\widehat{K}$ in $O(Nd)$ time and $\texttt{fullh}_k(\widehat{K})$ in $O(nd \log n)$ time, where $N := |\operatorname{flat}(\mathcal{I})|$, $n := |\mathcal{I}|$. So, we get an $O(Nd + nd \log n)$-time dMCBP algorithm having AAR T_k^d.

4.1 dBP with Rotations

As mentioned before, we can solve the rotational version of dBP by reducing it to dMCBP. Specifically, for each item i in the dBP instance, we create an itemset containing all orientations of i, and we pack the resulting dMCBP instance using $\texttt{fullh}_k$. Since an item can have up to $d!$ allowed orientations, this can take up to $O(nd! + nd \log n)$ time. Hence, the running time is large when d is large. However, we can do better for some special cases.

When the bin has the same length in each dimension, then for any item i, $\operatorname{vol}(f_k(i))$ is independent of how we orient i. Hence, we can orient the items I arbitrarily and then pack them using $\texttt{fullh}_k$ in $O(nd\log n)$ time.

Suppose there are no orientation constraints, i.e., all $d!$ orientations of each item are allowed. Let L_j be the length of the bin in the j^{th} dimension, for each $j \in [d]$. To use $\texttt{fullh}_k$ to pack I, we need to find the best orientation for each item $i \in I$, i.e., we need to find a permutation π for each item i such that $\prod_{j=1}^{d} f_k\left(\ell_{\pi_j}(i)/L_j\right)$ is minimized. This can be formulated as a maximum-weight bipartite matching problem on a graph with d vertices in each partition: for every $u \in [d]$ and $v \in [d]$, the edge (u, v) has a non-negative weight of $-\log(f_k(\ell_u(i)/L_v))$. So, using the Kuhn-Munkres algorithm [26], we can find the best orientation for each item in $O(d^3)$ time. Hence, we can pack I using $\texttt{fullh}_k$ in $O(nd^3 + nd\log n)$ time.

5 Better Algorithm for dMCBP ($\texttt{HGaP}_k$)

Here we describe a $T_k^{d-1}(1+\varepsilon)$-asymptotic-approximate algorithm for dMCBP based on $\texttt{HDH}_k$ and Lueker and Fernandez de la Vega's APTAS for 1BP [15]. We call our algorithm *Harmonic Guess-and-Pack* ($\texttt{HGaP}_k$). This improves upon $\texttt{fullh}_k$ that has AAR T_k^d.

▶ **Definition 8.** *For a dD item i, let $h(i) := \ell_d(i)$, $w(i) := \prod_{j=1}^{d-1} f_k(\ell_j(i))$ and $a(i) := w(i)h(i)$. Let $\texttt{round}(i)$ be a rectangle of height $h(i)$ and width $w(i)$. For a set X of dD items, define $w(X) := \sum_{i\in X} w(i)$ and $\texttt{round}(X) := \{\texttt{round}(i) : i \in X\}$.*

For any $\varepsilon > 0$, the algorithm $\texttt{HGaP}_k(\mathcal{I}, \varepsilon)$ returns a bin packing of $\mathcal{I}$, where $\mathcal{I}$ is a set of dD itemsets. $\texttt{HGaP}_k$ first converts $\mathcal{I}$ to a set $\widehat{\mathcal{I}}$ of 2D itemsets. It then computes P_{best}, which is a *structured* bin packing of $\widehat{\mathcal{I}}$ (we formally define *structured* later). Finally, it uses the algorithm $\texttt{inflate}$ to convert P_{best} into a bin packing of the dD itemsets $\mathcal{I}$, where $|\texttt{inflate}(P_{\text{best}})|$ is very close to $|P_{\text{best}}|$. See Algorithm 2 for a more precise description. We show that $|P_{\text{best}}| \lessapprox T_k^{d-1}(1+\varepsilon)\operatorname{opt}(\mathcal{I})$, which proves that $\texttt{HGaP}_k$ has an AAR of $T_k^{d-1}(1+\varepsilon)$. This approach of converting items to 2D, packing them, and then converting back to dD is very useful, because most of our analysis is about how to compute a structured 2D packing, and a packing of 2D items is easier to visualize and reason about than a packing of dD items.

Algorithm 2 $\texttt{HGaP}_k(\mathcal{I}, \varepsilon)$: Returns a bin packing of dD itemsets $\mathcal{I}$, where $\varepsilon \in (0, 1)$.

```
Let $\delta := \varepsilon/(2+\varepsilon)$.
$\widehat{\mathcal{I}} = \{\texttt{round}(I) : I \in \mathcal{I}\}$
Initialize $P_{\text{best}}$ to null.
for $P \in \texttt{guessShelves}(\widehat{\mathcal{I}}, \delta)$ do
    $\overline{P} = \texttt{chooseAndPack}(\widehat{\mathcal{I}}, P, \delta)$
    if $\overline{P}$ is not null and ($P_{\text{best}}$ is null or $|\overline{P}| \le |P_{\text{best}}|$) then
        $P_{\text{best}} = \overline{P}$
    end if
end for
return $\texttt{inflate}(P_{\text{best}})$
```

A 2D bin packing is called *shelf-based* if items are packed into *shelves* and the shelves are packed into bins, where a shelf is a rectangle of width 1. See Figure 2 for an example. A structured bin packing is a shelf-based bin packing where the heights of the shelves satisfy some additional properties (we describe these properties later). The algorithm $\texttt{guessShelves}$

repeatedly guesses the number and heights of shelves and computes a structured packing P of those shelves into bins. Then for each packing P, the algorithm $\texttt{chooseAndPack}(\widehat{\mathcal{I}}, P, \delta)$ *tries to* pack an assortment of $\widehat{\mathcal{I}}$ into the shelves in P plus one additional shelf. If chooseAndPack succeeds, call the resulting bin packing $\overline{P}$; else, chooseAndPack returns null. P_{best} is the value of $\overline{P}$ with the minimum number of bins across all guesses by guessShelves.

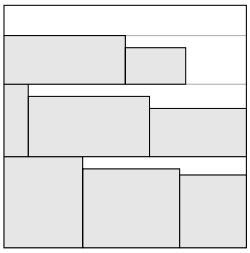


Figure 2 An example of shelf-based packing with 3 shelves.

To prove that $\texttt{HGaP}_k$ has AAR $T_k^{d-1}(1+\varepsilon)$, we show that for some $P^* \in \texttt{guessShelves}(\widehat{\mathcal{I}}, \delta)$, we have $|P^*| \lessapprox T_k^{d-1}(1+\varepsilon)\operatorname{opt}(\mathcal{I})$ and $\texttt{chooseAndPack}(\widehat{\mathcal{I}}, P^*, \delta)$ is not null.

We will now precisely define *structured* packing and state the main theorems on $\texttt{HGaP}_k$.

5.1 Structured Packing

▶ **Definition 9** (Slicing). *Slicing a 1D item i is the operation of replacing it by items i_1 and i_2 such that* $\operatorname{size}(i_1)+\operatorname{size}(i_2) = \operatorname{size}(i)$. *Slicing a rectangle i using a vertical cut is the operation of replacing i by two rectangles i_1 and i_2 where $h(i) = h(i_1) = h(i_2)$ and $w(i) = w(i_1)+w(i_2)$. Slicing i using a horizontal cut is the operation of replacing i by two rectangles i_1 and i_2 where $w(i) = w(i_1) = w(i_2)$ and $h(i) = h(i_1) + h(i_2)$.*

▶ **Definition 10** (Shelf-based δ-fractional packing). *Let $\delta \in (0, 1)$ be a constant. Let K be a set of rectangular items. Items in $K_L := \{i \in K : h(i) > \delta\}$ are called "δ-large" and items in $K_S := K - K_L$ are called "δ-small". A δ-fractional bin packing of K is defined to be a packing of K into bins where items in K_L can be sliced (recursively) using vertical cuts only, and items in K_S can be sliced (recursively) using both horizontal and vertical cuts.*

A shelf *is a rectangle of width 1 into which we can pack items such that the bottom edge of each item in the shelf touches the bottom edge of the shelf. A shelf can itself be packed into a bin. A δ-fractional bin packing of K is called* shelf-based *iff (all slices of) all items in K_L are packed into shelves, the shelves are packed into the bins, and items in K_S are packed outside the shelves (and inside the bins). Packing of items into a shelf S is called* tight *iff the top edge of some item (or slice) in S touches the top edge of S.*

▶ **Definition 11** (Structured packing). *Let K be a set of rectangles and let P be a packing of empty shelves into bins. Let H be the set of heights of shelves in P (note that H is not a multiset, i.e., we only consider distinct heights of shelves). Then P is called* structured *for (K, δ) iff $|H| \le \lceil 1/\delta^2 \rceil$ and each element in H is the height of some δ-large item in K.*

A shelf-based δ-fractional packing of K is called structured *iff the shelves in the packing are structured for (K, δ). Define $\operatorname{sopt}_\delta(K)$ to be the number of bins in the optimal structured δ-fractional packing of K.*

$\texttt{HGaP}_k$ relies on the following key structural theorem. We formally prove it in Section 5.5 and give an outline of the proof here.

▶ **Theorem 12** (Structural theorem). *Let I be a set of dD items. Let $\delta \in (0,1)$ be a constant. Then* $\operatorname{sopt}_\delta(\texttt{round}(I)) < T_k^{d-1}(1+\delta)\operatorname{opt}_{d\mathrm{BP}}(I) + \lceil 1/\delta^2 \rceil + 1 + \delta$.

Proof outline. Let $\widehat{I} := \texttt{round}(I)$. Let $\widehat{I}_L$ and $\widehat{I}_S$ be the δ-large and δ-small items in $\widehat{I}$, respectively. We give a simple greedy algorithm to pack $\widehat{I}_L$ into shelves. Let J be the shelves output by this algorithm. We can treat J as a 1BP instance, and $\widehat{I}_S$ as a sliceable 1D item of size $a(\widehat{I}_S)$. We prove that an optimal 1D bin packing of $J \cup \widehat{I}_S$ gives us an optimal shelf-based δ-fractional packing of $\widehat{I}$.

We use linear grouping by Lueker and Fernandez de la Vega [15]. We partition J into linear groups of size $\lfloor \delta \operatorname{size}(J) \rfloor + 1$ each. Let h_j be the height of the first 1D item in the j^{th} group. Let $J^{(\text{hi})}$ be the 1BP instance obtained by rounding up the height of each item in the j^{th} group to h_j for all j. Then $J^{(\text{hi})}$ contains at most $\lceil 1/\delta^2 \rceil$ distinct sizes, so the optimal packing of $J^{(\text{hi})} \cup \widehat{I}_S$ gives us a structured δ-fractional packing of $\widehat{I}$. Therefore, $\operatorname{sopt}_\delta(\widehat{I}) \le \operatorname{opt}(J^{(\text{hi})} \cup \widehat{I}_S)$. Let $J^{(\text{lo})}$ be the 1BP instance obtained by rounding down the height of each item in the j^{th} group to h_{j+1} for all j. We prove that $J^{(\text{lo})}$ contains at most $\lceil 1/\delta^2 \rceil - 1$ distinct sizes and that $\operatorname{opt}(J^{(\text{hi})} \cup \widehat{I}_S) < \operatorname{opt}(J^{(\text{lo})} \cup \widehat{I}_S) + \delta a(\widehat{I}_L) + (1+\delta)$.

We model packing $J^{(\text{lo})} \cup \widehat{I}_S$ as a linear program, denoted by $\operatorname{LP}(\widehat{I})$, that has at most $\lceil 1/\delta^2 \rceil^{1/\delta}$ variables and $\lceil 1/\delta^2 \rceil$ non-trivial constraints. The optimum extreme point solution to $\operatorname{LP}(\widehat{I})$, therefore, has at most $\lceil 1/\delta^2 \rceil$ positive entries, so $\operatorname{opt}(J^{(\text{lo})} \cup \widehat{I}_S) \le \operatorname{opt}(\operatorname{LP}(\widehat{I})) + \lceil 1/\delta^2 \rceil$.

We use techniques from Caprara [8] to obtain a monotonic weighting function η from the optimal solution to the dual of $\operatorname{LP}(\widehat{I})$. For each item $i \in I$, we define $p(i) := w(i)\eta(h(i))$ and prove that $p(I) \ge \operatorname{opt}(\operatorname{LP}(\widehat{I}))$. By Theorem 3, we get that $p(I) \le T_k^{d-1}\operatorname{opt}_{d\mathrm{BP}}(I)$ and $a(\widehat{I}_L) \le T_k^{d-1}\operatorname{opt}_{d\mathrm{BP}}(I)$. Combining the above facts gives us an upper-bound on $\operatorname{sopt}_\delta(\widehat{I})$ in terms of $\operatorname{opt}_{d\mathrm{BP}}(I)$. ◀

5.2 Subroutines

5.2.1 `guessShelves`

The algorithm $\texttt{guessShelves}(\widehat{\mathcal{I}}, \delta)$ takes a set $\widehat{\mathcal{I}}$ of 2D itemsets and a constant $\delta \in (0,1)$ as input. We will design `guessShelves` so that it satisfies the following theorem.

▶ **Theorem 13.** $\texttt{guessShelves}(\widehat{\mathcal{I}}, \delta)$ *returns all possible packings of empty shelves into at most* $|\widehat{\mathcal{I}}|$ *bins such that each packing is structured for* $(\operatorname{flat}(\widehat{\mathcal{I}}), \delta)$. $\texttt{guessShelves}(\widehat{\mathcal{I}}, \delta)$ *returns at most* $T := (N^{\lceil 1/\delta^2 \rceil} + 1)(n+1)^R$ *packings, where* $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$, $n := |\widehat{\mathcal{I}}|$, *and* $R := \binom{\lceil 1/\delta^2 \rceil + \lceil 1/\delta \rceil - 1}{\lceil 1/\delta \rceil - 1} \le (1 + \lceil 1/\delta^2 \rceil)^{1/\delta}$. *Its running time is* $O(T)$.

`guessShelves` works by first guessing at most $\lceil 1/\delta^2 \rceil$ distinct heights of shelves. It then enumerates all configurations, i.e., different ways in which shelves can be packed into a bin. It then guesses the configurations in a bin packing of the shelves. `guessShelves` can be easily implemented using standard techniques. For the sake of completeness, we give a more precise description of `guessShelves` and prove Theorem 13 in Appendix A.2.

5.2.2 `chooseAndPack`

$\texttt{chooseAndPack}(\widehat{\mathcal{I}}, P, \delta)$ takes as input a set $\widehat{\mathcal{I}}$ of 2D itemsets, a constant $\delta \in (0,1)$, and a bin packing P of empty shelves that is structured for $(\operatorname{flat}(\widehat{\mathcal{I}}), \delta)$. It tries to pack an assortment of $\widehat{\mathcal{I}}$ into the shelves in P.

`chooseAndPack` works by rounding up the width of all δ-large items in $\widehat{\mathcal{I}}$ to a multiple of $1/n$. This would increase the number of shelves required by 1, so it adds another empty shelf. It then uses dynamic programming to pack an assortment into the shelves, such that the area of the chosen δ-small items is minimum. This is done by maintaining a dynamic programming table that keeps track of the number of itemsets considered so far and the remaining space in shelves of each type. If it is not possible to pack the items into the shelves, then `chooseAndPack` outputs `null`. In Appendix A.3, we give the details of this algorithm and formally prove the following theorems:

▶ **Theorem 14.** *If there exists an assortment $\widehat{K}$ of $\widehat{\mathcal{I}}$ having a structured δ-fractional bin packing P, then* `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ *does not output* `null`*.*

▶ **Theorem 15.** *If the output of* `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ *is not* `null`*, then the output $\overline{P}$ is a shelf-based δ-fractional packing of some assortment of $\widehat{\mathcal{I}}$ such that $|\overline{P}| \le |P| + 1$ and the distinct shelf heights in $\overline{P}$ are the same as that in P.*

▶ **Theorem 16.** `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ *runs in $O(Nn^{2\lceil 1/\delta^2 \rceil})$ time. Here $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$, $n := |\widehat{\mathcal{I}}|$.*

5.2.3 `inflate`

For a set I of dD items, `inflate` is an algorithm that converts a shelf-based packing of `round`(I) into a packing of I having roughly the same number of bins.

For a dD item i, $\operatorname{btype}(i)$ (called *base type*) is defined to be a $(d-1)$-dimensional vector whose j^{th} component is $\operatorname{type}_k(\ell_j(i))$. Roughly, `inflate`$(P)$ works as follows: It first slightly modifies the packing P so that items of different base types are in different shelves and δ-small items are no longer sliced using horizontal cuts. Then it converts each 2D shelf to a dD shelf of the same height using `HDH-unit-pack`$_k$ (a dD shelf is a cuboid where the first $d-1$ dimensions are equal to 1).

In Appendix A.4, we formally describe `inflate` and prove the following theorem.

▶ **Theorem 17.** *Let I be a set of dD items having Q distinct base types (there can be at most k^{d-1} distinct base types, so $Q \le k^{d-1}$). Let P be a shelf-based δ-fractional packing of* `round`(I) *where shelves have t distinct heights. Then* `inflate`(P) *returns a packing of I into less than $|P|/(1-\delta) + t(Q-1) + 1 + \delta Q/(1-\delta)$ bins in $O(|I| d \log |I|)$ time.*

Now that we have mentioned the guarantees of all the subroutines used by `HGaP`$_k$, we can prove the correctness and running time of `HGaP`$_k$.

5.3 Correctness and Running Time of `HGaP`$_k$

▶ **Theorem 18.** *The number of bins used by* `HGaP`$_k(\mathcal{I}, \varepsilon)$ *to pack $\mathcal{I}$ is less than*

$$T_k^{d-1}(1+\varepsilon)\operatorname{opt}_{d\mathrm{MCBP}}(\mathcal{I}) + \left\lceil \left(\frac{2}{\varepsilon} + 1\right)^2 \right\rceil \left(Q + \frac{\varepsilon}{2}\right) + 3 + (Q+3)\frac{\varepsilon}{2}.$$

Here $Q \le k^{d-1}$ is the number of distinct base types in $\operatorname{flat}(\mathcal{I})$*.*

Proof. Let K^* be the assortment in an optimal bin packing of $\mathcal{I}$. Let $\widehat{K}^* =$ `round`(K^*). Let P^* be the optimal structured δ-fractional bin packing of $\widehat{K}^*$. Then $|P^*| = \operatorname{sopt}_\delta(\widehat{K}^*)$ by the definition of sopt. By Theorem 13, $P^* \in$ `guessShelves`$(\widehat{\mathcal{I}}, \delta)$. Let $\overline{P}^* =$ `chooseAndPack`$(\widehat{\mathcal{I}}, P^*, \delta)$. By Theorem 14, $\overline{P}^*$ is not `null`. By Theorem 15, P_{best} is structured for $(\operatorname{flat}(\widehat{\mathcal{I}}), \delta)$ and $|P_{\text{best}}| \le |\overline{P}^*| \le \operatorname{sopt}_\delta(\widehat{K}^*) + 1$.

By Theorem 17, we get that

$$|\,\mathtt{inflate}(P_{\text{best}})| < \frac{\operatorname{sopt}_\delta(\widehat{K}^*)}{1-\delta} + \left\lceil \frac{1}{\delta^2} \right\rceil (Q-1) + 1 + \frac{\delta Q + 1}{1-\delta}.$$

By Theorem 12 (structural theorem) and using $\operatorname{opt}_{d\mathrm{BP}}(K^*) = \operatorname{opt}_{d\mathrm{MCBP}}(\mathcal{I})$, we get

$$\operatorname{sopt}_\delta(\widehat{K}^*) < T_k^{d-1}(1+\delta)\operatorname{opt}_{d\mathrm{MCBP}}(\mathcal{I}) + \lceil 1/\delta^2 \rceil + 1 + \delta.$$

Therefore, $|\,\mathtt{inflate}(P_{\text{best}})|$ is less than

$$T_k^{d-1}\frac{1+\delta}{1-\delta}\operatorname{opt}_{d\mathrm{MCBP}}(\mathcal{I}) + \left\lceil \frac{1}{\delta^2} \right\rceil \left(Q + \frac{\delta}{1-\delta}\right) + 3 + \frac{\delta(3+Q)}{1-\delta}$$
$$= T_k^{d-1}(1+\varepsilon)\operatorname{opt}_{d\mathrm{MCBP}}(\mathcal{I}) + \left\lceil \left(\frac{2}{\varepsilon}+1\right)^2 \right\rceil \left(Q + \frac{\varepsilon}{2}\right) + 3 + (Q+3)\frac{\varepsilon}{2}.$$ ◀

▶ **Theorem 19.** *$\mathtt{HGaP}_k(\mathcal{I}, \varepsilon)$ runs in time $O(N^{1+\lceil 1/\delta^2 \rceil} n^{R+2\lceil 1/\delta^2 \rceil} + Nd + nd\log n)$, where $n := |\widehat{\mathcal{I}}|$, $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$, $\delta := \varepsilon/(2+\varepsilon)$ and $R := \binom{\lceil 1/\delta^2 \rceil + \lceil 1/\delta \rceil - 1}{\lceil 1/\delta \rceil - 1} \le (1 + \lceil 1/\delta^2 \rceil)^{1/\delta}$.*

Proof. Follows from Theorems 13, 16, and 17. ◀

Appendix A.5 gives hints on improving the running time of $\mathtt{HGaP}_k$.

5.4 *d*BP with Rotations

We can solve the rotational version of dBP by reducing it to dMCBP and using the $\mathtt{HGaP}_k$ algorithm. Since each item can have up to $d!$ orientations, the running time is polynomial in $nd!$, which is large when d is large. But we can do better for some special cases.

When the bin has the same length in each dimension, then for any item i, $w(i) := \prod_{j=1}^{d-1} f_k(\ell_j(i))$ is invariant to permuting the first $d-1$ dimensions. In the first step of $\mathtt{HGaP}_k$, we replace each dD item i by a rectangle of width $w(i)$ and height $\ell_d(i)$. So, instead of considering all $d!$ orientations, we just need to consider at most d different orientations, where each orientation has a different length in the d^{th} dimension.

Suppose there are no orientation constraints, i.e., all $d!$ orientations of each item are allowed. Let L_j be the length of the bin in the j^{th} dimension, for each $j \in [d]$. Analogous to the trick in Section 4.1, we first fix the d^{th} dimension of the item and then optimally permute the first $d-1$ dimensions using a max-weight bipartite matching algorithm. Hence, we need to consider only d orientations instead of $d!$.

5.5 Proof of the Structural Theorem

In this section, we give a formal proof of the Structural Theorem (Theorem 12).

5.5.1 Predecessors and Canonical Shelving

▶ **Definition 20.** *Let I_1 and I_2 be sets of 1D items. I_1 is called a predecessor of I_2 ($I_1 \preceq I_2$) iff there exists a one-to-one mapping $\pi : I_1 \mapsto I_2$ such that $\forall i \in I_1, \operatorname{size}(i) \le \operatorname{size}(\pi(i))$.*

▶ **Observation 21.** *Let $I_1 \preceq I_2$ and π be the corresponding mapping. We can get a packing of I_1 from a packing of I_2, by packing each $i \in I_1$ in the place of $\pi(i)$. Hence, $\operatorname{opt}(I_1) \le \operatorname{opt}(I_2)$.*

▶ **Definition 22** (Canonical shelving). *Let I be a set of rectangles. Order the items in I in non-increasing order of height (break ties arbitrarily but deterministically) and greedily pack them into tight shelves, slicing items using vertical cuts if necessary. The set of shelves thus obtained is called the* canonical shelving *of I, and is denoted by* $\operatorname{canShelv}(I)$. *(The canonical shelving is unique because ties are broken deterministically.)*

See Figure 3 for an example of canonical shelving.

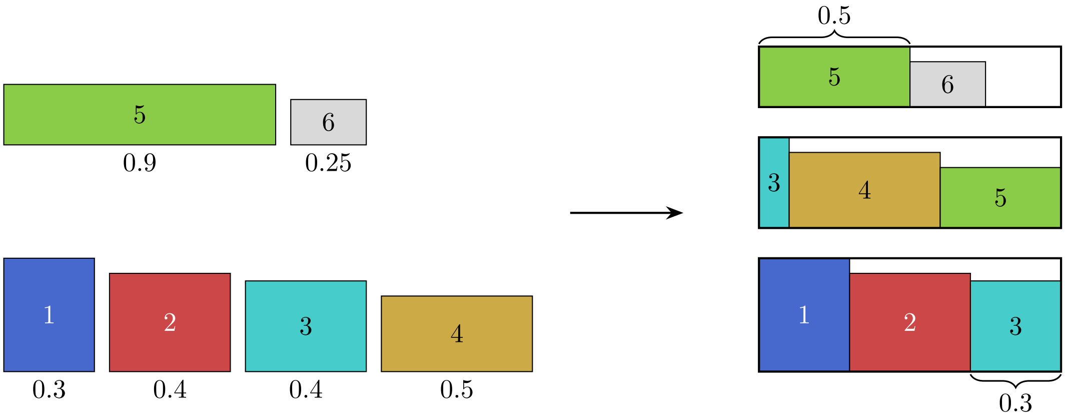


Figure 3 Six items and their canonical shelving into three tight shelves of width 1. The items are numbered by decreasing order of height. Each item has its width mentioned below it. Item 3 was sliced into two items of widths 0.3 and 0.1. Item 5 was sliced into two items of widths 0.4 and 0.5.

Suppose a set I of rectangular items is packed into a set J of shelves. Then we can interpret J as a 1BP instance where the height of each shelf is the size of the corresponding 1D item. We will now prove that the canonical shelving is optimal, i.e., any shelf-based bin packing of items can be obtained by first computing the canonical shelving and then packing the shelves into bins like a 1BP instance.

▶ **Lemma 23.** *If $J^* := \operatorname{canShelv}(I)$ and I can be packed into shelves J, then $J^* \preceq J$.*

Proof. We say that a shelf is full if the total width of items in a shelf is 1. Arrange the shelves J in non-increasing order of height, and arrange the items I in non-increasing order of height. Then try to pack I into J using the following greedy algorithm: For each item i, pack the largest possible slice of i into the first non-full shelf and pack the remaining slice (if any) in the next shelf. If this greedy algorithm succeeds, then within each shelf of J, there is a shelf of J^*, so $J^* \preceq J$. We will now prove that this greedy algorithm always succeeds.

For the sake of proof by contradiction, assume that the greedy algorithm failed, i.e., for an item (or slice) i there was a non-full shelf S but $h(i) > h(S)$. Let I' be the items (and slices) packed before i and J' be the shelves before S. Therefore, $w(I') = |J'|$.

Items in I' have height at least $h(i)$, so shelves in J' have height at least $h(i)$. Shelves after J' have height less than $h(i)$. So, J' is exactly the set of shelves of height at least $h(i)$. In the packing P, $I' \cup \{i\}$ can only be packed into shelves of height at least $h(i)$, so $w(I') + w(i) \le |J'|$. This contradicts $w(I') = |J'|$. So, the greedy algorithm cannot fail. ◀

5.5.2 Linear Grouping

Let I be a set of dD items. Let $\widehat{I} := \texttt{round}(I)$. Let $\delta \in (0,1)$ be a constant. Let $\widehat{I}_L := \{i \in \widehat{I} : h(i) > \delta\}$ and $\widehat{I}_S := \widehat{I} - \widehat{I}_L$. Let $J := \operatorname{canShelv}(\widehat{I}_L)$. Let $m := |J|$, i.e., J contains m shelves. We can interpret $\widehat{I}_S$ as a single sliceable 1D item of size $a(\widehat{I}_S)$.

To prove Theorem 12, we will show the existence of a structured δ-fractional packing of $\widehat{I}$ into at most $T_k^{d-1}(1+\delta)\operatorname{opt}_{d\mathrm{BP}}(I) + \lceil 1/\delta^2 \rceil + 1 + \delta$ bins.

▶ **Definition 24** (Linear grouping [15]). *Arrange the 1D items J in non-increasing order of size and number them from 1 to m. Let $q := \lfloor \delta \operatorname{size}(J) \rfloor + 1$. Let J_1 be the first q items, J_2 be the next q items, and so on. J_j is called the j^{th}* linear group *of J. This gives us $t := \lceil m/q \rceil$ linear groups. Note that the last group, J_t, may have less than q items.*

Let h_j be the size of the first item in J_j. Let $h_{t+1} := 0$. For $j \in [t-1]$, let $J_j^{(\mathrm{lo})}$ be the items obtained by decreasing the height of items in J_j to h_{j+1}. For $j \in [t]$, let $J_j^{(\mathrm{hi})}$ be the items obtained by increasing the height of items in J_j to h_j.

Let $J^{(\mathrm{lo})} := \bigcup_{j=1}^{t-1} J_j^{(\mathrm{lo})}$ and $J^{(\mathrm{hi})} := \bigcup_{j=1}^{t} J_j^{(\mathrm{hi})}$. We call $J^{(\mathrm{lo})}$ a down-rounding of J and $J^{(\mathrm{hi})}$ an up-rounding of J.

▶ **Lemma 25.** $t \le \lceil 1/\delta^2 \rceil$.

Proof. Since each shelf in J has height more than δ, $\operatorname{size}(J) > |J|\delta$.

$$t := \left\lceil \frac{|J|}{\lfloor \delta \operatorname{size}(J) \rfloor + 1} \right\rceil \le \left\lceil \frac{\operatorname{size}(J)/\delta}{\delta \operatorname{size}(J)} \right\rceil = \left\lceil \frac{1}{\delta^2} \right\rceil. \qquad ◀$$

▶ **Lemma 26.** $J^{(\mathrm{lo})} \preceq J \preceq J^{(\mathrm{hi})} \preceq J^{(\mathrm{lo})} \cup J_1^{(\mathrm{hi})}$.

Proof. It is trivial to see that $J^{(\mathrm{lo})} \preceq J \preceq J^{(\mathrm{hi})}$. For $j \in [t-1]$, all (1D) items in both $J_j^{(\mathrm{lo})}$ and $J_{j+1}^{(\mathrm{hi})}$ have height h_{j+1}, and $|J_{j+1}| \le q = |J_j|$. Therefore, $J_{j+1}^{(\mathrm{hi})} \preceq J_j^{(\mathrm{lo})}$ and hence

$$J^{(\mathrm{hi})} = J_1^{(\mathrm{hi})} \cup \bigcup_{j=1}^{t-1} J_{j+1}^{(\mathrm{hi})} \preceq J_1^{(\mathrm{hi})} \cup \bigcup_{j=1}^{t-1} J_j^{(\mathrm{lo})} = J_1^{(\mathrm{hi})} \cup J^{(\mathrm{lo})}. \qquad ◀$$

▶ **Lemma 27.** $\operatorname{size}(J) < 1 + a(\widehat{I}_L)$.

Proof. In the canonical shelving of $\widehat{I}_L$, let S_j be the j^{th} shelf. Let $h(S_j)$ be the height of S_j. Let $a(S_j)$ be the total area of the items in S_j. Since the shelves are tight, items in S_j have height at least $h(S_{j+1})$. So, $a(S_j) \ge h(S_{j+1})$ and

$$\operatorname{size}(J) = \sum_{j=1}^{|J|} h(S_j) \le 1 + \sum_{j=1}^{|J|-1} h(S_{j+1}) \le 1 + \sum_{j=1}^{|J|-1} a(S_j) < 1 + a(\widehat{I}_L). \qquad ◀$$

▶ **Lemma 28.** $\operatorname{sopt}_\delta(\widehat{I}) < \operatorname{opt}(J^{(\mathrm{lo})} \cup \widehat{I}_S) + \delta a(\widehat{I}_L) + (1+\delta)$.

Proof. By the definition of canShelv, $\widehat{I}_L$ can be packed into J. By Lemma 26, $J \preceq J^{(\mathrm{hi})}$, so $\widehat{I}_L$ can be packed into $J^{(\mathrm{hi})}$. By Lemma 25, the number of distinct sizes in $J^{(\mathrm{hi})}$ is at most $\lceil 1/\delta^2 \rceil$. So, the optimal 1D bin packing of $J^{(\mathrm{hi})} \cup \widehat{I}_S$ will give us a structured δ-fractional bin packing of $\widehat{I}$. Hence, $\operatorname{sopt}_\delta(\widehat{I}) \le \operatorname{opt}(J^{(\mathrm{hi})} \cup \widehat{I}_S)$. By Lemma 26 and Observation 21 we get

$$\operatorname{opt}(J^{(\mathrm{hi})} \cup \widehat{I}_S) \le \operatorname{opt}(J^{(\mathrm{lo})} \cup J_1^{(\mathrm{hi})} \cup \widehat{I}_S) \le \operatorname{opt}(J^{(\mathrm{lo})} \cup \widehat{I}_S) + \operatorname{opt}(J_1^{(\mathrm{hi})}).$$

By Lemma 27,

$$\operatorname{opt}(J_1^{(\mathrm{hi})}) \le |J_1^{(\mathrm{hi})}| \le q \le 1 + \delta \operatorname{size}(J) < 1 + \delta(1 + a(\widehat{I}_L)). \qquad ◀$$

5.5.3 LP for Packing $J^{(\mathrm{lo})} \cup \widehat{I}_S$

We will formulate an integer linear program for bin packing $J^{(\mathrm{lo})} \cup \widehat{I}_S$.

Let $C \in \mathbb{Z}_{\geq 0}^{t-1}$ such that $h_C := \sum_{j=1}^{t-1} C_j h_{j+1} \leq 1$. Then C is called a configuration. C represents a set of 1D items that can be packed into a bin and where C_j items are from $J_j^{(\mathrm{lo})}$. Let $\mathcal{C}$ be the set of all configurations. We can pack at most $\lceil 1/\delta \rceil - 1$ 1D items into a bin because $h_t > \delta$. By Lemma 1, we get $|\mathcal{C}| \leq \binom{\lceil 1/\delta \rceil - 1 + t - 1}{t-1} \leq \lceil 1/\delta^2 \rceil^{1/\delta}$.

Let x_C be the number of bins packed according to configuration C. Bin packing $J^{(\mathrm{lo})} \cup \widehat{I}_S$ is equivalent to finding the optimal integer solution to the following linear program, which we denote as $\mathrm{LP}(\widehat{I})$.

$$\begin{aligned} \min_{x \in \mathbb{R}^{|\mathcal{C}|}} \quad & \sum_{C \in \mathcal{C}} x_C \\ \text{where} \quad & \sum_{C \in \mathcal{C}} C_j x_C \geq q & \forall j \in [t-1] \\ & \sum_{C \in \mathcal{C}} (1 - h_C) x_C \geq a(\widehat{I}_S) \\ & x_C \geq 0 & \forall C \in \mathcal{C} \end{aligned}$$

Here the first set of constraints say that for each $j \in [t-1]$, all of the $q := \lfloor \delta \operatorname{size}(J) \rfloor + 1$ shelves $J_j^{(\mathrm{lo})}$ should be covered by the configurations in x. The second constraint says that we should be able to pack $a(\widehat{I}_S)$ into the non-shelf space in the bins.

▶ **Lemma 29.** $\operatorname{opt}(J^{(\mathrm{lo})} \cup \widehat{I}_S) \leq \operatorname{opt}(\mathrm{LP}(\widehat{I})) + t$.

Proof. Let x^* be an optimal extreme-point solution to $\mathrm{LP}(\widehat{I})$. Then x^* has at most t non-zero entries. Let $\widehat{x}$ be a vector where $\widehat{x}_C := \lceil x_C^* \rceil$. Then $\widehat{x}$ is an integral solution to $\mathrm{LP}(\widehat{I})$ and $\sum_C \widehat{x}_C < t + \sum_C x_C^* = \operatorname{opt}(\mathrm{LP}(\widehat{I})) + t$. ◀

The dual of $\mathrm{LP}(\widehat{I})$, denoted by $\mathrm{DLP}(\widehat{I})$, is

$$\begin{aligned} \max_{y \in \mathbb{R}^{t-1}, z \in \mathbb{R}} \quad & a(\widehat{I}_S) z + q \sum_{j=1}^{t-1} y_j \\ \text{where} \quad & \sum_{j=1}^{t-1} C_j y_j + (1 - h_C) z \leq 1 \quad \forall C \in \mathcal{C} \\ & z \geq 0 \text{ and } y_j \geq 0 \quad \forall j \in [t-1] \end{aligned}$$

We will now see how to obtain a monotonic weighting function $\eta : [0,1] \mapsto [0,1]$ from a feasible solution to $\mathrm{DLP}(\widehat{I})$. To do this, we adapt techniques from Caprara's analysis of $\mathtt{HDH}_k$ [8]: we first describe a transformation to convert any feasible solution of $\mathrm{DLP}(\widehat{I})$ to a feasible solution that is *monotonic*, and then show how to obtain a weighting function from this monotonic solution. Such a weighting function will help us upper-bound $\operatorname{opt}(\mathrm{LP}(\widehat{I}))$ in terms of $\operatorname{opt}_{d\mathrm{BP}}(I)$.

▶ **Definition 30.** *Let (y, z) be a feasible solution to $\mathrm{DLP}(\widehat{I})$. Let $h_{t+1} := 0$ and for $j \in [t-1]$ let $\widehat{y}_j := \max(y_j, \widehat{y}_{j+1} + (h_{j+1} - h_{j+2})z)$. Then $(\widehat{y}, z)$ is called the monotonization of (y, z).*

▶ **Lemma 31.** *Let (y, z) be a feasible solution to $\mathrm{DLP}(\widehat{I})$. Let $(\widehat{y}, z)$ be the monotonization of (y, z). Then $(\widehat{y}, z)$ is a feasible solution to $\mathrm{DLP}(\widehat{I})$.*

Proof. (See Appendix A.1.) ◀

Let (y^*, z^*) be an optimal solution to $\operatorname{DLP}(\widehat{I})$. Let $(\widehat{y}, z^*)$ be the monotonization of (y^*, z^*). Then define the function $\eta : [0,1] \mapsto [0,1]$ as

$$\eta(x) := \begin{cases} \widehat{y}_1 & \text{if } x \in [h_2, 1] \\ \widehat{y}_j & \text{if } x \in [h_{j+1}, h_j),\ \text{for } 2 \le j \le t-1 \\ xz^* & \text{if } x < h_t \end{cases}.$$

▶ **Lemma 32.** *η is a monotonic weighting function.*

Proof. (See Appendix A.1.) ◀

▶ **Lemma 33.** *For $i \in I$, let $p(i) := \eta(h(i))w(i)$. Then* $\operatorname{opt}(\operatorname{LP}(\widehat{I})) \le p(I) \le T_k^{d-1} \operatorname{opt}_{d\mathrm{BP}}(I)$.

Proof. Let (y^*, z^*) be an optimal solution to $\operatorname{DLP}(\widehat{I})$. Let $(\widehat{y}, z^*)$ be its monotonization.

In the canonical shelving of I, suppose a rectangular item i (or a slice thereof) lies in shelf S where $S \in J_j$. Then $h(i) \in [h_{j+1}, h_j]$, where $h_{t+1} := 0$. This is because shelves in $J := \operatorname{canShelv}(\widehat{I})$ are tight. If $j = 1$, then $\eta(h(i)) = \widehat{y}_1 \ge y_1^*$. If $2 \le j \le t-1$, then $\eta(h(i)) \in \{\widehat{y}_{j-1}, \widehat{y}_j\} \ge \widehat{y}_j \ge y_j^*$. We know that $w(S) = 1$ for each shelf $S \in J_j$ for $j \in [t-1]$.

$$\begin{aligned} p(I) &= \sum_{j=1}^{t} \sum_{S \in J_j} \sum_{i \in S} \eta(h(i))w(i) + \sum_{i \in \widehat{I}_S} \eta(h(i))w(i) && \text{(by definition of } p\text{)} \\ &\ge \sum_{j=1}^{t-1} \sum_{S \in J_j} \sum_{i \in S} y_j^* w(i) + \sum_{i \in \widehat{I}_S} (h(i)z^*)w(i) && \text{(by definition of } \eta\text{)} \\ &= \sum_{j=1}^{t-1} y_j^* q + a(\widehat{I}_S)z^* && \text{(since } w(J_j) = q \text{ for } j \in [t-1]\text{)} \\ &= \operatorname{opt}(\operatorname{DLP}(\widehat{I})). && \text{((}y^*, z^*\text{) is optimal for } \operatorname{DLP}(\widehat{I})\text{)} \end{aligned}$$

By strong duality of linear programs, $\operatorname{opt}(\operatorname{LP}(\widehat{I})) = \operatorname{opt}(\operatorname{DLP}(\widehat{I})) \le p(I)$. Since η and H_k are weighting functions (by Lemma 32), we get that $p(I) \le T_k^{d-1} \operatorname{opt}_{d\mathrm{BP}}(I)$ by Theorem 3. ◀

▶ **Theorem 12** (Structural theorem). *Let I be a set of dD items. Let $\delta \in (0,1)$ be a constant. Then* $\operatorname{sopt}_\delta(\texttt{round}(I)) < T_k^{d-1}(1+\delta)\operatorname{opt}_{d\mathrm{BP}}(I) + \lceil 1/\delta^2 \rceil + 1 + \delta$.

Proof.

$$a(\widehat{I}_L) \le a(\widehat{I}) = \sum_{i \in I} \left(\ell_d(i) \prod_{j=1}^{d-1} f_k(\ell_j(i)) \right) \le T_k^{d-1} \operatorname{opt}_{d\mathrm{BP}}(I). \qquad \text{(by Theorem 3)}$$

$$\begin{aligned} \operatorname{sopt}_\delta(\widehat{I}) &< \operatorname{opt}(J^{(\mathrm{lo})} \cup \widehat{I}_S) + \delta a(\widehat{I}_L) + (1+\delta) && \text{(by Lemma 28)} \\ &\le \operatorname{opt}(\operatorname{LP}(\widehat{I})) + \left\lceil \frac{1}{\delta^2} \right\rceil + \delta T_k^{d-1} \operatorname{opt}_{d\mathrm{BP}}(I) + (1+\delta) && \text{(by Lemmas 25 and 29)} \\ &\le T_k^{d-1}(1+\delta)\operatorname{opt}_{d\mathrm{BP}}(I) + \left\lceil \frac{1}{\delta^2} \right\rceil + 1 + \delta. && \text{(by Lemma 33)} \end{aligned}$$

◀

References

1 Mauro Maria Baldi, Guido Perboli, and Roberto Tadei. The three-dimensional knapsack problem with balancing constraints. *Applied Mathematics and Computation*, 218(19):9802–9818, 2012. doi:10.1016/j.amc.2012.03.052.

2 János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new and improved algorithm for online bin packing. In *European Symposium on Algorithms (ESA)*, pages 5:1–5:14, 2018. doi:10.4230/LIPIcs.ESA.2018.5.

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, 2010. doi:10.1137/080736831.

4 Nikhil Bansal, Xin Han, Kazuo Iwama, Maxim Sviridenko, and Guochuan Zhang. A harmonic algorithm for the 3d strip packing problem. *SIAM Journal on Computing*, 42(2):579–592, 2013. doi:10.1137/070691607.

5 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 13–25, 2014. doi:10.1137/1.9781611973402.2.

6 Nikhil Bansal and Maxim Sviridenko. New approximability and inapproximability results for 2-dimensional bin packing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 196–203, 2004.

7 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.

8 Alberto Caprara. Packing *d*-dimensional bins in *d* stages. *Mathematics of Operations Research*, 33:203–215, February 2008. doi:10.1287/moor.1070.0289.

9 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.

10 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.

11 János Csirik and André van Vliet. An on-line algorithm for multidimensional bin packing. *Operations Research Letters*, 13(3):149–158, 1993. doi:10.1016/0167-6377(93)90004-Z.

12 Leah Epstein and Rob van Stee. Optimal online algorithms for multidimensional packing problems. *SIAM Journal on Computing*, 35(2):431–448, 2005. doi:10.1137/S0097539705446895.

13 Leah Epstein and Rob van Stee. This side up! *ACM Transactions on Algorithms (TALG)*, 2(2):228–243, 2006. doi:10.1145/1150334.1150339.

14 Sándor P. Fekete and Jörg Schepers. A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, 60(2):311–329, 2004. doi:10.1007/s001860400376.

15 Wenceslas Fernandez de la Vega and George S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981. doi:10.1007/BF02579456.

16 Paul C. Gilmore and Ralph E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961. doi:10.1287/opre.9.6.849.

17 Xin Han, Francis YL Chin, Hing-Fung Ting, Guochuan Zhang, and Yong Zhang. A new upper bound 2.5545 on 2D online bin packing. *ACM Transactions on Algorithms (TALG)*, 7(4):1–18, 2011. doi:10.1145/2000807.2000818.

18 Klaus Jansen. A $(3/2+\varepsilon)$ approximation algorithm for scheduling moldable and non-moldable parallel tasks. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 224–235, 2012. doi:10.1145/2312005.2312048.

19 Klaus Jansen and Lars Prädel. New approximability results for two-dimensional bin packing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 919–936, 2013. doi:10.1007/s00453-014-9943-z.

20 Klaus Jansen and Lars Prädel. A new asymptotic approximation algorithm for 3-dimensional strip packing. In *SOFSEM*, pages 327–338, 2014. doi:10.1007/978-3-319-04298-5_29.

21 Klaus Jansen and Rob van Stee. On strip packing with rotations. In *Symposium on Theory of Computing (STOC)*, pages 755–761. ACM, 2005. doi:10.1145/1060590.1060702.

22 Claire Kenyon and Eric Rémila. Approximate strip packing. In *Foundations of Computer Science (FOCS)*, pages 31–36, 1996. doi:10.1109/SFCS.1996.548461.

23 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.

24 C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *Journal of the ACM*, 32(3):562–572, July 1985. doi:10.1145/3828.3833.

25 Flavio Keidi Miyazawa and Yoshiko Wakabayashi. Three-dimensional packings with rotations. *Computers & Operations Research*, 36(10):2801–2815, 2009. doi:10.1016/j.cor.2008.12.015.

26 James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

27 Boaz Patt-Shamir and Dror Rawitz. Vector bin packing with multiple-choice. *Discrete Applied Mathematics*, 160(10-11):1591–1600, 2012. doi:10.1016/j.dam.2012.02.020.

28 Prakash Ramanan, Donna J Brown, Chung-Chieh Lee, and Der-Tsai Lee. On-line bin packing in linear time. *Journal of Algorithms*, 10(3):305–326, 1989. doi:10.1016/0196-6774(89)90031-X.

29 Steven S Seiden. On the online bin packing problem. *Journal of the ACM*, 49(5):640–671, 2002. doi:10.1145/585265.585269.

30 Eklavya Sharma. Harmonic algorithms for packing d-dimensional cuboids into bins. *ArXiv*, 2011.10963, 2020. arXiv:2011.10963.

31 Y. G. Stoyan and Andrey M. Chugay. Packing different cuboids with rotations and spheres into a cuboid. *Advances in Decision Sciences*, 2014, 2014. doi:10.1155/2014/571743.

32 Hu Zhang and Klaus Jansen. Scheduling malleable tasks. In *Handbook of Approximation Algorithms and Metaheuristics*. Chapman & Hall/CRC, 2007.

A Details of the $\texttt{HGaP}_k$ Algorithm

A.1 Details of the Weighting Function from $\operatorname{DLP}(\widehat{I})$

▶ **Transformation 34.** *Let (y, z) be a feasible solution to $\operatorname{DLP}(\widehat{I})$ (see Section 5.5.3 for the definition of $\operatorname{DLP}(\widehat{I})$). Let $s \in [t-1]$. Define $y_t := 0$ and $h_{t+1} := 0$. Then change y_s to $\max(y_s, y_{s+1} + (h_{s+1} - h_{s+2})z)$.*

▶ **Lemma 35.** *Let (y, z) be a feasible solution to $\operatorname{DLP}(\widehat{I})$ and $(\widehat{y}, z)$ be the result of applying Transformation 34 to (y, z) with parameter $s \in [t-1]$. Then $(\widehat{y}, z)$ is feasible for $\operatorname{DLP}(\widehat{I})$.*

Proof. For a configuration C, let $f(C, y, z) := C^T y + (1 - h_C)z$, where $C^T y := \sum_{j=1}^{t-1} C_j y_j$. Since (y, z) is feasible for $\operatorname{DLP}(\widehat{I})$, $f(C, y, z) \le 1$. As per Transformation 34,

$$\widehat{y}_j := \begin{cases} \max(y_s, y_{s+1} + (h_{s+1} - h_{s+2})z) & j = s \\ y_j & j \ne s \end{cases}.$$

If $y_s \ge y_{s+1} + (h_{s+1} - h_{s+2})z$, then $\widehat{y} = y$, so $(\widehat{y}, z)$ would be feasible for $\operatorname{DLP}(\widehat{I})$. So now assume that $y_s < y_{s+1} + (h_{s+1} - h_{s+2})z$.

Let C be a configuration. Define $C_t := 0$. Let

$$\widehat{C}_j := \begin{cases} 0 & j = s \\ C_s + C_{s+1} & j = s+1 \\ C_j & \text{otherwise} \end{cases}.$$

Then, $C^T\widehat{y} - \widehat{C}^T y = C_s\widehat{y}_s + C_{s+1}\widehat{y}_{s+1} - \widehat{C}_s y_s - \widehat{C}_{s+1}y_{s+1} = C_s(h_{s+1} - h_{s+2})z$.

Also, $h_{\widehat{C}} - h_C = \widehat{C}_s h_{s+1} + \widehat{C}_{s+1}h_{s+2} - C_s h_{s+1} - C_{s+1}h_{s+2} = -C_s(h_{s+1} - h_{s+2})$.

Since $h_{\widehat{C}} \le h_C \le 1$, $\widehat{C}$ is a configuration.

$$\begin{aligned} f(C, \widehat{y}, z) &= C^T\widehat{y} + (1 - h_C)z \\ &= (\widehat{C}^T y + C_s(h_{s+1} - h_{s+2})z) + (1 - h_{\widehat{C}} - C_s(h_{s+1} - h_{s+2}))z \\ &= f(\widehat{C}, y, z) \le 1. \end{aligned}$$

Therefore, $(\widehat{y}, z)$ is feasible for $\operatorname{DLP}(\widehat{I})$. ◀

▶ **Lemma 31.** *Let (y, z) be a feasible solution to* $\operatorname{DLP}(\widehat{I})$*. Let $(\widehat{y}, z)$ be the monotonization of (y, z). Then $(\widehat{y}, z)$ is a feasible solution to* $\operatorname{DLP}(\widehat{I})$*.*

Proof. $(\widehat{y}, z)$ can be obtained by multiple applications of Transformation 34: first with $s = t-1$, then $s = t-2$, and so on till $s = 1$. By Lemma 35, $(\widehat{y}, z)$ is feasible for $\operatorname{DLP}(\widehat{I})$. ◀

▶ **Lemma 32.** *η is a monotonic weighting function.*

Proof. η is monotonic by the definition of monotonization.

Let $X \subseteq (0, 1]$ be a finite set such that $\operatorname{sum}(X) \le 1$. Let $X_0 := X \cap [0, h_t)$, let $X_1 := X \cap [h_2, 1]$ and for $2 \le j \le t-1$, let $X_j := X \cap [h_{j+1}, h_j)$. Let $C \in \mathbb{Z}^{t-1}_{\ge 0}$ such that $C_j := |X_j|$. Let $h_C := \sum_{j=1}^{t-1} C_j h_{j+1}$.

$$\begin{aligned} 1 \ge \operatorname{sum}(X) &= \operatorname{sum}(X_0) + \sum_{j=1}^{t-1}\operatorname{sum}(X_j) \\ &\ge \operatorname{sum}(X_0) + \sum_{j=1}^{t-1} C_j h_{j+1} && \text{(for } j \ge 1\text{, each element in } X_j \text{ is at least } h_{j+1}) \\ &= \operatorname{sum}(X_0) + h_C. \end{aligned}$$

Since $h_C \le 1 - \operatorname{sum}(X_0) \le 1$, C is a configuration. Therefore,

$$\begin{aligned} \sum_{x \in X}\eta(x) = \sum_{j=0}^{t-1}\sum_{x \in X_j}\eta(x) &= z^* \operatorname{sum}(X_0) + \sum_{j=1}^{t-1} C_j\widehat{y}_j && \text{(by definition of } \eta) \\ &\le (1 - h_C)z^* + C^T\widehat{y} && (h_C \le 1 - \operatorname{sum}(X_0)) \\ &\le 1. && \text{(} C \text{ is a configuration and } (\widehat{y}, z^*) \text{ is feasible for } \operatorname{DLP}(\widehat{I}) \text{ by Lemma 31)} \end{aligned}$$

◀

A.2 Guessing Shelves and Bins

We want `guessShelves`$(\widehat{\mathcal{I}}, \delta)$ to return all possible packings of empty shelves into at most $n := |\widehat{\mathcal{I}}|$ bins such that each packing is structured for $(\operatorname{flat}(\widehat{\mathcal{I}}), \delta)$.

Let $H := \{h(i) : i \in \operatorname{flat}(\widehat{\mathcal{I}})\}$. Let $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$. `guessShelves`$(\widehat{\mathcal{I}}, \delta)$ starts by picking the distinct heights of shelves by iterating over all subsets of H of size at most $\lceil 1/\delta^2 \rceil$. There are at most $N^{\lceil 1/\delta^2 \rceil} + 1$ such subsets. Let $\widetilde{H} := \{h_1, h_2, \ldots, h_t\}$ be one such guess, where $t \le \lceil 1/\delta^2 \rceil$. Without loss of generality, assume $h_1 > h_2 > \ldots > h_t > \delta$.

Next, `guessShelves` needs to decide the number of shelves of each height and a packing of those shelves into bins. Let $C \in \mathbb{Z}^t_{\ge 0}$ such that $h_C := \sum_{j=1}^{t-1} C_j h_j \le 1$. Then C is called a configuration. C represents a set of shelves that can be packed into a bin and where C_j shelves have height h_j. Let $\mathcal{C}$ be the set of all configurations. We can pack at most $\lceil 1/\delta \rceil - 1$ items into a bin because $h_t > \delta$. By Lemma 1, we get

$$|\mathcal{C}| \le \binom{\lceil 1/\delta \rceil - 1 + t}{t} \le \binom{\lceil 1/\delta \rceil - 1 + \lceil 1/\delta^2 \rceil}{\lceil 1/\delta \rceil - 1} \le \left(\left\lceil \frac{1}{\delta^2} \right\rceil + 1 \right)^{1/\delta}.$$

There can be at most n bins, and `guessShelves` has to decide the configuration of each bin. By Lemma 1, the number of ways of doing this is at most $\binom{|\mathcal{C}|+n}{|\mathcal{C}|} \le (n+1)^{|\mathcal{C}|}$. Therefore, `guessShelves` computes all configurations and then iterates over all $\binom{|\mathcal{C}|+n}{|\mathcal{C}|}$ combinations of these configs. This completes the description of `guessShelves` and proves Theorem 13.

A.3 `chooseAndPack`

`chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ takes as input a set $\widehat{\mathcal{I}}$ of 2D itemsets, a packing P of empty shelves into bins and constant $\delta \in (0, 1)$. It tries to pack $\widehat{\mathcal{I}}$ into P and one additional shelf. Before we design `chooseAndPack`, let us see how to handle a special case. $\widehat{\mathcal{I}}$ is called δ-*simple* iff the width of each δ-large item in $\operatorname{flat}(\widehat{\mathcal{I}})$ is a multiple of $1/|\widehat{\mathcal{I}}|$.

Let P be a bin packing of empty shelves. Let $h_1 > h_2 > \ldots > h_t$ be the distinct heights of the shelves in P, where $h_t > \delta$. We will use dynamic programming to either pack a simple instance $\widehat{\mathcal{I}}$ into P or claim that no assortment of $\widehat{\mathcal{I}}$ can be packed into P. Call this algorithm `simpleChooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$.

Let $\widehat{\mathcal{I}} := \{I_1, I_2, \ldots, I_n\}$. For $j \in \{0, 1, \ldots, n\}$, define $\widehat{\mathcal{I}}_j := \{I_1, I_2, \ldots, I_j\}$, i.e., $\widehat{\mathcal{I}}_j$ contains the first j itemsets from $\widehat{\mathcal{I}}$. Let $\vec{u} := [u_1, u_2, \ldots, u_t] \in \{0, 1, \ldots, n^2\}^t$ be a vector. Let $\Phi(j, \vec{u})$ be the set of all assortments of $\widehat{\mathcal{I}}_j$ that can be packed into t shelves, where the r^{th} shelf has height h_r and width u_r/n. For a set K of items, define $\operatorname{smallArea}(K)$ as the total area of δ-small items in K. Define $g(j, \vec{u}) := \min_{K \in \Phi(j,\vec{u})} \operatorname{smallArea}(K)$. If $\Phi(j, \vec{u}) = \emptyset$, then we let $g(j, \vec{u}) = \infty$.

We will show how to compute $g(j, \vec{u})$ for all $j \in \{0, 1, \ldots, n\}$ and all $\vec{u} \in \{0, 1, \ldots, n^2\}^t$ using dynamic programming. Let there be n_r shelves in P having height h_r. Then for $j = n$ and $u_r = n_r n$, $\widehat{\mathcal{I}}$ can be packed into P iff $g(j, \vec{u})$ is at most the area of non-shelf space in P.

Note that in any solution K corresponding to $g(j, \vec{u})$, we can assume without loss of generality that the item i from $K \cap I_j$ is placed in the smallest shelves possible. This is because we can always swap i with the slices of items in those shelves. This observation gives us the following recurrence relation for $g(j, \vec{u})$:

$$g(j, \vec{u}) = \begin{cases} \infty & \text{if } u_j < 0 \text{ for some } j \in [t] \\ 0 & \text{if } n = 0 \text{ and } u_j \ge 0 \text{ for all } j \in [t] \\ \min_{i \in I_j} \begin{pmatrix} \operatorname{smallArea}(\{i\}) \\ + g(j-1, \operatorname{reduce}(\vec{u}, i)) \end{pmatrix} & \text{if } n > 0 \text{ and } u_j \ge 0 \text{ for all } j \in [t] \end{cases} \tag{1}$$

Here $\operatorname{reduce}(\vec{u}, i)$ is a vector obtained as follows: If i is δ-small, then $\operatorname{reduce}(\vec{u}, i) := \vec{u}$. Otherwise, initialize x to $w(i)$. Let p_i be the largest integer r such that $h(i) \le h_r$. For r varying from p_i to 2, subtract $\min(x, u_j)$ from x and u_j. Then subtract x from u_1. The new value of $\vec{u}$ is defined to be the output of $\operatorname{reduce}(\vec{u}, i)$.

The recurrence relation allows us to compute $g(j, \vec{u})$ for all j and $\vec{u}$ using dynamic programming in time $O(Nn^{2t})$ time, where $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$. With a bit more work, we can also compute the corresponding assortment K, if one exists. Therefore, `simpleChooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ computes a packing of $\widehat{\mathcal{I}}$ into P if one exists, or returns `null` if no assortment of $\widehat{\mathcal{I}}$ can be packed into P.

Now we will look at the case where $\widehat{\mathcal{I}}$ is not δ-simple. Let $\widehat{\mathcal{I}}'$ be the instance obtained by rounding up the width of each δ-large item in $\widehat{\mathcal{I}}$ to a multiple of $1/n$, where $n := |\widehat{\mathcal{I}}|$. Let $\overline{P}$ be the bin packing obtained by adding another bin to P containing a single shelf of height h_1. `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ computes $\widehat{\mathcal{I}}'$ and $\overline{P}$ and returns the output of `simpleChooseAndPack`$(\widehat{\mathcal{I}}', \overline{P}, \delta)$.

▶ **Theorem 15.** *If the output of* `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ *is not* `null`*, then the output* $\overline{P}$ *is a shelf-based δ-fractional packing of some assortment of* $\widehat{\mathcal{I}}$ *such that* $|\overline{P}| \le |P| + 1$ *and the distinct shelf heights in* $\overline{P}$ *are the same as that in* P.

Proof. Follows from the definition of `simpleChooseAndPack`. ◀

▶ **Theorem 14.** *If there exists an assortment* $\widehat{K}$ *of* $\widehat{\mathcal{I}}$ *having a structured δ-fractional bin packing P, then* `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ *does not output* `null`.

Proof. Let $\widehat{K}'$ be the items obtained by rounding up the width of each item in $\widehat{K}$ to a multiple of $1/n$. Then $\widehat{K}'$ is an assortment of $\widehat{\mathcal{I}}'$. We will show that $\widehat{K}'$ fits into $\overline{P}$, so `simpleChooseAndPack`$(\widehat{\mathcal{I}}', \overline{P}, \delta)$ will not output `null`.

Slice each item $i \in \widehat{K}'$ into two pieces using a vertical cut such that one piece has width equal to the original width of i in $\widehat{K}$, and the other piece has width less than $1/n$. This splits $\widehat{K}'$ into sets $\widehat{K}$ and T. T contains at most n items, each of width less than $1/n$. Therefore, we can pack $\widehat{K}$ into P and we can pack T into the newly-created shelf of height h_1. Therefore, $\widehat{K}'$ can be packed into $\overline{P}$, so `simpleChooseAndPack`$(\widehat{\mathcal{I}}', \overline{P}, \delta)$ won't output `null`. ◀

▶ **Theorem 16.** `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ *runs in* $O(Nn^{2\lceil 1/\delta^2\rceil})$ *time. Here* $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$, $n := |\widehat{\mathcal{I}}|$.

Proof. The running time of `chooseAndPack`$(\widehat{\mathcal{I}}, P, \delta)$ is dominated by computing $g(j, \vec{u})$ for all j and $\vec{u}$, which takes $O(Nn^{2t})$ time. Since P is structured for $(\widehat{\mathcal{I}}, \delta)$, the number of distinct shelves in P, which is t, is at most $\lceil 1/\delta^2 \rceil$. ◀

A.4 `inflate`

Let I be a set of dD items. Let P be a shelf-based δ-fractional bin packing of $\widehat{I} :=$ `round`(I) into m bins, where the shelves have t distinct heights: $h_1 > \ldots > h_t > \delta$. We will design an algorithm `inflate`(P) that packs I into approximately $|P|$ bins. Let $\widehat{I}_L := \{i \in \widehat{I} : h(i) > \delta\}$ and $\widehat{I}_S := \widehat{I} - \widehat{I}_L$. Let there be Q distinct base types in I (so $Q \le k^{d-1}$).

A.4.1 Separating Base Types

We will now impose an additional constraint over P: items in each shelf must have the same btype. This will be helpful later, when we will try to compute a packing of dD items I.

Separating base types of $\widehat{I}_S$ is easy, since we can slice them in both directions. An analogy is to think of a mixture of multiple immiscible liquids settling into equilibrium.

Let there be n_j shelves of height h_j. Let $\widehat{I}_j$ be the items packed into shelves of height h_j. Therefore, $w(\widehat{I}_j) \le n_j$. Let $\widehat{I}_{j,q} \subseteq \widehat{I}_j$ be the items of base type $q \in [Q]$.

For each q, pack $\widehat{I}_{j,q}$ into $\lceil w(\widehat{I}_{j,q}) \rceil$ shelves of height h_j (slicing items if needed). For these newly-created shelves, define the btype of the shelf to be the btype of the items in it. Let the number of newly-created shelves of height h_j be n'_j. Then

$$n'_j = \sum_{q=1}^{Q} \lceil w(\widehat{I}_{j,q}) \rceil < \sum_{q=1}^{Q} w(\widehat{I}_{j,q}) + Q \le n_j + Q \implies n'_j \le n_j + Q - 1.$$

n_j of these shelves can be packed into existing bins in place of the old shelves. The remaining $n'_j - n_j \le Q - 1$ shelves can be packed on the base of new bins.

Therefore, by using at most $t(Q-1)$ new bins, we can ensure that for every shelf, all items in that shelf have the same btype. These new bins don't contain any items from $\widehat{I}_S$. Call this new bin packing P'. This transformation takes $O(|I| d \log |I|)$ time.

A.4.2 Forbidding Horizontal Slicing

We will now use P' to compute a shelf-based bin packing P'' of $\widehat{I}$ where items in $\widehat{I}$ can be sliced using vertical cuts only.

Let $\widehat{I}_{q,S}$ be the items in $\widehat{I}_S$ of base type q. Pack items $\widehat{I}_{q,S}$ into shelves using canShelv. Suppose canShelv used m_q shelves to pack $\widehat{I}_{q,S}$. For $j \in [m_q]$, let $h_{q,j}$ be the height of the j^{th} shelf. Let $H_q := \sum_{j=1}^{m_q} h_{q,j}$ and $H := \sum_{q=1}^{Q} H_q$. Since for $j \in [m_q - 1]$, all items in the j^{th} shelf have height at least $h_{q,j+1}$,

$$a(\widehat{I}_{q,S}) > \sum_{j=1}^{m_q-1} h_{q,j+1} \ge H_q - h_{q,1} \ge H_q - \delta.$$

Therefore, $H < a(\widehat{I}_S) + Q\delta$. Let $\widehat{J}_S$ be the set of these newly-created shelves.

Use Next-Fit to pack $\widehat{J}_S$ into the space used by $\widehat{I}_S$ in P'. $\widehat{I}_S$ uses at most m bins in P' (recall that $m := |P|$). A height of less than δ will remain unpacked in each of those bins. The total height occupied by $\widehat{I}_S$ in P' is $a(\widehat{I}_S)$. Therefore, Next-Fit will pack a height of more than $a(\widehat{I}_S) - \delta m$.

Some shelves in $\widehat{J}_S$ may still be unpacked. Their total height will be less than $H - (a(\widehat{I}_S) - \delta m) < \delta(Q + m)$. We will pack these shelves into new bins using Next-Fit. The number of new bins used is at most $\lceil \delta(Q+m)/(1-\delta) \rceil$. Call this bin packing P''. The number of bins in P'' is at most $m' := m + t(Q-1) + \lceil \delta(Q+m)/(1-\delta) \rceil$.

A.4.3 Shelf-Based dD packing

We will now show how to convert the packing P'' of $\widehat{I}$ that uses m' bins into a packing of I that uses m' dD bins.

First, we repack the items into the shelves. For each $q \in [Q]$, let $\widehat{J}_q$ be the set of shelves in P'' of btype q. Let $\widehat{I}^{[q]}$ be the items packed into $\widehat{J}_q$. Compute $\widehat{J}_q^* := \text{canShelv}(\widehat{I}^{[q]})$ and pack the shelves $\widehat{J}_q^*$ into $\widehat{J}_q$. This is possible by Lemma 23.

This repacking gives us an ordering of shelves in $\widehat{J}_q$. Number the shelves from 1 onwards. All items have at most 2 slices. If an item has 2 slices, and one slice is packed into shelf number p, then the other slice is packed into shelf number $p+1$. The slice in shelf p is called the leading slice. Every shelf has at most one leading slice.

Let S_j be the j^{th} shelf of $\widehat{J}_q$. Let R_j be the set of unsliced items in S_j and the item whose leading slice is in S_j. Order the items in R_j arbitrarily, except that the sliced item, if any, should be last. Then $w(R_j - \text{last}(R_j)) < 1$. So, we can use $\texttt{HDH-unit-pack}_k^{[q]}(R_j)$ to pack R_j into a $(d-1)$D bin. This $(d-1)$D bin gives us a dD shelf whose height is the same as that of S_j. On repeating this process for all shelves in $\widehat{J}_q$ and for all $q \in [Q]$, we get a packing of I into shelves. Since each dD shelf corresponds to a shelf in P'' of the same height, we can pack these dD shelves into bins in the same way as P''. This gives us a bin packing of I into m' bins.

A.4.4 The Algorithm

Appendices A.4.1–A.4.3 describe how to convert a shelf-based δ-fractional packing P of $\widehat{I}$ having t distinct shelf heights into a shelf-based dD bin packing of I. We call this conversion algorithm `inflate`.

It is easy to see that the time taken by `inflate` is $O(|I|d \log |I|)$.

If P has m bins, then the number of bins in $\texttt{inflate}(P)$ is at most

$$m + t(Q-1) + \left\lceil \frac{\delta(Q+m)}{1-\delta} \right\rceil < \frac{m}{1-\delta} + t(Q-1) + 1 + \frac{\delta Q}{1-\delta}.$$

This proves Theorem 17.

A.5 Improving Running Time

For simplicity of presentation, we left out some opportunities for improving the running time of $\texttt{HGaP}_k$. Here we briefly describe a way of speeding up $\texttt{HGaP}_k$ which reduces its running time from $O(N^{1+\lceil 1/\delta^2 \rceil} n^{R+2\lceil 1/\delta^2 \rceil} + Nd + nd \log n)$ to $O(N^{1+\lceil 1/\delta^2 \rceil} n^{2\lceil 1/\delta^2 \rceil} + Nd + nd \log n)$. Here $N := |\operatorname{flat}(\widehat{\mathcal{I}})|$, $n := |\widehat{\mathcal{I}}|$, $\delta := \varepsilon/(2+\varepsilon)$ and $R := \binom{\lceil 1/\delta^2 \rceil + \lceil 1/\delta \rceil - 1}{\lceil 1/\delta \rceil - 1} \le (1 + \lceil 1/\delta^2 \rceil)^{1/\delta}$.

In `guessShelves`, we guess two things simultaneously: (i) the number and heights of shelves (ii) the packing of the shelves into bins. This allows us to guess the optimal structured δ-fractional packing. But we don't need that; an approximate structured packing would do.

Therefore, we only guess the number and heights of shelves. We guess at most $N^{\lceil 1/\delta^2 \rceil} + 1$ distinct heights of shelves, and by Lemma 1, we guess at most $(n+1)^{\lceil 1/\delta^2 \rceil}$ vectors of shelf-height frequencies. Then we can use Lueker and Fernandez de la Vega's $O(n \log n)$-time APTAS for 1BP [15] to pack the shelves into bins.

Also, once we guess the distinct heights of shelves, we don't need to run `chooseAndPack` afresh for every packing of empty shelves. We can reuse the dynamic programming table.

The running time is, therefore,

$$\begin{aligned} &O\left(N^{\lceil 1/\delta^2 \rceil}\left(n^{\lceil 1/\delta^2 \rceil} n \log n + N n^{2\lceil 1/\delta^2 \rceil}\right) + Nd + nd \log n\right) \\ &= O(N^{1+\lceil 1/\delta^2 \rceil} n^{2\lceil 1/\delta^2 \rceil} + Nd + nd \log n). \end{aligned}$$

Resilience of Timed Systems

S. Akshay
IIT Bombay, Mumbai, India

Blaise Genest
Univ. Rennes, CNRS, IRISA, Rennes, France

Loïc Hélouët
Univ. Rennes, INRIA, IRISA, Rennes, France

S. Krishna
IIT Bombay, Mumbai, India

Sparsa Roychowdhury
IIT Bombay, Mumbai, India

Abstract

This paper addresses reliability of timed systems in the setting of *resilience*, that considers the behaviors of a system when unspecified timing errors such as missed deadlines occur. Given a fault model that allows transitions to fire later than allowed by their guard, a system is *universally resilient* (or self-resilient) if after a fault, it always returns to a timed behavior of the non-faulty system. It is *existentially resilient* if after a fault, there exists a way to return to a timed behavior of the non-faulty system, that is, if there exists a controller which can guide the system back to a normal behavior. We show that universal resilience of timed automata is undecidable, while existential resilience is decidable, in EXPSPACE. To obtain better complexity bounds and decidability of universal resilience, we consider untimed resilience, as well as subclasses of timed automata.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models

Keywords and phrases Timed automata, Fault tolerance, Integer-resets, Resilience

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.33

Funding Work supported by the DST/CEFIPRA/INRIA associated team EQuaVE and the Maveriq ANR project.

1 Introduction

Timed automata [2] are a natural model for cyber-physical systems with real-time constraints that have led to an enormous body of theoretical and practical work. Formally, timed automata are finite-state automata equipped with real valued variables called clocks, that measure time and can be reset. Transitions are guarded by logical assertions on the values of these clocks, which allows for the modeling of real-time constraints, such as the time elapsed between the occurrence of two events. A natural question is whether a real-time system can handle unexpected delays. This is a crucial need when modeling systems that must follow a priori schedules such as trains, metros, buses, etc. Timed automata are not a priori tailored to handle unspecified behaviors: guards are mandatory time constraints, i.e., transition firings must occur within the prescribed delays. Hence, transitions cannot occur late, except if late transitions are explicitly specified in the model. This paper considers the question of resilience for timed automata, i.e., study whether a system returns to its normal specified timed behavior after an unexpected but unavoidable delay.

Several works have addressed timing errors as a question of *robustness* [10, 8, 7], to guarantee that a property of a system is preserved for some small imprecision of up to ϵ time units. Timed automata have an ideal representation of time: if a guard of a transition contains a constraint of the form $x = 12$, it means that this transition occurs *exactly* when

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 33; pp. 33:1–33:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Summary of results for resilience.

	Universal Resilience	Existential Resilience
Timed	Undecidable for TA (Prop. 18) EXPSPACE-C for IRTA (Thm. 20)	EXPSPACE (Thm. 14) PSPACE-Hard (Thm. 15, Thm. 32)
Untimed	EXPSPACE-C (Thm. 21)	PSPACE-C (Thm. 16, Rmk. 17)

the value of clock x is 12. Such an arbitrary precision is impossible in an implementation [10]. One way of addressing this is through guard enlargement, i.e., by checking that there exists a small value $\epsilon > 0$ such that after replacing guards of the form $x \in [a, b]$ by $x \in [a - \epsilon, b + \epsilon]$, the considered property is still valid, as shown in [7] for ω-regular properties. In [15], robust automata are defined that accept timed words and their neighbors i.e., words whose timing differences remain at a small distance, while in [16, 12, 19, 1], the authors consider robustness via modeling clock drifts. Our goal is different: rather than being robust w.r.t. to slight imprecisions, we wish to check the capacity to recover from a *possibly large* time deviation. Thus, for a bounded number of steps, the system can deviate arbitrarily, after which, it must return to its specified timed behavior.

The first contribution of this paper is a formalization of resilience in timed automata. We capture delayed events with *faulty transitions*. These occur at dates deviating from the original specification and may affect clock values for an arbitrarily long time, letting the system diverge from its expected behavior. A system is *resilient* if it recovers in a finite number of steps after the fault. More precisely, we define two variants. A timed automaton is K-$\forall$-*resilient* if for *every* faulty timed run, the behavior of the system K steps after the fault cannot be distinguished from a non-faulty behavior. In other words, the system *always* repairs itself in at most K steps after a fault, whenever a fault happens. This means that, after a fault happens, *all* the subsequent behaviors (or extensions) of the system are restored to normalcy within K steps. A timed automaton is K-$\exists$-*resilient* if for every timed run ending with a fault, there exists *an* extension in which, the behavior of the system K steps after the fault cannot be distinguished from a non-faulty behavior. There can still be some extensions which are beyond repair, or take more than K steps after fault to be repaired, but there is a guarantee of at least one repaired extension within K steps after the fault. In the first case, the timed automaton is fully self-resilient, while in the second case, there exist controllers choosing dates and transitions so that the system gets back to a normal behavior. We also differentiate between timed and untimed settings: in timed resilience recovered behaviors must be indistinguishable w.r.t. actions and dates, while in untimed resilience recovered behaviors only need to match actions.

Our results are summarized in Table 1: we show that the question of universal resilience and inclusion of timed languages are inter-reducible. Thus *timed* universal resilience is undecidable in general, and decidable for classes for which inclusion of timed languages is decidable and which are stable under our reduction. This includes the class of Integer Reset Timed Automata (IRTA) [18] for which we obtain EXPSPACE containment. Further, *untimed* universal resilience is EXPSPACE-Complete in general.

Our main result concerns existential resilience, which requires new non-trivial core contributions because of the $\forall\exists$ quantifier alternation. The classical region construction is not precise enough: we introduce *strong regions* and develop novel techniques based on these, which ensure that all runs following a strong region have (i) matching integral time elapses, and (ii) the fractional time can be re-timed to visit the same set of locations and (usual) regions. Using this technique, we show that existential timed resilience is decidable, in EXPSPACE. We also show that untimed existential resilience is PSPACE-Complete.

Related Work. Resilience has been considered with different meanings: In [13], faults are modeled as conflicts, the system and controller as *deterministic* timed automata, and avoiding faults reduces to checking reachability. This is easier than universal resilience which reduces to timed language inclusion, and existential resilience which requires a new notion of regions. In [14] a system, modeled as an untimed I/O automaton, is considered "sane" if its runs contain at most k errors, and allow a sufficient number s of error-free steps between two violations of an LTL property. It is shown how to synthesize a sane system, and compute (Pareto-optimal) values for s and k. In [17], the objective is to synthesize a transducer E, possibly with memory, that reads a timed word σ produced by a timed automaton $\mathcal{A}$, and outputs a timed word $E(\sigma)$ obtained by deleting, delaying or forging new timed events, such that $E(\sigma)$ satisfies some timed property. A related problem, shield synthesis [5], asks given a network of deterministic I/O timed automata $\mathcal{N}$ that communicate with their environment, to synthesize two additional components, a pre-shield, that reads outputs from the environment and produces inputs for $\mathcal{N}$, and a post-shield, that reads outputs from $\mathcal{N}$ and produces outputs to the environment to satisfy timed safety properties when faults (timing, location errors,...) occur. Synthesis is achieved using timed games. Unlike these, our goal is not to avoid violation of a property, but rather to verify that the system *recovers within boundedly many steps*, from a possibly large time deviation w.r.t. its behavior. Finally, *faults* in timed automata have also been studied in a diagnosis setting, e.g. in [6], where faults are detected within a certain delay from partial observation of runs.

2 Preliminaries

Let Σ be a finite non-empty alphabet and $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ a set of finite or infinite words over Σ. $\mathbb{R}, \mathbb{R}_{\geq 0}, \mathbb{Q}, \mathbb{N}$ respectively denote the set of real numbers, non-negative reals, rationals, and natural numbers. We write $(\Sigma \times \mathbb{R}_{\geq 0})^\infty = (\Sigma \times \mathbb{R}_{\geq 0})^* \cup (\Sigma \times \mathbb{R}_{\geq 0})^\omega$ for finite or infinite timed words over Σ. A finite (infinite) timed word has the form $w = (a_1, d_1) \dots (a_n, d_n)$ (resp. $w = (a_1, d_1) \dots)$ where for every i, $d_i \leq d_{i+1}$. For $i \leq j$, we denote by $w_{[i,j]}$ the sequence $(a_i, d_i) \dots (a_j, d_j)$. The *untiming* of a timed word $w \in (\Sigma \times \mathbb{R}_{\geq 0})^\infty$ denoted $Unt(w)$, is its projection on the first component, and is a word in Σ^∞. A *clock* is a real-valued variable x and an *atomic clock constraint* is an inequality of the form $a \bowtie_l x \bowtie_u b$, with $\bowtie_l, \bowtie_u \in \{\leq, <\}$, $a \in \mathbb{N}, b \in \mathbb{N} \cup \{\infty\}$. An atomic *diagonal constraint* is of the form $a \bowtie_l x - y \bowtie_u b$, where x and y are different clocks. *Guards* are conjunctions of atomic constraints on a set X of clocks.

▶ **Definition 1.** *A timed automaton [2] is a tuple $\mathcal{A} = (L, I, X, \Sigma, T, F)$ with finite set of locations L, initial locations $I \subseteq L$, finitely many clocks X, finite action set Σ, final locations $F \subseteq L$, and transition relation $T \subseteq L \times \mathcal{G} \times \Sigma \times 2^X \times L$ where $\mathcal{G}$ are guards on X.*

A *valuation* of a set of clocks X is a map $\nu : X \to \mathbb{R}_{\geq 0}$ that associates a non-negative real value to each clock in X. For every clock x, $\nu(x)$ has an integral part $\lfloor \nu(x) \rfloor$ and a fractional part $\mathsf{frac}(\nu(x)) = \nu(x) - \lfloor \nu(x) \rfloor$. We will say that a valuation ν on a set of clocks X satisfies a guard g, denoted $\nu \models g$ if and only if replacing every $x \in X$ by $\nu(x)$ in g yields a tautology. We will denote by $[g]$ the set of valuations that satisfy g. Given $\delta \in \mathbb{R}_{\geq 0}$, we denote by $\nu + \delta$ the valuation that associates value $\nu(x) + \delta$ to every clock $x \in X$. A *configuration* is a pair $C = (l, \nu)$ of a location of the automaton and valuation of its clocks. The semantics of a timed automaton is defined in terms of discrete and timed moves from a configuration to the next one. A *timed move of duration δ* lets $\delta \in \mathbb{R}_{\geq 0}$ time units elapse from a configuration $C = (l, \nu)$ which leads to configuration $C' = (l, \nu + \delta)$. A *discrete move* from configuration

$C = (l, \nu)$ consists of taking one of the transitions leaving l, i.e., a transition of the form $t = (l, g, a, R, l')$ where g is a guard, $a \in \Sigma$ a particular action name, R is the set of clocks reset by the transition, and l' the next location reached. A discrete move with transition t is allowed only if $\nu \models g$. Taking transition t leads the automaton to configuration $C' = (l', \nu')$ where $\nu'(x) = \nu(x)$ if $x \notin R$, and $\nu'(x) = 0$ otherwise.

▶ **Definition 2** (Runs, Maximal runs, Accepting runs). *An (infinite)* run *of a timed automaton $\mathcal{A}$ is a sequence $\rho = (l_0, \nu_0) \xrightarrow{(t_1,d_1)} (l_1, \nu_1) \xrightarrow{(t_2,d_2)} \cdots$ where every pair (l_i, ν_i) is a configuration, and there exists an (infinite) sequence of timed and discrete moves $\delta_1.t_1.\delta_2.t_2 \ldots$ in $\mathcal{A}$ such that $\delta_i = d_{i+1} - d_i$, and a timed move of duration δ_i from (l_i, ν_i) to $(l_i, \nu_i + \delta_i)$ and a discrete move from $(l_i, \nu_i + \delta_i)$ to (l_{i+1}, ν_{i+1}) via transition t_i. A run is* maximal *if it is infinite, or if it ends at a location with no outgoing transitions. A finite run is* accepting *if its last location is final, while an infinite run is* accepting *if it visits accepting locations infinitely often.*

We assume that all runs start from a configuration (l_0, ν_0), where $l_0 \in I$ and ν_0 is the initial valuation, assigning value 0 to every clock of X. One can associate a finite/infinite *timed word* w_ρ to every run ρ of $\mathcal{A}$ by letting $w_\rho = (a_1, d_1)\ (a_2, d_2) \ldots (a_n, d_n) \ldots$, where a_i is the action in transition t_i and d_i is the time stamp of t_i in ρ. A (finite/infinite) *timed word* w is accepted by $\mathcal{A}$ if there exists a (finite/infinite) accepting run ρ such that $w = w_\rho$. The *timed language* of $\mathcal{A}$ is the set of all timed words accepted by $\mathcal{A}$, and is denoted by $\mathcal{L}(\mathcal{A})$. The *untimed language* of $\mathcal{A}$ is the language $Unt(\mathcal{L}(\mathcal{A})) = \{Unt(w) \mid w \in \mathcal{L}(\mathcal{A})\}$. As shown in [2], the untimed language of a timed automaton can be captured by an abstraction called the *region automaton.* Formally, given a clock x, let c_x be the largest constant in an atomic constraint of a guard of $\mathcal{A}$ involving x. Two valuations ν, ν' of clocks in X are *equivalent*, written $\nu \sim \nu'$ if and only if:
i) $\forall x \in X$, either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or both $\nu(x) \geq c_x$ and $\nu'(x) \geq c_x$
ii) $\forall x, y \in X$ with $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, $\mathsf{frac}(\nu(x)) \leq \mathsf{frac}(v(y))$ iff $\mathsf{frac}(\nu'(x)) \leq \mathsf{frac}(\nu'(y))$
iii) For all $x \in X$ with $\nu(x) \leq c_x$, $\mathsf{frac}(\nu(x)) = 0$ iff $\mathsf{frac}(\nu'(x)) = 0$.

A *region* r of $\mathcal{A}$ is the equivalence class induced by $\sim$. For a valuation ν, we denote by $[\nu]$ the region of ν, i.e., its equivalence class. We will also write $\nu \in r$ (ν is a valuation in region r when $r = [\nu]$. For a given automaton $\mathcal{A}$, there exists only a finite number of regions, bounded by 2^K, where K is the size of the constraints set in $\mathcal{A}$. It is well known for a clock constraint ψ that, if $\nu \sim \nu'$, then $\nu \models \psi$ if and only if $\nu' \models \psi$. A region r' is a *time successor* of another region r if for every $\nu \in r$, there exists $\delta \in \mathbb{R}_{>0}$ such that $\nu + \delta \in r'$. We denote by $Reg(X)$ the set of all possible regions of the set of clocks X. A region r satisfies a guard g if and only if there exists a valuation $\nu \in r$ such that $\nu \models g$. The region automaton of a timed automaton $\mathcal{A} = (L, I, X, \Sigma, T, F)$ is the untimed automaton $\mathcal{R}(\mathcal{A}) = (S_R, I_R, \Sigma, T_R, F_R)$ that recognizes the untimed language $Unt(\mathcal{L}(\mathcal{A}))$. States of $\mathcal{R}(\mathcal{A})$ are of the form (l, r), where l is a location of $\mathcal{A}$ and r a region, i.e., $S_R \subseteq L \times Reg(X), I_R \subseteq I \times Reg(X)$, and $F_R \subseteq F \times Reg(X)$. The transition relation T_R is such that $\big((l, r), a, (l', r')\big) \in T_R$ if there exists a transition $t = (l, g, a, R, l') \in T$ such that there exists a time successor region r'' of r such that r'' satisfies the guard g, and r' is obtained from r'' by resetting values of clocks in R. The size of the region automaton is the number of states in $\mathcal{R}(\mathcal{A})$ and is denoted $|\mathcal{R}(\mathcal{A})|$. For a region r defined on a set of clocks Y, we define a projection operator $\Pi_X(r)$ to represent the region r projected on the set of clocks $X \subseteq Y$. Let $\rho = (l_0, \nu_0) \xrightarrow{(t_1,d_1)} (l_1, \nu_1) \cdots$ be a run of $\mathcal{A}$, where every t_i is of the form $t_i = (l_i, g_i, a_i, R_i, l'_i)$. The *abstract run* $\sigma_\rho = (l_0, r_0) \xrightarrow{a_1} (l_1, r_1) \cdots$ of ρ is a path in the region automaton $\mathcal{R}(\mathcal{A})$ such that, $\forall i \in \mathbb{N}, r_i = [\nu_i]$. We represent runs using variables ρ, π and the corresponding abstract runs with σ_ρ, σ_π respectively. The automaton $\mathcal{R}(\mathcal{A})$ can be used to prove non-emptiness of $\mathcal{L}(\mathcal{A})$, as $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{R}(\mathcal{A})$ accepts some word.

3 Resilience Problems

We define the semantics of timed automata when perturbations can delay the occurrence of an action. Consider a transition $t = (l, g, a, R, l')$, with $g ::= x \leq 10$, where action a can occur as long as x has not exceeded 10. Timed automata have an idealized representation of time, and do not consider perturbations that occur in real systems. Consider, for instance that "a" is a physical event planned to occur at a maximal time stamp 10: a water tank reaches its maximal level, a train arrives in a station etc. These events can be delayed, and nevertheless occur. One can even consider that uncontrollable delays are part of the normal behavior of the system, and that $\mathcal{L}(\mathcal{A})$ is the ideal behavior of the system, when all delays are met. In the rest of the paper, we propose a fault model that assigns a maximal error to each fireable action. This error model is used to encode the fact that an action might occur at a greater date than allowed in the original model semantics.

▶ **Definition 3** (Fault model). *A* fault model *$\mathcal{P}$ is a map $\mathcal{P} : \Sigma \to \mathbb{Q}_{\geq 0}$ that associates to every action in $a \in \Sigma$ a possible maximal delay $\mathcal{P}(a) \in \mathbb{Q}_{\geq 0}$.*

For simplicity, we consider only executions in which a single timing error occurs. The perturbed semantics defined below easily adapts to a setting with multiple timing errors. With a fault model, we can define a new timed automaton, for which every run $\rho = (l_0, \nu_0) \xrightarrow{(t_1,d_1)} (l_1, \nu_1) \xrightarrow{(t_2,d_2)} \cdots$ contains *at most* one transition $t_i = (l, g, a, r, l')$ occurring later than allowed by guard g, and agrees with a run of $\mathcal{A}$ until this faulty transition is taken.

▶ **Definition 4** (Enlargement of a guard). *Let ϕ be an inequality of the form $a \bowtie_l x \bowtie_u b$, where $\bowtie_l, \bowtie_u \in \{\leq, <\}$. The* enlargement *of ϕ by a time error δ is the inequality $\phi_{\rhd\delta}$ of the form $a \bowtie_l x \leq b + \delta$. Let g be a guard of the form*

$$g = \bigwedge_{i \in 1..m} \phi_i = a_i \bowtie_{l_i} x_i \bowtie_{u_i} b_i \wedge \bigwedge_{j \in 1..q} \phi_j = a_j \bowtie_{l_j} x_j - y_j \bowtie_{u_j} b_j.$$

The enlargement *of g by δ is the guard $g_{\rhd\delta} = \bigwedge_{i \in 1..m} \phi_{i_{\rhd\delta}} \wedge \bigwedge_{j \in 1..q} \phi_j$*

For every transition $t = (l, g, a, R, l')$ with enlarged guard

$$g_{\rhd\mathcal{P}(a)} = \bigwedge_{i \in 1..m} \phi_i = a_i \bowtie_{l_i} x_i \leq b_i + \mathcal{P}(a) \wedge \bigwedge_{j \in 1..q} \phi_j = a_j \bowtie_{l_j} x_j - y_j \bowtie_{u_j} b_j,$$

we can create a new transition $t_{f,\mathcal{P}} = (l, g_{f,\mathcal{P}}, a, R, \overset{\bullet}{l'})$ called a faulty *transition such that,*
$g_{f,\mathcal{P}} = \bigwedge_{i \in 1..m} \phi_i = b_i \bar{\bowtie}_{l_i} x_i \leq b_i + \mathcal{P}(a) \wedge \bigwedge_{j \in 1..q} \phi_j = a_j \bowtie_{l_j} x_j - y_j \bowtie_{u_j} b_j$ *with* $\bar{\bowtie}_{l_i} \in \{<, \leq\} \setminus \bowtie_{u_i}$

Diagonal constraints remain unchanged under enlargement, as the difference between clocks x and y is preserved by time elapsing, and operator $\bar{\bowtie}_{l_i}$ guarantee that normal and faulty behaviors occur at different dates. From now, we fix a fault model $\mathcal{P}$ and write t_f and g_f instead of $t_{f,\mathcal{P}}$ and $g_{f,\mathcal{P}}$. Clearly, g and g_f are disjoint, and $g \vee g_f$ is equivalent to $g_{\rhd\delta}$. We take this particular definition of enlargement to consider late events as faults. We can easily adapt the definition to handle early events, or any variation where non-specified faulty transitions can be identified through a guard g_f disjoint from g, without harming the results shown in the rest of the paper.

▶ **Definition 5** (Enlargement of automata). *Let $\mathcal{A} = (L, I, X, \Sigma, T, F)$ be a timed automaton. The enlargement of $\mathcal{A}$ by a fault model $\mathcal{P}$ is the automaton $\mathcal{A}_\mathcal{P} = (L_\mathcal{P}, I, X, \Sigma, T_\mathcal{P}, F_\mathcal{P})$, where*

- *$L_\mathcal{P} = L \cup \{\overset{\bullet}{l} \mid l \in L\}$ and $F_\mathcal{P} = F \cup \{\overset{\bullet}{l} \mid l \in F\}$. A location $\overset{\bullet}{l}$ indicates that an unexpected delay has occurred.*
- *$T_\mathcal{P} = T \cup \overset{\bullet}{T}$ such that, $\overset{\bullet}{T} = \{(l, g_f, a, R, \overset{\bullet}{l'}) \mid (l, g, a, R, l') \in T\} \cup \{(\overset{\bullet}{l}, g, a, R, \overset{\bullet}{l'}) \mid (l, g, a, R, l') \in T\}$ i.e., $\overset{\bullet}{T}$ is the set of transitions occurring after a fault.*

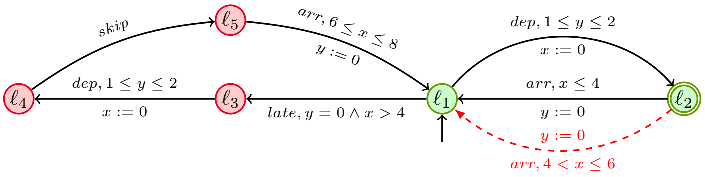


Figure 1 Model of a train system with a mechanism to recover from delays.

A run of $\mathcal{A}_{\mathcal{P}}$ is *faulty* if it contains a transition of $\overset{\bullet}{T}$. It is *just faulty* if its last transition belongs to $\overset{\bullet}{T}$ and all other transitions belong to T. Note that while faulty runs can be finite or infinite, *just faulty* runs are always finite prefix of a faulty run, and end in a location $\overset{\bullet}{l}$.

▶ **Definition 6** (Back To Normal (BTN)). *Let $K \geq 1$, $\mathcal{A}$ be a timed automaton with fault model $\mathcal{P}$. Let $\rho = (l_0, \nu_0) \xrightarrow{(t_1,d_1)} (l_1, \nu_1) \xrightarrow{(t_2,d_2)} \cdots$ be a (finite or infinite) faulty accepting run of $\mathcal{A}_{\mathcal{P}}$, with associated timed word $(a_1, d_1)(a_2, d_2)\ldots$ and let $i \in \mathbb{N}$ be the position of the faulty transition in ρ. Then ρ is* back to normal (BTN) *after K steps if there exists an accepting run $\rho' = (l'_0, \nu'_0) \xrightarrow{(t'_1,d'_1)} (l'_1, \nu'_1) \xrightarrow{(t'_2,d'_2)} \cdots$ of $\mathcal{A}$ with associated timed word $(a'_1, d'_1)(a'_2, d'_2)\ldots$ and an index $\ell \in \mathbb{N}$ such that $(a'_\ell, d'_\ell)(a'_{\ell+1}, d'_{\ell+1}) \cdots = (a_{i+K}, d_{i+K})(a_{i+K+1}, d_{i+K+1}) \cdots$. ρ is* untimed back to normal (untimed BTN) *after K steps if there exists an accepting run $\rho' = (l'_0, \nu'_0) \xrightarrow{(t'_1,d'_1)} (l'_1, \nu'_1) \xrightarrow{(t'_2,d'_2)} \cdots$ of $\mathcal{A}$ and an index $\ell \in \mathbb{N}$ s.t. $a'_\ell a'_{\ell+1} \cdots = a_{i+K} a_{i+K+1} \cdots$*

In other words, if w is a timed word having a faulty accepting run (i.e., $w \in \mathcal{L}(\mathcal{A}_{\mathcal{P}})$), the suffix of w, K steps after the fault, matches with the suffix of some word $w' \in \mathcal{L}(\mathcal{A})$. Note that the accepting run of w' in $\mathcal{A}$ is not faulty, by definition. The conditions in untimed BTN are simpler, and ask the same sequence of actions, but not equality on dates. Words w and w' need not have an identical prefix: this means that a BTN run has returned to *some* normal behavior, but not necessarily *the* behavior originally planned before the fault.

Our current definition of back-to-normal in K steps means that a system recovered from a fault (a primary delay) in $\leq K$ steps and remained error-free. We can generalize our definition, to model real life situations where more than one fault happens due to time delays, but the system recovers from each one in a small number of steps and eventually achieves its fixed goal (a reachability objective, some ω-regular property...). A classical example of this is a metro network, where trains are often delayed, but nevertheless recover from these delays to reach their destination on time. This motivates the following definition of resilience.

▶ **Definition 7** (Resilience). *A timed automaton $\mathcal{A}$ is*

- (untimed) *K-$\forall$-resilient if every finite faulty accepting run is (untimed) BTN in K steps.*
- (untimed) *K-$\exists$-resilient if every just faulty run ρ_{jf} can be extended into a maximal accepting run ρ_f which is (untimed) BTN in K steps.*

Intuitively, a faulty run of $\mathcal{A}$ is BTN if the system has definitively recovered from a fault, i.e., it has recovered and will follow the behavior of the original system after its recovery. The definition of existential resilience considers maximal (infinite, or finite but ending at a location with no outgoing transitions) runs to avoid situations where an accepting faulty run ρ_f is BTN, but all its extensions i.e., suffixes ρ' are such that $\rho_f.\rho'$ *is not* BTN.

▶ **Example 8.** We model train services to a specific destination such as an airport. On an average, the distance between two consecutive stations is covered in ≤ 4 time units. At each stop in a station, the dwell time is in between 1 and 2 time units. To recover from a

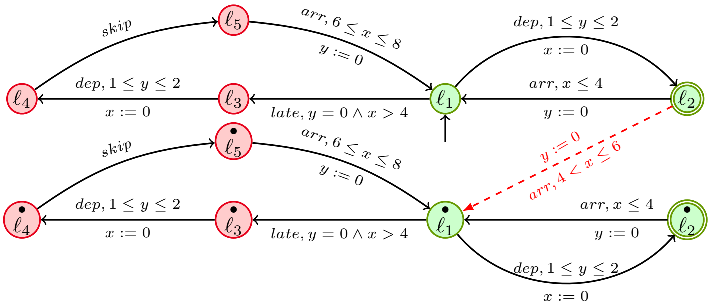


Figure 2 Enlarged automaton for the train system (with recovery) model of Figure 1.

delay, the train is allowed to skip an intermediate station (as long as the next stop is not the destination). Skipping a station is a choice, and can only be activated if there is a delay. We model this system with the timed automaton of Figure 1. There are 5 locations: ℓ_1, and ℓ_2 represent the normal behavior of the train and ℓ_3, ℓ_4, ℓ_5 represent the skipping mechanism. These locations can only be accessed if the faulty transition (represented as a red dotted arrow in Figure 1) is fired. A transition t_{ij} goes from ℓ_i to ℓ_j, and $\overset{\bullet}{t}_{21}$ denotes the faulty transition from ℓ_2 to $\overset{\bullet}{\ell}_1$. The green locations represent the behavior of the train without any delay, and the red locations represent behaviors when the train chooses to skip the next station to recover from a delay. This mechanism is invoked once the train leaves the station where it arrived late (location ℓ_3). When it departs, x is reset as usual; the next arrival to a station (from location ℓ_4) happens after skipping stop at the next station. The delay can be recovered since the running time since the last stop (covering 2 stations) is between 6 and 8 units of time. Formally, verifying that this system can recover from a delay within K steps can be done by setting as fault model $\mathcal{P}(arr) = 2$, and then checking a K-$\exists$-*resilience* problem. It then amounts to asking if the enlarged automaton of Figure 2 can recognize a suffix of a word recognized by the automaton of Figure 1, K steps after visiting location $\overset{\bullet}{\ell}_1$.

Consider the faulty run $\rho_f = (\ell_1, 0|0) \xrightarrow{(t_{12},2)} (\ell_2, 0|2) \xrightarrow{(\overset{\bullet}{t}_{21},8)} (\ell_1, 6|0) \xrightarrow{(t_{13},8)} (\overset{\bullet}{\ell}_3, 6|0) \xrightarrow{(t_{34},10)} (\overset{\bullet}{\ell}_4, 0|2) \xrightarrow{(t_{45},10)} (\overset{\bullet}{\ell}_5, 0|2) \xrightarrow{(t_{51},18)} (\overset{\bullet}{\ell}_1, 8|0) \xrightarrow{(t_{12},19)} (\overset{\bullet}{\ell}_2, 0|1)$ reading $(dep, 2)(arr, 8)(late, 8)(dep, 10)(skip, 10)(arr, 18)(dep, 19)$. Run ρ_f is BTN in 4 steps. It matches the non-faulty run $\rho = (\ell_1, 0|0) \xrightarrow{(t_{12},2)} (\ell_2, 0|2) \xrightarrow{(t_{21},6)} (\ell_1, 4|0) \xrightarrow{(t_{12},8)} (\ell_2, 0|2) \xrightarrow{(t_{21},12)} (\ell_1, 4|0) \xrightarrow{(t_{12},14)} (\ell_2, 0|2) \xrightarrow{(t_{21},18)} (\ell_1, 4|0) \xrightarrow{(t_{12},19)} (\ell_2, 0|1)$ reading $(dep, 2)(arr, 6)(dep, 8)(arr, 12)(dep, 14)(arr, 18)(dep, 19)$. This automaton is K-$\exists$-*resilient* for $K = 4$ and fault model $\mathcal{P}$, as skipping a station after a delay of $\leq$2 time units allows to recover the time lost. It is not K-$\forall$-*resilient*, for any K, as skipping is not mandatory, and a train can be late for an arbitrary number of steps. In Appendix A we give another example that is 1-$\forall$-*resilient*.

K-$\forall$-*resilience* always implies K-$\exists$-*resilience*. In case of K-$\forall$-*resilience*, every faulty run ρ_w has to be BTN in $\leq K$ steps after the occurrence of a fault. This implies K-$\exists$-*resilience* since, any just faulty run ρ_w that is the prefix of an accepting run ρ of $\mathcal{A}_{\mathcal{P}}$ is BTN in less than K steps. The converse does not hold: $\mathcal{A}_{\mathcal{P}}$ can have a pair of runs ρ_1, ρ_2, sharing a common just faulty run ρ_f as prefix such that ρ_1 is BTN in K steps, witnessing existential resilience, while ρ_2 is not. Finally, an accepting run $\rho = \rho_f \rho_s$ in $\mathcal{A}_{\mathcal{P}}$ s.t., ρ_f is *just faulty* and $|\rho_s| < K$, is BTN in K steps since ε is a suffix of a run accepted by $\mathcal{A}$.

4 Existential Resilience

In this section, we consider existential resilience both in the timed and untimed settings.

Existential Timed Resilience. As the first step, we define a product automaton $\mathcal{B} \otimes_K \mathcal{A}$ that recognizes BTN runs. Intuitively, the product synchronizes runs of $\mathcal{B}$ and $\mathcal{A}$ as soon as $\mathcal{B}$ has performed K steps after a fault, and guarantees that actions performed by $\mathcal{A}$ and $\mathcal{B}$ are performed at the same date in the respective runs of $\mathcal{A}$ and $\mathcal{B}$. Before this synchronization, $\mathcal{A}$ and $\mathcal{B}$ take transitions or stay in the same location, but let the same amount of time elapse, guaranteeing that synchronization occurs after runs of $\mathcal{A}$ and $\mathcal{B}$ of identical durations. The only way to ensure this with a timed automaton is to track the global timing from the initial state of both automata $\mathcal{A}$ and $\mathcal{B}$ till K steps after the fault, even though we do not need the timing for individual actions till K steps after the fault.

▶ **Definition 9** (Product). *Let $\mathcal{A} = (L_A, I_A, X_A, \Sigma, T_A, F_A)$ and $\mathcal{B} = (L_B, I_B, X_B, \Sigma, T_B, F_B)$ be two timed automata, where $\mathcal{B}$ contains faulty transitions. Let $K \in \mathbb{N}$ be an integer. Then, the product $\mathcal{B} \otimes_K \mathcal{A}$ is a tuple $(L, I, X_A \cup X_B, (\Sigma \cup \{*\})^2, T, F)$ where $L \subseteq \{L_B \times L_A \times [-1, K]\}$, $F = L_B \times F_A \times [-1, K]$, and initial set of locations $I = I_B \times I_A \times \{-1\}$. Intuitively, -1 means no fault has occurred yet. Then we assign K and decrement to 0 to denote that K steps after fault have passed. The set of transitions T is as follows: We have $\big((l_B, l_A, n), g, < x, y >, R, (l'_B, l'_A, n')\big) \in T$ if and only if either:*

- *$n \neq 0$ (no fault has occurred, or less than K steps of $\mathcal{B}$ have occurred), the action is $< x, y >=< a, * >$, we have transition $t_B = (l_B, g, a, R, l'_B) \in T_B$, $l_A = l'_A$ (the location of $\mathcal{A}$ is unchanged) and either: $n = -1$, the transition t_B is faulty and $n' = K$, or $n = -1$, the transition t_B is non faulty and $n' = -1$, or $n > 0$ and $n' = n - 1$.*
- *$n = n' \neq 0$ (no fault has occurred, or less than K steps of $\mathcal{B}$ have occurred), the action is $< x, y >=< *, a >$, we have the transition $t_A = (l_A, g, a, R, l'_A) \in T_A$, $l_B = l'_B$ (the location of $\mathcal{B}$ is unchanged).*
- *$n = n' = 0$ (at least K steps after a fault have occured), the action is $< x, y >=< a, a >$ and there exists two transitions $t_B = (l_B, g, a, R_B, l'_B) \in T_B$ and $t_A = (l_A, g_A, a, R_A, l'_A) \in T_A$ with $g = g_A \wedge g_B$, and $R = R_B \cup R_A$ (t_A and t_B occur synchronously).*

Runs of $\mathcal{B} \otimes_K \mathcal{A}$ are sequences of the form $\rho^\otimes = (l_0, l_0^A, n_0) \xrightarrow{(t_1, t_1^A), d_1} \cdots \xrightarrow{(t_k, t_k^A), d_k} (l_k, l_k^A, n_k)$ where each $(t_i, t_i^A) \in (T_B \cup \{t_*\}) \times (T_A \cup \{t_*^A\})$ defines uniquely the transition of $\mathcal{B} \otimes_K \mathcal{A}$, where t_* corresponds to the transitions with action $*$. Transitions are of types (t_i, t_*^A) or (t_*, t_i^A) up to a fault and K steps of T_B, and $(t_i, t_i^A) \in T_B \times T_A$ from there on.

For any timed run $\rho^\otimes$ of $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$, the projection of $\rho^\otimes$ on its first component is a timed run ρ of $\mathcal{A}_\mathcal{P}$, that is projecting $\rho^\otimes$ on transitions of $\mathcal{A}_\mathcal{P}$ and remembering only location and clocks of $\mathcal{A}_\mathcal{P}$ in states. In the same way, the projection of $\rho^\otimes$ on its second component is a timed run ρ' of $\mathcal{A}$. Given timed runs ρ of $\mathcal{A}_\mathcal{P}$ and ρ' of $\mathcal{A}$, we denote by $\rho \otimes \rho'$ the timed run (if it exists) of $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$ such that the projection on the first component is ρ and the projection on the second component is ρ'. For $\rho \otimes \rho'$ to exist, we need ρ, ρ' to have the same duration, and for ρ_s the suffix of ρ starting K steps after a fault (if there is a fault and K steps, $\rho_s = \varepsilon$ the empty run otherwise), ρ_s needs to be suffix of ρ' as well.

A run $\rho^\otimes$ of $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$ is accepting if its projection on the second component ($\mathcal{A}$) is accepting (i.e., ends in an accepting state if it is finite and goes through an infinite number of accepting state if it is infinite). We can now relate the product $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$ to BTN runs.

▶ **Proposition 10.** *Let ρ_f be a faulty accepting run of $\mathcal{A}_\mathcal{P}$. The following are equivalent:*

i *ρ_f is BTN in K-steps*

ii *there is an accepting run $\rho^\otimes$ of $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$ s.t., the projection on its first component is ρ_f*

Let ρ be a finite run of $\mathcal{A}_\mathcal{P}$. We denote by $T^{\otimes_K}_\rho$ the set of configurations of $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$ such that there exists a run $\rho^\otimes$ of $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$ ending in this configuration, whose projection on the first component is ρ. We then define $S^{\otimes_K}_\rho$ as the set of states of $\mathcal{R}(\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A})$ corresponding to $T^{\otimes_K}_\rho$, i.e., $S^{\otimes_K}_\rho = \{(s,[\nu]) \in \mathcal{R}(\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}) \mid (s,\nu) \in T^{\otimes_K}_\rho\}$. If we can compute the set $\mathbb{S} = \{S^{\otimes_k}_\rho \mid \rho$ is a finite run of $\mathcal{A}_\mathcal{P}\}$, we would be able to solve *timed* universal resilience, because from this set, one can check existence of a run accepted by $\mathcal{A}_\mathcal{P}$ and not by $\mathcal{A}$. Proposition 18 shows that universal resilience is undecidable. Hence, computing $\mathbb{S}$ is impossible. Roughly speaking, it is because this set depends on the exact timing in a run ρ, and in general one cannot use the region construction.

We can however show that in some restricted cases, we can use a *modified* region construction to build $S^{\otimes_K}_\rho$, which will enable decidability of timed existential resilience. First, we restrict to *just faulty runs*, i.e., consider runs of $\mathcal{A}_\mathcal{P}$ and $\mathcal{A}$ of equal durations, but that did not yet synchronize on actions in the product $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$. For a timed run ρ, by its duration, we mean the time-stamp or date of occurrence of its last event. Second, we consider abstract runs $\widetilde{\sigma}$ through a so-called *strong region automaton*, as defined below. Intuitively, $\widetilde{\sigma}$ keeps more information than in the usual region automaton to ensure that for two timed runs $\rho_1 = (t_1,d_1)(t_2,d_2)\dots$, and $\rho_2 = (t_1,e_1)(t_2,e_2)\dots$ associated with the same run of the strong region automaton, we have $\lfloor e_i \rfloor = \lfloor d_i \rfloor$ for all i. Formally, we build the strong region automaton $\mathcal{R}_{\text{strong}}(\mathcal{B})$ of a timed automaton $\mathcal{B}$ as follows. We add a virtual clock x_ι to $\mathcal{B}$ which is reset at each integral time point, add constraint $x_\iota < 1$ to each transition guard, and add a virtual self loop transition with guard $x_\iota = 1$ resetting x_ι on each state. Standard regions are equivalence classes for clock values, but not for elapsed time. Adding a virtual clock resetting at every integral time point allows to consider the fractional part of elapsed global time in regions. Lemma 12 below shows that if two abstract runs σ_1, σ_2 visit the same sequence of strong regions, then there are two runs of identical duration that have σ_1, σ_2 as abstractions. We then make the usual region construction on this extended timed automaton to obtain $\mathcal{R}_{\text{strong}}(\mathcal{B})$. The strong region construction thus has the same complexity as the standard region construction. Let $\mathcal{L}(\mathcal{R}_{\text{strong}}(\mathcal{B}))$ be the language of this strong region automaton, where these self loops on the virtual clock are projected away. These additional transitions capture ticks at integral times, but do not change the behavior of $\mathcal{B}$, i.e., we have $Unt(\mathcal{L}(\mathcal{B})) \subseteq \mathcal{L}(\mathcal{R}_{\text{strong}}(\mathcal{B})) \subseteq \mathcal{L}(\mathcal{R}(\mathcal{B})) = Unt(\mathcal{L}(\mathcal{B}))$ so $Unt(\mathcal{L}(\mathcal{B})) = \mathcal{L}(\mathcal{R}_{\text{strong}}(\mathcal{B}))$.

For a finite abstract run $\widetilde{\sigma}$ of the *strong* region automaton $\mathcal{R}_{\text{strong}}(\mathcal{A}_\mathcal{P})$, we define the set $S^{\otimes_K}_{\widetilde{\sigma}}$ of states of $\mathcal{R}_{\text{strong}}(\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A})$ (the virtual clock is projected away, and our region is w.r.t original clocks) such that there exists a run $\widetilde{\sigma}^\otimes$ through $\mathcal{R}_{\text{strong}}(\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A})$ ending in this state and whose projection on the first component is $\widetilde{\sigma}$. Let $\widetilde{\sigma}_\rho$ be the run of $\mathcal{R}_{\text{strong}}(\mathcal{A}_\mathcal{P})$ associated with a run ρ of $\mathcal{A}_\mathcal{P}$. It is easy to see that $S^{\otimes_K}_{\widetilde{\sigma}} = \bigcup_{\rho \mid \widetilde{\sigma}_\rho = \widetilde{\sigma}} S^{\otimes_K}_\rho$. For a *just faulty* timed run ρ of $\mathcal{A}_\mathcal{P}$, we have a stronger relation between $S^{\otimes_K}_\rho$ and $S^{\otimes_K}_{\widetilde{\sigma}_\rho}$:

▶ **Proposition 11.** *Let ρ be a* just faulty run *of $\mathcal{A}_\mathcal{P}$. Then $S^{\otimes_K}_\rho = S^{\otimes_K}_{\widetilde{\sigma}_\rho}$.*

Proof. First, notice that given a just faulty timed run ρ of $\mathcal{A}_\mathcal{P}$ and a timed run ρ' of $\mathcal{A}$ of same duration, the timed run $\rho \otimes \rho'$ (the run of $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$ such that ρ is the projection on the first component and ρ' on the second component) exists.

To show that $S^{\otimes_K}_\rho = S^{\otimes_K}_{\widetilde{\sigma}_\rho}$, we show that for any pair of just faulty runs ρ_1, ρ_2 of $\mathcal{A}_\mathcal{P}$ with $\widetilde{\sigma}_{\rho_1} = \widetilde{\sigma}_{\rho_2}$, we have $S^{\otimes_K}_{\rho_1} = S^{\otimes_K}_{\rho_2}$, which yields the result as $S^{\otimes_K}_{\widetilde{\sigma}_\rho} = \bigcup_{\rho' \mid \widetilde{\sigma}_{\rho'} = \widetilde{\sigma}_\rho} S^{\otimes_K}_{\rho'}$. Consider ρ_1, ρ_2, two just faulty timed runs of $\mathcal{A}_\mathcal{P}$ with $\widetilde{\sigma}_{\rho_1} = \widetilde{\sigma}_{\rho_2}$ and let $(l_{\mathcal{A}_\mathcal{P}}, l_\mathcal{A}, K, r) \in S^{\otimes_K}_{\rho_1}$. Then, this implies that there exists $\nu_1 \models r$ and a timed run ρ'_1 of $\mathcal{A}$ with the same duration as ρ_1, such that $\rho_1 \otimes \rho'_1$ ends in state $(l_{\mathcal{A}_\mathcal{P}}, l_\mathcal{A}, K, \nu_1)$. The following lemma completes the proof:

▶ **Lemma 12.** *There exists $\nu_2 \models r$ and a timed run ρ_2' of $\mathcal{A}$ with the same duration as ρ_2, such that $\rho_2 \otimes \rho_2'$ ends in state $(l_{\mathcal{A}_\mathcal{P}}, l_\mathcal{A}, K, \nu_2)$.*

The main idea of the proof is to show that we can construct ρ_2' which will have the same transitions as ρ_1', with same integral parts in timings (thanks to the information from the strong region automaton), but possibly different timings in the fractional parts, called a re-timing of ρ_1'. Notice that ρ_2 *is* a re-timing of ρ_1, as $\widetilde{\sigma}_{\rho_1} = \widetilde{\sigma}_{\rho_2}$. We translate the requirement on ρ_2' into a set of constraints (which is actually a partial ordering) on the fractional parts of the dates of its transitions, and show that we can indeed set the dates accordingly. This translation follows the following idea: the value of a clock x just before firing transition t is obtained by considering the date d of t minus the date d^x of the latest transition t^x at which x has been last reset before t. In particular, the difference $x - y$ between clocks x, y just before firing transition t is $(d - d^x) - (d - d^y) = d^y - d^x$. That is, the value of a clock or its difference can be obtained by considering the difference between two dates of transitions. A constraint given by $x - y \in (n, n+1)$ is equivalent with the constraint given by $d^y - d^x \in (n, n+1)$, and similar constraints on the fractional parts can be given.

Proof. Let $t_1, \ldots, t_n$ be the sequence of transitions of ρ_1, ρ_2 taken respectively, at dates $d_1, \ldots, d_n$ and $e_1, \ldots, e_n$. Similarly, we will denote by $t_1', \ldots, t_k'$ the sequence of transitions of ρ_1', taken at dates $d_1', \ldots, d_k'$. Run ρ_2' will pass by the same transitions $t_1', \ldots, t_k'$, but with possibly different dates $e_1', \ldots, e_k'$ such that:

- the duration of ρ_2' is the same as the duration of ρ_2,
- $\widetilde{\sigma}_{\rho_2'}$ follows the same sequence of states of $\mathcal{R}_{\text{strong}}(\mathcal{A})$ as $\widetilde{\sigma}_{\rho_1'}$ (in particular, ρ_2' is a valid run as it fullfils the guards of its transitions, which are the same as those of ρ_1').
- $\widetilde{\sigma}_{\rho_2 \otimes \rho_2'}$ reaches the same state of $\mathcal{R}_{\text{strong}}(\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A})$ as $\widetilde{\sigma}_{\rho_1 \otimes \rho_1'}$.

We translate these into three requirements on the dates $(e_i')_{i \le k}$ of ρ_2':

R1. We have $e_k' = e_n$,

R2. For every $i \le k$, the integral part $\lfloor e_i' \rfloor = \lfloor d_i' \rfloor$. Remark that we already have $\lfloor e_k' \rfloor = \lfloor e_n \rfloor = \lfloor d_n \rfloor = \lfloor d_k' \rfloor$ by $R1$ and by the hypothesis,

R3. Fractional parts $(\mathsf{frac}(e_i'))_{i \le k}$ satisfy a set of constraints, defined hereafter as a partial ordering on $(\mathsf{frac}(e_i'))_{i \le k} \cup (\mathsf{frac}(e_i))_{i \le n}$.

Notice that the value of a clock x just before firing transition t_i is obtained by considering the date d_i of t_i minus the date d_i^x of the latest transition $t_j, j < i$ at which x has been last reset before i. In particular, the difference $x - y$ between clocks x, y just before firing transition t_i is $(d_i - d_i^x) - (d_i - d_i^y) = d_i^y - d_i^x$. That is, the value of a clock or its difference can be obtained by considering the difference between two dates of transitions. A constraint c given by $x - y \in (n, n+1)$ is equivalent with the constraint $d(c)$ given by $d_i^y - d_i^x \in (n, n+1)$.

We then characterize the conditions required for the run $\rho_2 \otimes \rho_2'$ to reach the same region r of $\mathcal{R}_{\text{strong}}(\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A})$ which was reached by $\rho_1 \otimes \rho_1'$. These conditions are described as on region r in the following equivalent ways:

1. A set of constraints C on the disjoint union $X'' = X_{\mathcal{A}_\mathcal{P}} \uplus X_\mathcal{A}$ of clocks of $\mathcal{A}_\mathcal{P}$ and $\mathcal{A}$, of the form $x - y \in (n, n+1)$ or $x - y = n$ or $x - y > Max$ (possibly considering a null clock y) for $n \in \mathbb{Z}$,
2. The associated set of constraints $C' = \{d(c) \mid c \in C\}$ on $D = \{d_x \mid x \in X_{\mathcal{A}_\mathcal{P}}\} \uplus \{d'_{x'} \mid x' \in X_\mathcal{A}\}$, with d_x the date of the latest transition $t_j^\otimes$ that resets the clock $x \in X_{\mathcal{A}_\mathcal{P}}$, and $d'_{x'}$ the date of the latest transition $t_l^\otimes$ that resets clock $x' \in X_\mathcal{A}$,
3. An ordering $\le'$ over $FP = \{\mathsf{frac}(\tau) \mid \tau \in D\}$ defined as follows: for each constraint $\tau - \tau' \in (n, n+1)$ of C', if $\lfloor \tau \rfloor = \lfloor \tau' \rfloor + n$ then $\mathsf{frac}(\tau) <' \mathsf{frac}(\tau')$, and if $\lfloor \tau \rfloor = \lfloor \tau' \rfloor + n + 1$ then $\mathsf{frac}(\tau') <' \mathsf{frac}(\tau)$.

- For each constraint $\tau - \tau' = n$ of C', then $\mathsf{frac}(\tau') =' \mathsf{frac}(\tau)$.
- For each constraint $\tau - \tau' > \mathsf{c}_{\max}$ of C' such that $\lfloor \tau \rfloor = \lfloor \tau' \rfloor + \mathsf{c}_{\max}$, we have $\mathsf{frac}(\tau') >' \mathsf{frac}(\tau)$ (if $\lfloor \tau \rfloor \geq \lfloor \tau' \rfloor + \mathsf{c}_{\max} + 1$, then we dont need to do anything), where $\mathsf{c}_{\max} = \max(\{c_x \mid x \in X\})$.

Further, path ρ'_2 needs to visit the regions $r_1, \ldots r_k$ visited by ρ'_1. For each i, visiting region r_i is characterized by a set of constraints C_i, which we translate as above as an ordering $\leq'_i$ on $FP' = \{\mathsf{frac}(d'_i) \mid i \leq k\}$.

Thus, finally, we can collect all the requirements for having ρ' with required properties by defining $\leq''$ over $FP' \cup FP$ (notice that it is not a disjoint union) as the transitive closure of the union of all $\leq'_i$ and of $\leq'$. As the union of constraints on C'_i and on C' is satisfied by the dates $(d_i)_{i \leq n}$ and $(d'_i)_{i \leq k}$ of ρ_1 and ρ'_1, the union of constraints is satisfiable. Equivalently, $\leq''$ is a partial ordering, respecting the total natural ordering $\leq$ on $FP \cup FP'$. We will denote $\tau ='' \tau'$ whenever $\tau \leq'' \tau'$ and $\tau' \leq'' \tau$, and $\tau <'' \tau'$ if $\tau \leq'' \tau'$ but we dont have $\tau ='' \tau'$. Because $\leq''$ is a partial ordering, there is no τ, τ' with $\tau <'' \tau' <'' \tau$.

Note that there is only one way of fulfilling the first two requirements R1. and R2; namely by matching e'_k and e_n, and by witnessing dates with the same integral parts in e'_k, e_n as well as d'_k, d_n. While this takes care of the last values, to obtain the remaining values, we can apply any greedy algorithm fixing successively $\mathsf{frac}(e'_{k-1}) \ldots \mathsf{frac}(e'_1)$ and respecting $\leq''$ to yield the desired result. We provide a concrete such algorithm for completeness:

We will start from the fixed value of $\mathsf{frac}(e'_{k-1})$ and work backwards. Let us assume inductively that $\mathsf{frac}(e'_{k-1}) \ldots \mathsf{frac}(e'_{i+1})$ have been fixed. We now describe how to obtain $\mathsf{frac}(e'_i)$. If $\mathsf{frac}(d'_i) ='' \mathsf{frac}(d'_j)$, $j > i$ then we set $\mathsf{frac}(e'_i) = \mathsf{frac}(e'_j)$. If $\mathsf{frac}(d'_i) ='' \mathsf{frac}(d_j)$, then we set $\mathsf{frac}(e'_i) = \mathsf{frac}(e_j)$. Otherwise, consider the sets $L_i = \{\mathsf{frac}(e_j) \mid j \leq n, \mathsf{frac}(d_j) <'' \mathsf{frac}(d'_i)\} \cup \{\mathsf{frac}(e'_j) \mid i < j \leq n, \mathsf{frac}(d'_j) <'' \mathsf{frac}(d'_i)\}$. Also, consider $U_i = \{\mathsf{frac}(e_j) \mid j \leq n, \mathsf{frac}(d_j) >'' \mathsf{frac}(d'_i)\} \cup \{\mathsf{frac}(e'_j) \mid i < j \leq n, \mathsf{frac}(d'_j) >'' \mathsf{frac}(d'_i)\}$. We let $l_i = max(L_i)$ and $u_i = min(U_i)$. We then set $\mathsf{frac}(e'_i)$ to any value in (l_i, u_i). It remains to show that we always have $l_i < u_i$, which will show that such a choice of value for the fractional part of e'_i is indeed possible.

By contradiction, consider that there exists i such that $l_i \geq u_i$, and consider the maximal (first) such i. First, assume that both l_i and u_i are of the form $\mathsf{frac}(e_j), \mathsf{frac}(e_k)$ respectively, i.e. corresponds to clock values in the last regions of ρ_2. The contradiction hypothesis is $l_i = \mathsf{frac}(e_j) \geq u_i = \mathsf{frac}(e_k)$. By definition of L_i and U_i, we also have $\mathsf{frac}(d_j) <'' \mathsf{frac}(d'_i) <'' \mathsf{frac}(d_k)$. In particular, $\mathsf{frac}(d_j) < \mathsf{frac}(d_k)$. This is a contradiction with $\widetilde{\sigma}_{\rho_1} = \widetilde{\sigma}_{\rho_2}$, as the strong region reached by ρ_1 and ρ_2 are the same. A contradiction.

Otherwise, at least one of l_i, u_i is of the form $\mathsf{frac}(e'_j)$, with $j > i$ (consider j minimal if both are of this form). By symetry, let say $l_i = \mathsf{frac}(e'_j) \geq u_i$. Let say $u_i = \mathsf{frac}(e_k)$, as $u_i = \mathsf{frac}(e'_k)$ with $k > j$ is similar since it has been fixed before $\mathsf{frac}(e'_j)$. We have $\mathsf{frac}(d'_j) <'' d'_i <'' \mathsf{frac}(d_k)$ by definition of L_i, U_i. In particular $\mathsf{frac}(d'_j) <'' \mathsf{frac}(d_k)$: That is, $k \in U_j$, and by construction, and as $j > i$, we have $l_i = \mathsf{frac}(e'_j) < \mathsf{frac}(e_k) = u_i$, a contradiction. ◀

Lemma 12 completes the proof of Proposition 11 immediately. Indeed, the lemma implies that $(l_{\mathcal{A}_\mathcal{P}}, l_A, K, r) \in S^{\otimes_K}_{\rho_2}$ from which we infer that $S^{\otimes_K}_{\rho_1} \subseteq S^{\otimes_K}_{\rho_2}$. By a symmetric argument we get the other containment also, and hence we conclude that $S^{\otimes_K}_{\rho_1} = S^{\otimes_K}_{\rho_2}$. ◀

Lemma 12, which is crucial for our decidability results for existential timed resilience, shows that a timed run can be re-timed, i.e., it shows the existence of a timed run with the same transitions but possibly different timestamps. For this, the global time-stamps (d_j) of actions need to be fixed, and in particular the ordering between their fractional parts

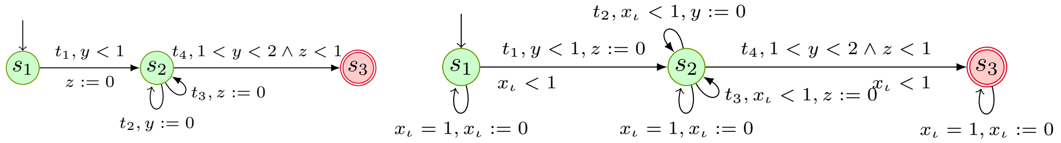

s_1
$t_1, y < 1$
$z := 0$
s_2
$t_4, 1 < y < 2 \wedge z < 1$
s_3
$t_3, z := 0$
$t_2, y := 0$
$t_2, x_\iota < 1, y := 0$
$t_1, y < 1, z := 0$
$x_\iota < 1$
$t_4, 1 < y < 2 \wedge z < 1$
$x_\iota < 1$
$t_3, x_\iota < 1, z := 0$
$x_\iota = 1, x_\iota := 0$
$x_\iota = 1, x_\iota := 0$
$x_\iota = 1, x_\iota := 0$

Figure 3 Example timed automaton (left) and its strong timed automaton (right).

$\mathsf{frac}(d_j)$. The normal region automaton only ensures ordering between the differences of (d_j)'s, but not (d_j) themselves. Let us illustrate this with an concrete example of a TA c.f., Figure 3 (left), having 3 locations s_1, s_2, s_3, 2 clocks y, z and transitions $t_1 = (y < 1, z := 0), t_2 = (y := 0), t_3 = (z := 0), t_4 = (1 < y < 2, z < 1)$ such that t_1 goes from location s_1 to s_2, t_2, t_3 are loops at s_2 and t_4 goes from s_2 to s_3. We can see the run in the standard region automaton $\sigma = (s_1, [\{0\}, \{0\}]) \xrightarrow{t_1} (s_2, [(0,1), \{0\}]) \xrightarrow{t_2} (s_2, [\{0\}, (0,1)]) \xrightarrow{t_3} (s_2, [(0,1), \{0\}]) \xrightarrow{t_4} (s_3, [(1,2), (0,1), \mathsf{frac}(y) < \mathsf{frac}(z)])$. The following two timed runs $\rho_1 = (t_1, d_1 = 0.8)(t_2, d_2 = 1.2)(t_3, d_3 = 1.9)(t_4, d_4 = 2.4)$ and $\rho_2 = (t_1, d'_1 = 0.9)(t_2, d'_2 = 1.89)(t_3, d'_3 = 2.69)(t_4, d'_4 = 3.39)$ correspond to abstract run σ. Note that $\mathsf{frac}(d_2) < \mathsf{frac}(d_3)$ but $\mathsf{frac}(d'_2) > \mathsf{frac}(d'_3)$.

We build the strong region automaton by adding a virtual clock x_ι reset at all integer points (reset x when $x_\iota = 1$) c.f., Figure 3 (right). As explained above, concrete runs ρ_1 and ρ_2 have the same abstract run σ in the standard region automaton. Now, if we consider abstract runs in the strong region automaton (i.e. with the addition of a clock x_ι reset at integral time points), the concrete run ρ_1 will correspond to abstract run $\sigma_1 = (s_1, [\{0\}, \{0\}, \{0\}]) \xrightarrow{t_1} (s_2, [(0,1), (0,1), \{0\}, \mathsf{frac}(x_\iota) = \mathsf{frac}(y)]) \xrightarrow{t_2} (s_2, [(0,1), \{0\}, (0,1), \mathsf{frac}(x_\iota) < \mathsf{frac}(z)]) \xrightarrow{t_3} (s_2, [(0,1), (0,1), \{0\}, \mathsf{frac}(y) < \mathsf{frac}(x_\iota)]) \xrightarrow{t_4} (s_3, [(0,1), (1,2), (0,1), \mathsf{frac}(y) < \mathsf{frac}(x_\iota) < \mathsf{frac}(z)])$, and the concrete run ρ_2 will correspond to abstract run $\sigma_2 = (s_1, [\{0\}, \{0\}, \{0\}]) \xrightarrow{t_1} (s_2, [(0,1), (0,1), \{0\}, \mathsf{frac}(x_\iota) = \mathsf{frac}(y)]) \xrightarrow{t_2} (s_2, [(0,1), \{0\}, (0,1), \mathsf{frac}(x_\iota) < \mathsf{frac}(z)]) \xrightarrow{t_3} (s_2, [(0,1), (0,1), \{0\}, \mathsf{frac}(x_\iota) < \mathsf{frac}(y)]) \xrightarrow{t_4} (s_3, [(0,1), (1,2), (0,1), \mathsf{frac}(x_\iota) < \mathsf{frac}(y) < \mathsf{frac}(z)])$. The abstract run σ_1 ends with a relation $\mathsf{frac}(y) < \mathsf{frac}(x_\iota) < \mathsf{frac}(z)$ on fractional parts of clocks x_ι, y, z, the abstract runs σ_2 end with the relation $\mathsf{frac}(x_\iota) < \mathsf{frac}(y) < \mathsf{frac}(z)$. Thus, ρ_1 and ρ_2, do not have the same abstract "strong" run.

Algorithm to solve Existential Timed Resilience. We can now consider existential timed resilience, and prove that it is decidable thanks to Propositions 10 and 11. The main idea is to reduce the existential resilience question to a question on the sets of regions reachable after just faulty runs. Indeed, focusing on just faulty runs means that we do not have any actions to match, only the duration of the run till the fault, whereas if we had tried to reason on faulty runs in general, actions have to be synchronized K steps after the fault and then we cannot compute the set of $S_{\rho_f}^{\otimes_K}$. We can show that reasoning on $S_{\rho_f}^{\otimes_K}$ for just faulty runs is sufficient. Let ρ_f be a just faulty timed run of $\mathcal{A}_\mathcal{P}$. We say that $s \in S_{\rho_f}^{\otimes_K}$ is *safe* if there exists a (finite or infinite) maximal accepting run of $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$ from s, and that $S_{\rho_f}^{\otimes_K}$ is safe if there exists $s \in S_{\rho_f}^{\otimes_K}$ which is safe.

▶ **Lemma 13.** *There exists a maximal accepting extension of a just faulty run ρ_f that is BTN in K-steps iff $S_{\rho_f}^{\otimes_K}$ is safe. Further, deciding if $S_{\rho_f}^{\otimes_K}$ is safe can be done in PSPACE.*

Proof. Let ρ_f a just faulty run. By Proposition 10, there exists an extention ρ of ρ_f that is BTN in K steps if and only if there exists an accepting run $\rho^{\otimes_K}$ of $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$ such that ρ_f is a prefix of the projection of $\rho^{\otimes_K}$ on its first component, if and only if there exists a just faulty run $\rho_f^{\otimes_K}$ of $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$ such that its projection on the first component is ρ_f, and such that an accepting state of $\mathcal{A}_\mathcal{P} \otimes_K \mathcal{A}$ can be reached after $\rho_f^{\otimes_K}$, if and only if $S_{\rho_f}^{\otimes_K}$ is safe.

Safety of $S^{\otimes_K}_{\rho_f}$ can be verified using a construction similar to the one in Theorem 16: it is hence a reachability question in a region automaton, solvable with a PSPACE complexity. ◀

This lemma means that it suffices to consider the set of $S^{\otimes_K}_{\rho_f}$ over all ρ_f just faulty, which we can compute using region automaton thanks to Prop. 11, which gives:

▶ **Theorem 14.** *K-$\exists$-resilience of timed automata is in EXPSPACE.*

Proof. Lemma 13 implies that $\mathcal{A}$ is not K-timed existential resilient if and only if there exists a just faulty run ρ_f such that $S^{\otimes_K}_{\rho_f}$ is not safe. This latter condition can be checked. Let us denote by $\mathcal{R}_{\text{strong}}(\mathcal{A}_{\mathcal{P}}) = (S_{\mathcal{R}(\mathcal{A}_{\mathcal{P}})}, I_{\mathcal{R}(\mathcal{A}_{\mathcal{P}})}, \Sigma, T_{\mathcal{R}(\mathcal{A}_{\mathcal{P}})}, F_{\mathcal{R}(\mathcal{A}_{\mathcal{P}})})$ the strong region automaton associated with $\mathcal{A}_{\mathcal{P}}$. We also denote $\mathcal{R}_{\otimes_K} = (S_{\mathcal{R}_{\otimes_K}}, I_{\mathcal{R}_{\otimes_K}}, \Sigma, T_{\mathcal{R}_{\otimes_K}}, F_{\mathcal{R}_{\otimes_K}})$ the strong region automaton $\mathcal{R}_{\text{strong}}(\mathcal{A}_{\mathcal{P}} \otimes_K \mathcal{A})$. Let ρ_f be a just faulty run, and let $\sigma = \widetilde{\sigma}_{\rho_f}$ denote the run of $\mathcal{R}_{\text{strong}}(\mathcal{A}_{\mathcal{P}})$ associated with ρ_f. Thanks to Proposition 11, we have $S^{\otimes_K}_{\rho_f} = S^{\otimes_K}_{\sigma}$, as $S^{\otimes_K}_{\rho_f}$ does not depend on the exact dates in ρ_f, but only on their regions, i.e., on σ. So it suffices to find a reachable witness $S^{\otimes_K}_{\sigma}$ of $\mathcal{R}_{\otimes_K}$ which is not safe, to conclude that $\mathcal{A}$ is not existentially resilient. For that, we build an (untimed) automaton $\mathfrak{B}$. Intuitively, this automaton follows σ up to a fault of the region automaton $\mathcal{R}_{\text{strong}}(\mathcal{A}_{\mathcal{P}})$, and maintains the set $S^{\otimes_K}_{\sigma}$ of regions of $\mathcal{R}_{\otimes_K}$. This automaton stops in an accepting state immediately after occurrence of a fault. Formally, the product subset automaton $\mathfrak{B}$ is a tuple $(S_{\mathfrak{B}}, I, \Sigma, T, F)$ with set of states $S_{\mathfrak{B}} = S_{\mathcal{R}_{\text{strong}}(\mathcal{A}_{\mathcal{P}})} \times 2^{S_{\mathcal{R}_{\otimes_K}}} \times \{0,1\}$, set of initial states $I = I_{\mathcal{R}_{\text{strong}}(\mathcal{A}_{\mathcal{P}})} \times \{I_{\mathcal{R}_{\otimes_K}}\} \times \{0\}$, and set of final states $F = S_{\mathcal{R}_{\text{strong}}(\mathcal{A}_{\mathcal{P}})} \times 2^{S_{\mathcal{R}_{\otimes_K}}} \times \{1\}$. The set of transitions $T \subseteq S_{\mathfrak{B}} \times \Sigma \times S_{\mathfrak{B}}$ is defined as follows,

- $\big((l,r,S,0),a,(l',r',S',\flat)\big) \in T$ if and only if $t_R = \big((l,r),a,(l',r')\big) \in T_{\mathcal{R}_{\text{strong}}(\mathcal{A}_{\mathcal{P}})}$ and $\flat = 1$ if and only if t_R is faulty and $\flat = 0$ otherwise.
- S' is the set of states s' of $\mathcal{R}_{\text{strong}}(\mathcal{A}_{\mathcal{P}} \otimes_K \mathcal{A})$ whose first component is (l',r') and such that there exists $s \in S, (s,a,s') \in T_{\mathcal{R}(\otimes_K)}$.

Intuitively, 0 in the states means no fault has occurred yet, and 1 means that a fault has just occurred, and thus no transition exists from this state. We have that for every prefix σ of a just faulty abstract run of $\mathcal{R}_{\text{strong}}(\mathcal{A}_{\mathcal{P}})$, ending on a state (l,r) of $\mathcal{R}_{\text{strong}}(\mathcal{A}_{\mathcal{P}})$ then, there exists a unique accepting path $\sigma^{\otimes}$ in $\mathfrak{B}$ such that σ is the projection of $\sigma^{\otimes}$ on its first component. Let $(l,r,S,1)$ be the state reached by $\sigma^{\otimes}$. Then $S^{\otimes_K}_{\sigma} = S$. Thus, non-existential resilience can be decided by checking reachability of a state $(l,r,S,1)$ such that S is not safe in automaton $\mathfrak{B}$. Recall (from Lemma 13) that checking safety of S is in PSPACE. As $\mathfrak{B}$ is of doubly exponential size, reachability can be checked in EXPSPACE. As EXPSPACE is closed under complement, checking existential resilience is in EXPSPACE. ◀

While we do not have a matching lower bound, we complete this subsection with following (easy) hardness result (we leave the details in Appendix B due to lack of space).

▶ **Theorem 15.** *The K-$\exists$-resilience problem for timed automata is PSPACE-Hard.*

Proof. We proceed by reduction from the language emptiness problem, which is known to be PSPACE-Complete for timed automata. We can reuse the gadget $\mathcal{G}_{und}$ of Figure 4. We take any automaton $\mathcal{A}$ and collapse its initial state to state s_1 in the gadget. We recall that s_1 is accessible at date 15 only after a fault. We add a self loop with transition, $t_e = (s_2, \sigma, true, \emptyset, s_2)$ for every $\sigma \in \Sigma$. This means that after reaching s_2, which is accessible only at date 15 if no fault has occurred, the automaton accepts any letter with any timing. Then, if $\mathcal{A}$ has no accepting word, there is no timed word after a fault which is a suffix of a word in $\mathcal{L}(\mathcal{A})$, and conversely, if $\mathcal{L}(\mathcal{A}) \neq \emptyset$, then any word recognized from s_1 is also recognized from q_e. So the language emptiness problem reduces to 2-$\exists$-*resilience*. ◀

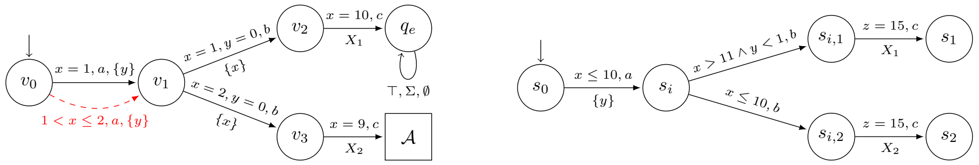


Figure 4 The gadget automaton $\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}$ (left) and the gadget $\mathcal{G}_{und}$ (right).

Existential Untimed Resilience. We next address untimed existential resilience, which we show can be solved by enumerating states (l, r) of $\mathcal{R}(A)$ reachable after a fault, and for each of them proving existence of a BTN run starting from (l, r). This enumeration and the following check uses polynomial space, yielding PSPACE-Completeness of K-$\exists$-*resilience*.

▶ **Theorem 16.** *Untimed K-$\exists$-resilience is PSPACE-Complete.*

Proof (sketch). *Membership:* $\mathcal{A}$ is untimed K-$\exists$-*resilient* if and only if for all states $q = (l, r)$ reached by a just faulty run of $\mathcal{R}(\mathcal{A}_\mathcal{P})$, there exists a maximal accepting path σ from q such that its suffix σ_s after K steps is also the suffix of a path of $\mathcal{R}(\mathcal{A})$. This property can be verified in PSPACE. A detailed proof is provided in Appendix B.

Hardness: We can now show that untimed K-$\exists$-*resilience* is PSPACE-Hard. Consider a timed automaton $\mathcal{A}$ with alphabet Σ and the construction of an automata that uses a gadget shown in Figure 4 (left). Let us call this automaton $\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}$. This automaton reads a word $(a, 1)(b, 1)(c, 11)$ and then accepts all timed words 2 steps after a fault, via Σ loop on a particular accepting state q_e. If $\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}$ takes the faulty transition (marked in dotted red) then it resets all clocks of $\mathcal{A}$ and behaves as $\mathcal{A}$. The accepting states are $q_e \cup F$. Then, $\mathcal{A}$ has an accepting word if and only if $\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}$ is untimed 2-$\exists$-*resilient*. Since the emptiness problem for timed automata is PSPACE-Complete, the result follows. ◀

▶ Remark 17. The hardness reduction in the proof of Theorem 16 holds even for deterministic timed automata. It is known [2] that PSPACE-Hardness of emptiness still holds for deterministic TAs. Hence, considering deterministic timed automata will not improve the complexity of K-$\exists$-*resilience*. Considering IRTAs does not change complexity either, as the gadget used in Theorem 16 can be adapted to become an IRTA (as shown in Appendix C).

5 Universal Resilience

In this section, we consider the problem of universal resilience and show that it is very close to the language inclusion question in timed automata, albeit with a few subtle differences. One needs to consider timed automata with ε-transitions [11], which are strictly more expressive than timed automata. First, we show a reduction from the language inclusion problem.

▶ **Proposition 18.** *Language inclusion for timed automata can be reduced in polynomial time to K-$\forall$-resilience. Thus, K-$\forall$-resilience is undecidable in general for timed automata.*

Proof. Let $\mathcal{A}_1 = (L_1, \{l_{0_1}\}, X_1, \Sigma_1, T_1, F_1)$ and $\mathcal{A}_2 = (L_2, \{l_{0_2}\}, X_2, \Sigma_2, T_2, F_2)$ be two timed automata with only one initial state (w.l.o.g). We build a timed automaton $\mathcal{B}$ such that $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$ if and only if $\mathcal{B}$ is 2-$\forall$-*resilient*.

We first define a gadget $\mathcal{G}_{und}$ that allows to reach a state s_1 at an arbitrary date $d_1 = 15$ when a fault happens, and a state s_2 at date $d_2 = d_1 = 15$ when no fault occur. This gadget is shown in Fig 4(right). $\mathcal{G}_{und}$ has 6 locations $s_0, s_i, s_{i,1}, s_1, s_2 \notin L_1 \cup L_2$, three new clocks $x, y, z \notin X_1 \cup X_2$, three new actions $a, b, c \notin \Sigma_1 \cup \Sigma_2$, and 5 transitions $t_0, t_1, t_2, t_3, t_4 \notin$

$T_1 \cup T_2$ defined as: $t_0 = (s_0, a, g_0, \{y\}, s_i)$ with $g_0 ::= x \leq 10$, $t_1 = (s_i, b, g_1, \emptyset, s_{i,1})$ with $g_1 ::= x > 11 \wedge y < 1$, $t_2 = (s_i, b, g_2, \emptyset, s_{i,2})$ with $g_2 ::= x \leq 10$, $t_3 = (s_{i,1}, c, g_3, X_1, s_1)$ with $g_3 ::= z = 15$, and $t_4 = (s_{i,2}, c, g_4, X_2, s_2)$ with $g_4 ::= z = 15$. Clearly, in this gadget, transition t_1 can never fire, as a configuration with $x > 11$ and $y < 1$ is not accessible.

We build a timed automaton $\mathcal{B}$ that contains all transitions of $\mathcal{A}_1$ and $\mathcal{A}_2$, but preceded by $\mathcal{G}_{und}$ by collapsing the initial location of $\mathcal{A}_1$ i.e., l_{0_1} with s_1 and the initial location of $\mathcal{A}_2$ i.e., l_{0_2} with s_2. We also use a fault model $\mathcal{P} : a \to [0, 2]$, that can delay transitions t_0 with action a by up to 2 time units. The language $\mathcal{L}(\mathcal{B})$ is the set of words:

$$\mathcal{L}(\mathcal{B}) = \{\ (a, d_1)(b, d_2)(c, 15)(\sigma_1, d_3) \dots (\sigma_n, d_{n+2}) \mid (d_1 \leq 10) \wedge (d_2 \leq 10) \wedge (d_2 - d_1 < 1)$$
$$\wedge \exists w = (\sigma_1, d'_3) \dots (\sigma_n, d'_{n+2}) \in \mathcal{L}(\mathcal{A}_2), \forall i \in 3..n+2, d'_i = d_i - 15\}$$

The enlargement of $\mathcal{B}$ is denoted by $\mathcal{B}_{\mathcal{P}}$. The words in $\mathcal{L}(\mathcal{B}_{\mathcal{P}})$ is the set of words in $\mathcal{L}(\mathcal{B})$ (when there is no fault) plus the set of words in:

$$\mathcal{L}^F(\mathcal{B}_{\mathcal{P}}) = \{(a, d_1)(b, d_2)(c, 15)(\sigma_1, d_3) \dots (\sigma_n, d_{n+2}) \mid (10 < d_1 \leq 12) \wedge d_2 > 11$$
$$\wedge (d_2 - d_1 < 1) \wedge \exists w = (\sigma_1, d'_3) \dots (\sigma_n, d'_{n+2}) \in \mathcal{L}(\mathcal{A}_1), \forall i \in 3..n+2, d'_i = d_i - 15\}$$

Now, $\mathcal{B}$ is K-$\forall$-*resilient* for $K = 2$ if and only if every word in $\mathcal{L}^F(\mathcal{B}_{\mathcal{P}})$ is BTN after 2 steps ($K = 2$), i.e., for every word $w = (a, d_1)(b, d_2)(c, 15)(\sigma_1, d_3) \dots (\sigma_n, d_{n+2})$ in $\mathcal{L}^F(\mathcal{B}_{\mathcal{P}})$, if there exists a word $w = (a, d'_1)(b, d'_2)(c, 15)(\sigma_1, d_3) \dots (\sigma_n, d_{n+2})$ in $\mathcal{L}(\mathcal{B})$. This means that every word of $\mathcal{A}_1$ is a word of $\mathcal{A}_2$. So $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$ if and only if $\mathcal{B}$ is 2-$\forall$-*resilient*.

As language inclusion for timed automata is undecidable [2], an immediate consequence is that K-$\forall$-*resilience* of timed automata is undecidable. ◀

Next we show that the reduction is also possible in the reverse direction.

▶ **Proposition 19.** *K-$\forall$-resilience can be reduced in polynomial time to language inclusion for timed automata with ε-transitions.*

Proof. Given a timed automaton $\mathcal{A} = (L, I, X, \Sigma, T, F)$, we can build a timed automaton $\mathcal{A}^S$ that recognizes all suffixes of timed words recognized by $\mathcal{A}$ (see Appendix B, Figure 7 for an example). Formally, $\mathcal{A}^S$ contains the original locations and transitions of $\mathcal{A}$, a copy of all location, a copy of all transitions where letters are replaced by ε, and a transition from copies to original locations labeled by their original letters.

We have $\mathcal{A}^S = (L^S, I^S, X, \Sigma \cup \{\varepsilon\}, T^S, F)$, where $L^S = L \cup \{l' \mid l \in L\}$, $I^S = \{l' \in L_S, l \in I\}$ $T^S = T \cup \{(l'_1, g, \varepsilon, R, l'_2) \mid \exists (l_1, g, \sigma, R, l_2) \in T\} \cup \{(l'_1, g, \sigma, R, l_2) \mid \exists (l_1, g, \sigma, R, l_2) \in T\}$. Obviously, for every timed word $(a_1, d_1)(a_2, d_2) \dots (a_n, d_n)$ recognized by $\mathcal{A}$, and every index $k \in 1..n$, the words $(\varepsilon, d_1)(\varepsilon, d_k)(a_{k+1}, d_{k+1}) \dots (a_n, d_n) = (a_{k+1}, d_{k+1}) \dots (a_n, d_n)$ is recognized by $\mathcal{A}^S$.

Given a timed automaton $\mathcal{A}$ and a fault model $\mathcal{P}$, we build an automaton $\mathcal{B}^{\mathcal{P}}$ which remembers if a fault has occurred, and how many transitions have been taken since a fault (see Definition 9 in Appendix B). Then, we can build an automaton $\mathcal{B}^{\mathcal{P},\varepsilon}$ by re-labeling every transition occurring before a fault and until K steps after the fault by ε, keeping the same locations, guards and resets, and leave transitions occurring more than K steps after a fault unchanged. The relabeled transitions are transitions starting from a location (l, n) with $n \neq 0$. Accepting locations of $\mathcal{B}^{\mathcal{P},\varepsilon}$ are of the form $(l, 0)$ where l is an accepting locations of $\mathcal{A}$ occurring after a fault in $\mathcal{B}^{\mathcal{P}}$. Then, every faulty run accepted by $\mathcal{B}^{\mathcal{P},\varepsilon}$ is associated with a word of the form $\rho = (t_1, d_1) \dots (t_f, d_f)(t_{f+1}, d_{f+1}) \dots (t_{f+K}, d_{f+K}) \dots . (t_n, d_n)$ where $t_1, \dots t_{f+K}$ are ε transitions. A run ρ is BTN if and only if $(a_{f+K+1}, d_{f+K+1}) \dots (a_n, d_n)$ is a suffix of a timed word of $\mathcal{A}$, i.e., is recognized by $\mathcal{A}^S$.

Now one can check that every word in $\mathcal{B}^{\mathcal{P},\varepsilon}$ (reading only ε before that fault) is recognized by the suffix automaton $\mathcal{A}^S$, i.e. solve a language inclusion problem for timed automata with ε transitions. ◀

We note that ε-transitions are critical for the reduction of Proposition 19. To get decidability of K-$\forall$-*resilience*, it is thus necessary (but not sufficient) to be in a class with decidable timed language inclusion, such as Event-Recording timed automata [3], Integer Reset timed automata (IRTA) [18], or Strongly Non-Zeno timed automata [4]. However, to obtain decidability of K-$\forall$-*resilience* using Proposition 19, one needs also to ensure that inclusion is still decidable for automata in the presence of ε transitions. When a subclass C of timed automata is closed by enlargement (due to the fault model), and if timed language inclusion is decidable, even with ε transitions, then Proposition 19 implies that K-$\forall$-*resilience* is decidable for C. We show that this holds for the case of IRTA and leave other subclasses for future work. For IRTA [18], we know that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ is decidable in EXPSPACE when $\mathcal{B}$ is an IRTA [18] (even with ε transitions), from which we obtain an upper bound for K-$\forall$-*resilience* of IRTA. The enlargement of guards due to the fault can add transitions that reset clocks at non-integral times, but it turns out that the suffix automaton $\mathcal{A}^S$ of Proposition 19 is still an IRTA. A matching lower bound is obtained by encoding inclusion for IRTA with K-$\forall$-*resilience* using a trick to replace the gadget in Proposition 18 by an equivalent IRTA. Thus, we have Theorem 20 (proof in Appendix C).

▶ **Theorem 20.** *K-$\forall$-resilience is EXPSPACE-Complete for IRTA.*

Finally, we conclude this section by remarking that universal *untimed* resilience is decidable for timed automata in general, using the reductions of Propositions 18 and 19:

▶ **Theorem 21.** *Untimed K-$\forall$-resilience is EXPSPACE-Complete.*

Proof. Recall that untimed language inclusion of timed automata is EXPSPACE-Complete [9]. The lower bound is readily obtained by using the reduction of Proposition 18.

For the upper bound, we will use the construction of automata $\mathcal{A}^S$ and $\mathcal{B}^{\mathcal{P},\varepsilon}$ built during the reduction of Proposition 19. We however need inclusion of TA with ε transitions, and thus we adapt the EXPSPACE algorithm in the presence of ε transitions:

We can consider ε transitions as transitions labeled by any letter, and build the region automata $\mathcal{A}_\sharp = \mathcal{R}(\mathcal{A}^S)$ and $\mathcal{B}_\sharp = \mathcal{R}(\mathcal{B}^{\mathcal{P},\varepsilon})$. The size of these untimed automata is exponential in the number of clocks, with ε transitions. We can perform an ε reduction on $\mathcal{A}_\sharp$ to obtain an automaton $\mathcal{A}_U^S$ with the same number of states as $\mathcal{A}_\sharp$ that recognizes untimed suffixes of words of $\mathcal{A}$. Similarly, we can perform an ε reduction on $\mathcal{B}_\sharp$ to obtain an automaton $\mathcal{B}_U^{\mathcal{P}}$ with the same number of states as $\mathcal{B}_\sharp$ that recognizes suffixes of words played K steps after a fault. We then check $\mathcal{L}(\mathcal{B}_U^{\mathcal{P}}) \subseteq \mathcal{L}(\mathcal{A}_U^S)$ with an usual PSPACE inclusion algorithm, which yields the EXPSPACE upper bound, as $\mathcal{A}_U^S, \mathcal{B}_U^{\mathcal{P}}$ have an exponential number of states w.r.t. $|\mathcal{A}|$. ◀

6 Conclusion

Resilience allows to check robustness of a timed system to unspecified delays. A universally resilient timed system recovers from any delay in some fixed number of steps. Existential resilience guarantees the existence of a controller that can bring back the system to a normal behavior within a fixed number of steps after an unexpected delay. Interestingly, we show that existential resilience enjoys better complexities/decidability than universal resilience. Universal resilience is decidable only for well behaved classes of timed automata such as IRTA, or in the untimed setting. A future work is to investigate resilience for other determinizable classes of timed automata, and a natural extension of resilience called *continuous resilience*, where a system recovers within some fixed duration rather than within some number of steps. Another natural question is to consider resilience questions when K is not fixed, i.e., check existence of a value for K such that $\mathcal{A}$ is K-$\exists$-*resilient* (resp. K-$\forall$-*resilient*).

References

1 S. Akshay, B. Bollig, P. Gastin, M. Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. *Fundam. Informaticae*, 130(4):377–407, 2014.

2 R. Alur and D.L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

3 R. Alur, L. Fix, and T.A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999.

4 C. Baier, N. Bertrand, P. Bouyer, and T. Brihaye. When are timed automata determinizable? In *Proc. of ICALP'09*, volume 5556 of *LNCS*, pages 43–54, 2009.

5 Roderick Bloem, Peter Gjøl Jensen, Bettina Könighofer, Kim Guldstrand Larsen, Florian Lorber, and Alexander Palmisano. It's time to play safe: Shield synthesis for timed systems. *CoRR*, abs/2006.16688, 2020. arXiv:2006.16688.

6 P. Bouyer, F. Chevalier, and D. D'Souza. Fault diagnosis using timed automata. In *Proc. of FOSSACS 2005*, pages 219–233, 2005.

7 P. Bouyer, N. Markey, and O. Sankur. Robust model-checking of timed automata via pumping in channel machines. In *Proc. of FORMATS 2011*, volume 6919 of *LNCS*, pages 97–112, 2011.

8 P. Bouyer, N. Markey, and O. Sankur. Robustness in timed automata. In *International Workshop on Reachability Problems*, volume 8169 of *LNCS*, pages 1–18, 2013.

9 R. Brenguier, S. Göller, and O. Sankur. A comparison of succinctly represented finite-state systems. In *Proc. of CONCUR 2012*, volume 7454 of *LNCS*, pages 147–161. Springer, 2012.

10 M. De Wulf, L. Doyen, N. Markey, and J-F Raskin. Robust safety of timed automata. *Formal Methods Syst. Des.*, 33(1-3):45–84, 2008.

11 V. Diekert, P. Gastin, and A. Petit. Removing epsilon-transitions in timed automata. In *Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '97, pages 583–594, Berlin, Heidelberg, 1997. Springer-Verlag.

12 Catalin Dima. Dynamical properties of timed automata revisited. In *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings*, volume 4763 of *Lecture Notes in Computer Science*, pages 130–146. Springer, 2007.

13 D. D'Souza, M. Gopinathan, S. Ramesh, and P. Sampath. Conflict-tolerant real-time features. In *Fifth International Conference on the Quantitative Evaluaiton of Systems (QEST 2008)*, pages 274–283. IEEE Computer Society, 2008.

14 Rüdiger Ehlers and Ufuk Topcu. Resilience to intermittent assumption violations in reactive synthesis. In Martin Fränzle and John Lygeros, editors, *17th International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'14, Berlin, Germany, April 15-17, 2014*, pages 203–212. ACM, 2014.

15 V. Gupta, T.A. Henzinger, and R. Jagadeesan. Robust timed automata. In *Proc. Of HART'97, Hybrid and Real-Time Systems*, volume 1201 of *LNCS*, pages 331–345, 1997.

16 A. Puri. Dynamical properties of timed automata. *In DEDS*, 10(1-2):87–113, 2000.

17 Matthieu Renard, Yliès Falcone, Antoine Rollet, Thierry Jéron, and Hervé Marchand. Optimal enforcement of (timed) properties with uncontrollable events. *Math. Struct. Comput. Sci.*, 29(1):169–214, 2019.

18 P. V. Suman, P.K. Pandya, S.N. Krishna, and L. Manasa. Timed automata with integer resets: Language inclusion and expressiveness. In *Proc. of FORMATS'08*, volume 5215 of *LNCS*, pages 78–92, 2008.

19 M. Swaminathan, M. Fränzle, and J-P. Katoen. The surprising robustness of (closed) timed automata against clock-drift. In *Fifth IFIP International Conference On Theoretical Computer Science - TCS 2008, IFIP 20th World Computer Congress, TC 1, Foundations of Computer Science, September 7-10, 2008, Milano, Italy*, volume 273 of *IFIP*, pages 537–553. Springer, 2008.

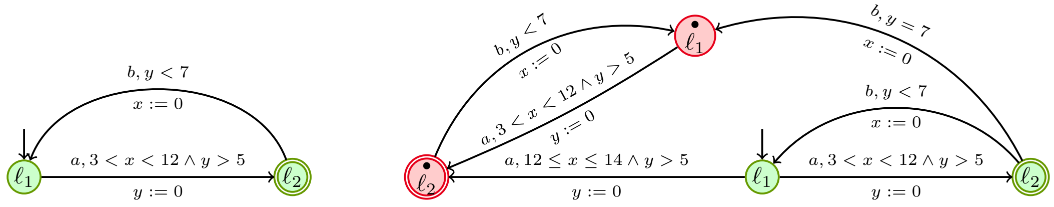


Figure 5 $\mathcal{A}$ on the left; Enlargement $\mathcal{A}_{\mathcal{P}}$ on the right, $\mathcal{P}(a) = 2, \mathcal{P}(b) = 0$.

A Example for Universal Resilience

▶ **Example 22.** Consider the automaton $\mathcal{A}$ in Figure 5, with two locations ℓ_1 and ℓ_2, a transition t_{12} from ℓ_1 to ℓ_2 and a transition t_{21} from ℓ_2 to ℓ_1. The enlarged automaton $\mathcal{A}_{\mathcal{P}}$ has two extra locations $\dot{\ell}_1, \dot{\ell}_2$, extra transitions between $\dot{\ell}_1$ and $\dot{\ell}_2$, and from ℓ_1 to $\dot{\ell}_2$ and from ℓ_2 to $\dot{\ell}_1$ respectively. We represent a configuration of the automata with a pair $(\ell, \nu(x)|\nu(y))$ where, ℓ belongs to the set of the locations and $\nu(x)$ (resp. $\nu(y)$) represents the valuation of clock x (resp. clock y). Let $\rho_f = (\ell_1, 0|0) \xrightarrow{(t_{12},6)} (\ell_2, 6|0) \xrightarrow{(\dot{t}_{21},13)} (\dot{\ell}_1, 0|7) \xrightarrow{(\ddot{t}_{12},19)} (\dot{\ell}_2, 4|0)$ be a *faulty run* reading the faulty word $(a, 6)(b, 13)(a, 19) \in \mathcal{L}(\mathcal{A}_{\mathcal{P}})$. This run is 1-BTN since the run $\sigma = (\ell, 0|0) \xrightarrow{(t_{12},6)} (\ell_2, 6|0) \xrightarrow{(t_{21},12)} (\ell_1, 0|6) \xrightarrow{(t_{12},19)} (\ell_2, 7|0)$ is an accepting run of $\mathcal{A}$, reading timed word $w_\sigma = (a, 6)(b, 12)(a, 19) \in \mathcal{L}(\mathcal{A})$. Similarly, the run $\rho' = (\ell, 0|0) \xrightarrow{(\dot{t}_{12},14)} (\dot{\ell}_2, 14|0) \xrightarrow{(\ddot{t}_{21},20)} (\dot{\ell}_1, 0|6) \xrightarrow{(\ddot{t}_{12},31)} (\dot{\ell}_2, 11|0)$ of $\mathcal{A}_{\mathcal{P}}$ reading word $(a, 14)(b, 20)(a, 31)$ is 1-BTN because of run $\sigma' = (\ell_1, 0|0) \xrightarrow{(t_{12},10)} (\ell_2, 10|0) \xrightarrow{(t_{21},15)} (\ell_1, 0|5) \xrightarrow{(t_{12},19)} (\ell_2, 4|0) \xrightarrow{(t_{21},20)} (\ell_1, 0|1) \xrightarrow{(t_{12},31)} (\ell_2, 11|0)$ reading the word $w_{\sigma'} = (a, 10)(b, 15)(a, 19)(b, 20)(a, 31)$. One can notice that ρ' and σ' are of different lengths. In fact, we can say something stronger, namely it is 1-$\forall$-*resilient* (and hence 1-$\exists$-*resilient*) as explained below.

The example consists of a single $(a.b)^*$ loop, where action a occurs between 3 and 12 time units after entering location ℓ_1, and action b occurs less than 7 time units after entering ℓ_2. A fault occurs either from ℓ_1, in which case action a occurs $12 + d$ time units after entering ℓ_1, with $d \in [0, 2]$, or from ℓ_2, i.e., when b occurs exactly 7 time units after entering ℓ_2. Once a fault has occurred, the iteration of a and b continues on $\dot{\ell}_1$ and $\dot{\ell}_2$ with non-faulty constraints. Consider a just faulty run ρ_f where fault occurs on event a. The timed word generated in ρ_f is of the form $w_f = (a, d_1).(b, d_2) \dots (a, d_k).(b, d_{k+1}).(a, d_{k+2})$, where $d_{k+2} = d_{k+1} + 12 + x$ with $x \in [0, 2]$. The word $w = (a, d_1).(b, d_2) \dots (a, d_k).(b, d_{k+1}).(a, d_{k+1} + 5).(b, d_{k+1} + 5 + x).(a, d_{k+1} + 5 + x + 7)$ is also recognized by the normal automaton, and ends at date $d_{k+1} + 12 + x$. Hence, for every just faulty word w_f which delays action a, there exists a word w such for every timed word v, if $w_f.v$ is accepted by the faulty automaton, $w.v$ is accepted by the normal automaton. Now, consider a fault occurring when playing action b. The just faulty word ending with a fault is of the form $w_f = (a, d_1).(b, d_2) \dots (a, d_k).(b, d_k + 7)$. All occurrences of a occur at a date between $d_j + 3$ and $d_j + 12$ for some date d_j at which location ℓ_1 is reached, (except the first time stamp $d_1 \in (5, 12)$) and all occurrences of b at a date strictly smaller than $d_i + 7$, where d_i is the date of last occurrence of a. Also, for any value $\epsilon \leq 7$ the word $w_\epsilon = (a, d_1).(b, d_2) \dots (a, d_k).(b, d_k + 7 - \epsilon)$ is non-faulty. Let $v_1 = 12 - d_1$, recall that $d_1 \in (5, 12)$. If we choose $\epsilon < v_1$ then the run $w_\epsilon^+ = (a, d_1 + \epsilon).(b, d_2 + \epsilon) \dots (a, d_k + \epsilon).(b, d_k + 7)$ is also non-faulty because $5 < d_1 + \epsilon < d_1 + v_1 = 12$. Clearly, we can extend w_ϵ^+ to match transitions fired from w_ϵ hence, the automaton of the example is 1-$\forall$-*resilient*.

B K-$\exists$-*resilience* and untimed K-$\exists$-*resilience*

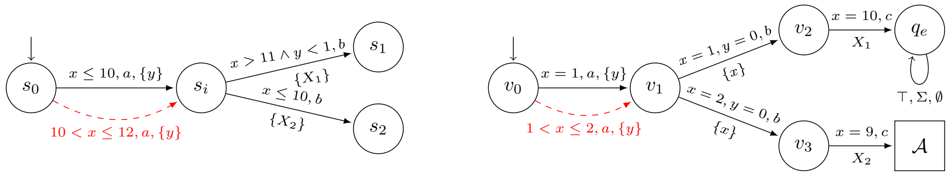


Figure 6 The gadgets $\mathcal{G}$ (left) and $\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}$ (right) which is untimed 2-$\exists$-*resilient* iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

▶ **Theorem 16** *Untimed K-$\exists$-resilience is PSPACE-Complete.*

Proof. *Membership:* For every run of $\mathcal{A}$, there is a path in $\mathcal{R}(\mathcal{A})$. So, $\mathcal{A}$ is untimed K-$\exists$-*resilient* if and only if, for all states q reached by a just faulty run, there exists a maximal accepting path σ from q such that, K steps after, the sequence of actions on its suffix σ_s agrees with that of an accepting path σ in $\mathcal{R}(\mathcal{A})$. We now prove that this property can be verified in PSPACE.

Let $q = (l, r)$ be a state of $\mathcal{R}(\mathcal{A}_\mathcal{P})$ reached after a just faulty run. K steps after reaching $q = (l, r)$ of $\mathcal{R}(\mathcal{A}_\mathcal{P})$, one can check in PSPACE, if there exists a path σ_s whose sequence of actions is the same as the suffix of an accepting path σ of $\mathcal{R}(\mathcal{A})$. That is, either both these end in a pair of accepting states from which no transitions are defined (both paths are maximal), or visit a pair of states twice such that the cyclic part of the path contains both an accepting state of $\mathcal{R}(\mathcal{A}_\mathcal{P})$ and an accepting state of $\mathcal{R}(\mathcal{A})$. To find these paths σ, σ_s, one just needs to guess them, i.e., build them synchronously by adding a pair of transitions to the already built path only if they have the same label. One needs to remember the current pair of states reached, and possibly guess a pair of states $(s_\mathcal{A}, s_{\mathcal{A}_\mathcal{P}})$ on which a cycle starts, and two bits $b_\mathcal{A}$ (resp. $b_{\mathcal{A}_\mathcal{P}}$) to remember if an accepting state of $\mathcal{A}$ (resp. $\mathcal{A}_\mathcal{P}$) has been seen since $(s_\mathcal{A}, s_{\mathcal{A}_\mathcal{P}})$. A maximal finite path or a lasso can be found on a path of length smaller than $|\mathcal{R}(\mathcal{A}_\mathcal{P})| \times |\mathcal{R}(\mathcal{A})|$, and the size of the currently explored path can be memorized with $\log_2(|\mathcal{R}(\mathcal{A}_\mathcal{P})| \times |\mathcal{R}(\mathcal{A})|)$ bits. This can be done in PSPACE. The complement of this, i.e., checking that no maximal path originating from q with the same labeling as a suffix of a word recognized by $\mathcal{R}(\mathcal{A})$ K steps after a fault exists, is in PSPACE too.

Now, to show that $\mathcal{A}$ is *not* untimed K-$\exists$-*resilient*, we simply have to find one untimed non-K-$\exists$-*resilient* witness state q reachable immediately after a fault. To find it, non deterministically guess such a witness state q along with a path of length not more than the size of $|\mathcal{R}(\mathcal{A}_\mathcal{P})|$ and apply the PSPACE procedure above to decide whether it is a untimed non-K-$\exists$-*resilience* witness. Guess of q is non-deterministic, which gives an overall NPSPACE complexity, but again, using Savitch's theorem, we can say that untimed K-$\exists$-*resilience* is in PSPACE.

Hardness: We can now show that untimed K-$\exists$-*resilience* is PSPACE-Hard. Consider a timed automaton $\mathcal{A}$ with alphabet Σ and the construction of an automata that uses a gadget shown in Figure 6 (right). Let us call this automaton $\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}$. This automaton reads a word $(a, 1).(b, 1).(c, 11)$ and then accepts all timed words 2 steps after a fault, via Σ loop on a particular accepting state q_e. If $\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}$ takes the faulty transition (marked in dotted red) then it resets all clocks of $\mathcal{A}$ and behaves as $\mathcal{A}$. The accepting states are $q_e \cup F$. Then, $\mathcal{A}$ has an accepting word if and only if $\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}$ is untimed 2-$\exists$-*resilient*. Since the emptiness problem for timed automata is PSPACE-Complete, the result follows. ◀

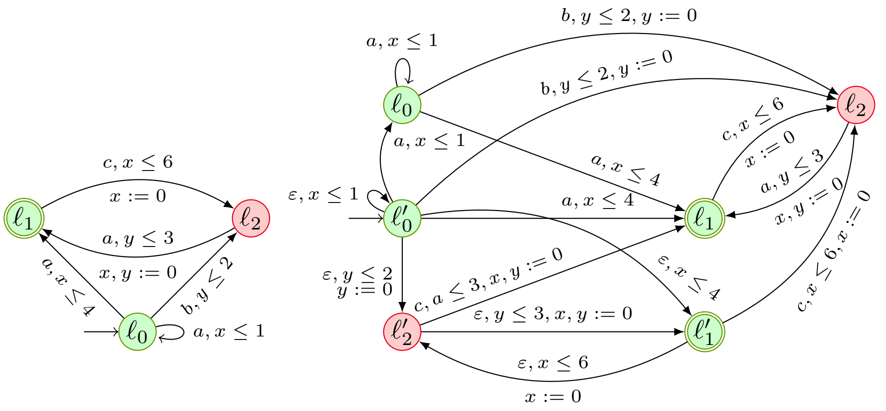


Figure 7 An example automaton $\mathcal{A}$ (left) and its suffix automaton $\mathcal{A}^S$ (right).

▶ **Definition 23** (Counting automaton). *Let $\mathcal{A}_{\mathcal{P}} = (L, I, X, \Sigma, T, F)$ and be a timed automaton with faulty transitions. Let $K \in \mathbb{N}$ be an integer. Then, the* faulty automaton *$\mathcal{B}^{\mathcal{P}}$ is a tuple $\mathcal{B}^{\mathcal{P}} = (L^{\mathcal{P}}, I^{\mathcal{P}}, X, \Sigma, T^{\mathcal{P}}, F^{\mathcal{P}})$ where $L^{\mathcal{P}} \subseteq \{L \times \{0\}\}$, $F^{\mathcal{P}} = F \times [-1, K]$, and initial set of states $I^{\mathcal{P}} = I \times \{-1\}$. Intuitively, -1 means no fault has occurred yet. Then we assign K and decrement to 0 to denote that K steps after fault have passed. The set of transitions $T^{\mathcal{P}}$ is as follows: We have $\big((l, n), g, a, R, (l', n')\big) \in T^{\mathcal{P}}$ if and only if either:*

- *$n \neq 0$ (no fault has occurred, or less than K steps of $\mathcal{B}$ have occurred), we have transition $t = (l, g, a, R, l) \in T$, and either: $n = -1$, the transition t is faulty and $n' = K$, or $n = -1$, the transition t is non faulty and $n' = -1$, or $n > 0$ and $n' = n - 1$.*
- *$n = n' = 0$ (at least K steps after a fault have occurred), and there exists a transition $t = (l, g, a, R, l') \in T$.*

C Resilience of Integer Reset Timed Automata

Let us recall some elements used to prove decidability of language inclusion in IRTA. For a given IRTA $\mathcal{A}$ we can define a map $f : \rho \to w_{unt}$ that maps every run ρ of $\mathcal{A}$ to an untimed word $w_{unt} \in (\{\checkmark, \delta\} \cup \Sigma)^*$. For a real number x with $k = \lfloor x \rfloor$, we define a map $dt(x)$ from $\mathbb{R}$ to $\{\checkmark, \delta\}^*$ as follows : $dt(x) = (\delta.\checkmark)^k$ if x is integral, and $dt(x) = (\delta.\checkmark)^k.\delta$ otherwise. Then, for two reals $x < y$, the map $dte(x, y)$ is the suffix that is added to $dt(x)$ to obtain $dt(y)$. Last, the map f associates to a word $w = (a_1, d_1) \dots (a_n, d_n)$ the word $f(w) = w_1.a_1.w_2.a_2 \dots w_n.a_n$ where each w_i is the word $w_i = dte(d_{i-1}, d_i)$. The map f maps global time elapse to a word of $\checkmark$ and δ but keeps actions unchanged. We define another map $f_\downarrow : w \to \{\checkmark, \delta\}^*$ that maps every word w of $\mathcal{A}$ to a word in $\{\checkmark, \delta\}^*$ dropping the actions from $f(w)$. Consider for example, a word $w = (a, 1.6)(b, 2.7)(c, 3.4)$ then, $f(w) = \delta\checkmark\delta a\checkmark\delta b\checkmark\delta c$, and $f_\downarrow(w) = \delta\checkmark\delta\checkmark\delta\checkmark\delta$. It is shown in [18] for two timed words ρ_1, ρ_2 with $f(\rho_1) = f(\rho_2)$ then $\rho_1 \in \mathcal{L}(\mathcal{A})$ if and only if $\rho_2 \in \mathcal{L}(\mathcal{A})$. It is also shown that we can construct a Marked Timed Automaton (MA) from $\mathcal{A}$ with one extra clock and polynomial increase in the number of locations such that $Unt(\mathcal{L}(MA)) = f(\mathcal{L}(\mathcal{A}))$. The MA of $\mathcal{A}$ duplicates transitions of $\mathcal{A}$ to differentiate firing at integral/non integral dates, plus transitions that make time elapsing visible using the additional clock which is reset at each global integral time stamp.

▶ **Definition 24** (Marked Timed Automaton (MA)). *Given a timed automaton $\mathcal{A} = (L, L_0, X, \Sigma, T, F)$ the Marked Timed Automaton of $\mathcal{A}$ is a tuple $MA = (L', L'_0, X \cup \{n\}, \Sigma \cup \{\checkmark, \delta\}, T', F')$ such that*

i) $n \notin X$

ii) $L' = L^0 \cup L^+$ *where for* $\alpha \in \{0, +\}, L^\alpha = \{l^\alpha \mid l \in L\}$

iii) $L'_0 = \{l^0 \mid l \in L_0\}$, $F' = \{l^0, l^+ \mid l \in F\}$ *and*

iv) T' *is defined by*

$$\begin{aligned} T' &= \{(l^0, a, g \wedge n = 0?, R, l'^0) \mid (l, a, g, R, l') \in E\} \\ &\cup \{(l^+, a, g \wedge 0 < n < 1?, R, l'^+) \mid (l, a, g, R, l') \in E\} \\ &\cup \bigcup_{l \in L} (l^0, \delta, 0 < n < 1, \emptyset, l^+) \cup \bigcup_{l \in L} (l^+, \checkmark, n = 1?, \{n\}, l^0) \end{aligned}$$

Then we have the following results.

▶ **Theorem 25** ([18], Thm. 5). *Let $\mathcal{A}$ be a timed automaton and* MA *be its marked automaton. Then $Unt(\mathcal{L}(MA)) = f(\mathcal{L}(\mathcal{A}))$*

▶ Remark 26. The marked timed automaton of an IRTA is also an IRTA.

The proofs of resilience for IRTA will also rely on the following properties,

▶ **Theorem 27** ([18], Thm. 3). *If $\mathcal{A}$ is an IRTA and $f(w) = f(w')$, then $w \in \mathcal{L}(\mathcal{A})$ if and only if $w' \in \mathcal{L}(\mathcal{A})$*

▶ **Lemma 28.** *The timed suffix language of an IRTA $\mathcal{A}$ can be recognized by an ε-IRTA $\mathcal{A}^S$*

Proof. Let $\mathcal{A} = (L, X, \Sigma, T, \mathcal{G}, F)$ be a timed automaton. We create an automaton $\mathcal{A}^S = (L^S, X, \Sigma \cup \{\varepsilon\}, T^S, \mathcal{G}, F)$ as follows. We set $L^S = L \cup L_\varepsilon$, where $L_\varepsilon = \{l_\varepsilon \mid l \in L\}$ i.e., L^S contains a copy of locations in $\mathcal{A}$ and another "silent" copy. The initial location of $\mathcal{A}^S$ is $l_{0,\varepsilon}$. We set $T^S = T \cup T_\varepsilon \cup T'_\varepsilon$, where $T_\varepsilon = \{(l_\varepsilon, \varepsilon, true, \emptyset, l) \mid l \in L\}$ and $T'_\varepsilon = \{(l_\varepsilon, \varepsilon, g, R, l'_\varepsilon) \mid \exists (l, a, g, R, l') \in T\}$. Clearly, for every timed word $w = (a_1, d_1) \dots (a_i, d_i)(a_{i+1}, d_{i+1}) \dots (a_n, d_n)$ of $\mathcal{L}(\mathcal{A})$ and index i, the word $w' = (\varepsilon, d_1) \dots . (\varepsilon, d_i)(a_{i+1}, d_{i+1}) \dots (a_n, d_n) = (a_{i+1}, d_{i+1}) \dots (a_n, d_n)$ is a recognized by $\mathcal{A}^S$, and it is easy to verify that $\mathcal{A}^s$ is an ε-IRTA. ◀

▶ **Lemma 29.** *For two IRTA $\mathcal{A}$ and $\mathcal{B}$ and their corresponding marked automata $\mathcal{A}_M$ and $\mathcal{B}_M$, $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ if and only if $untime(\mathcal{L}(\mathcal{A}_M)) \subseteq untime(\mathcal{L}(\mathcal{B}_M))$.*

Proof. ($\Rightarrow$) Assume, $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ and assume there exists a word $w \in untime(\mathcal{L}(\mathcal{A}_M))$, but $w \notin untime(\mathcal{L}(\mathcal{B}_M))$. Now, there exists a timed word $\rho \in \mathcal{L}(\mathcal{A})$ such that, $f(\rho) = w$. Clearly, $\rho \in \mathcal{L}(\mathcal{B})$, then clearly $f(\rho) = w \in untimed(\mathcal{L}(\mathcal{B}_M))$ a contradiction. So, $untime(\mathcal{L}(\mathcal{A}_m)) \subseteq untime(\mathcal{L}(\mathcal{B}_m))$.

($\Leftarrow$)Assume, $untime(\mathcal{L}(\mathcal{A}_M)) \subseteq untime(\mathcal{L}(\mathcal{B}_M))$, and $\mathcal{L}(\mathcal{A}) \nsubseteq \mathcal{L}(\mathcal{B})$. Then, there exists a timed word $\rho \in \mathcal{L}(\mathcal{A})$ such that $\rho \notin \mathcal{L}(\mathcal{B})$. Assume $f(\rho) = w$, then clearly, $w \in untime(\mathcal{L}(\mathcal{A}_M))$ and $w \in untime(\mathcal{L}(\mathcal{B}_M))$. So, there exists a timed word $\rho' \in \mathcal{L}(\mathcal{A})$ such that, $f(\rho') = w = f(\rho)$. According to Theorem 27 we can conclude that, $\rho \in \mathcal{L}(\mathcal{B})$ a contradiction. ◀

▶ Remark 30. Lemma 29 shows that the timed and untimed language inclusion problems for IRTA are in fact the same problem. So, as we can solve the timed language inclusion problem by solving an untimed language inclusion problem of IRTA and vice-versa, the untimed language inclusion for IRTA is also EXPSPACE-Complete.

▶ **Theorem 31.** *Timed K-$\forall$-resilience of IRTA is EXPSPACE-Hard.*

Proof. We proceed by a reduction from the language inclusion problem of IRTA, known to be EXPSPACE-Complete [4]. The idea of the proof follows the same lines as the untimed *K-$\forall$-resilience* of timed automata. Assume we are given IRTA $\mathcal{A}_1, \mathcal{A}_2$. a, b, c are symbols not in the alphabets of $\mathcal{A}_1, \mathcal{A}_2$. Consider $\mathcal{B}$ in Figure 8 (left). It is easy to see that $L(\mathcal{B})$=$(a, 1)(b, 1)(c, 11)(L(\mathcal{A}_1)$+$11)$, where $L(\mathcal{A}_1)+k = \{(a_1, d_1+k)(a_2, d_2+k) \dots (a_n, d_n+k) \mid$

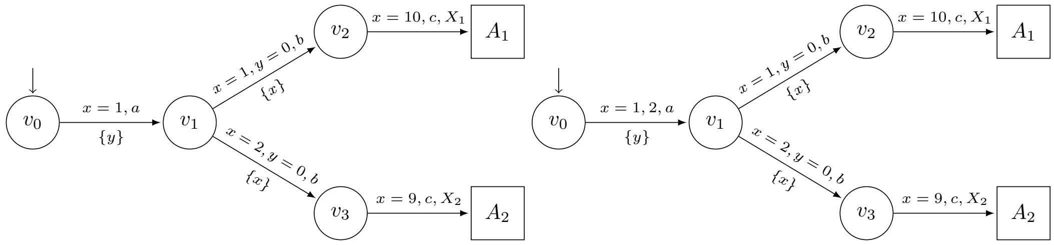


Figure 8 The automaton B (left) and the faulty automaton $B_{\mathcal{P}}$ (right).

$(a_1, d_1) \dots (a_n, d_n) \in L(\mathcal{A}_1)\}$. Associate a fault model $\mathcal{P}(a) = 1$, where the fault of a is 1. We construct an IRTA $\mathcal{B}_{\mathcal{P}}$ as shown in Figure 8 (right). Notice that in general, IRTAs are not closed under the fault insertion; the enlarged transition in $\mathcal{B}$ has guard $1 \leq x \leq 2$, and resets y. This violates the integer reset condition; however, since a value $1 < x < 2$ when resetting y clearly does not lead to acceptance in $\mathcal{B}_{\mathcal{P}}$, we prune away that transition resulting in $\mathcal{B}_{\mathcal{P}}$ as in Figure 8 (right). This resulting faulty automaton is an IRTA.

The language accepted by $\mathcal{B}_{\mathcal{P}}$ is $L(\mathcal{B}) \cup (a, 2)(b, 2)(c, 11)(L(\mathcal{A}_2) + 11)$. Considering $K = 2$, $\mathcal{B}_{\mathcal{P}}$ is BTN in 2 steps after the fault if and only if $L(\mathcal{A}_2) \subseteq L(\mathcal{A}_1)$. The EXPSPACE hardness of the timed K-$\forall$-*resilience* of IRTA follows from the EXPSPACE completeness of the inclusion of IRTA. ◀

▶ **Theorem 32.** *K-$\exists$-resilience for IRTA is PSPACE-Hard.*

Proof. Consider an IRTA $\mathcal{A}$ with alphabet Σ and the construction of an automata that uses a gadget shown below in Figure 9 (left). Let us call this automaton $\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}$. It is easy to see that the $L(\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}})$=$(a, 1)(b, 1)(c, 11)\big((\Sigma \times \mathbb{R})^* + 11\big)$, where $L(A_1) + k = \{(a_1, d_1 + k)(a_2, d_2 + k) \dots (a_n, d_n + k) \mid (a_1, d_1) \dots (a_n, d_n) \in L(A_1)\}$. The Σ loop on a particular accepting state q_e is responsible for acceptance of all timed word. Now, associate a fault model $\mathcal{P}(a) \to 1$ with $\mathcal{B}$, where the fault of a is 1. Let us call this enlarged automaton $\mathcal{B}_{(\Sigma^* \subseteq \mathcal{A})_P}$. We can prune away the transition $1 < x < 2$ resetting y which does not lead to acceptance, and resulting in an IRTA with the same language, represented in Figure 9 (right). The language accepted by $\mathcal{B}_{(\Sigma^* \subseteq \mathcal{A})_P}$ is $L(\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}) \cup (a, 2)(b, 2)(c, 11)(L(\mathcal{A}) + 11)$. The accepting states are $q_e \cup F$, where F is the set of final states of $\mathcal{A}$. Then $\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}$ is K-$\exists$-*resilient* if and only if $L(\mathcal{A}) \neq \emptyset$. ◀

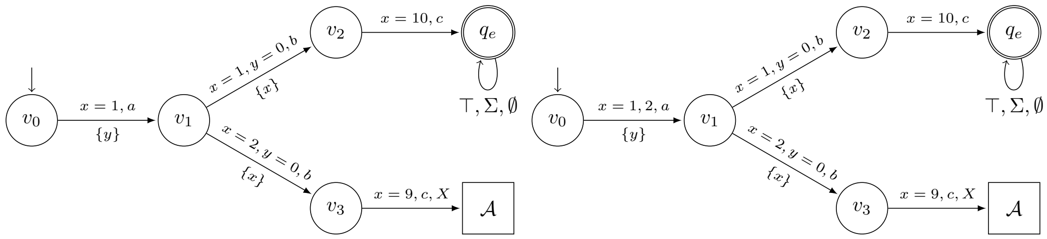


Figure 9 The IRTA $\mathcal{B}_{\Sigma^* \subseteq \mathcal{A}}$ (left) and the faulty IRTA $\mathcal{B}_{(\Sigma^* \subseteq \mathcal{A})_P}$ (right).

▶ Remark 33. The untimed language inclusion problem is shown to be EXPSPACE-Complete in Remark 30. The emptiness checking of timed automata is done by checking the emptiness of its untimed region automaton. So, to show the hardness of untimed K-$\forall$-*resilient* or K-$\exists$-*resilient* problems for IRTA, it is sufficient to reduce the untimed language inclusion problem and untimed language emptiness problem of IRTA respectively. This reduction can be done by using the same gadget as shown in Theorem 31 and Theorem 32 respectively.

On the Complexity of Intersection Non-emptiness for Star-Free Language Classes

Emmanuel Arrighi
University of Bergen, Norway

Henning Fernau
Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Stefan Hoffmann
Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Markus Holzer
Institut für Informatik, Universität Giessen, Germany

Ismaël Jecker
Institute of Science and Technology, Klosterneuburg, Austria

Mateus de Oliveira Oliveira
University of Bergen, Norway

Petra Wolf
Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Abstract

In the Intersection Non-emptiness problem, we are given a list of finite automata $A_1, A_2, \dots, A_m$ over a common alphabet Σ as input, and the goal is to determine whether some string $w \in \Sigma^*$ lies in the intersection of the languages accepted by the automata in the list. We analyze the complexity of the Intersection Non-emptiness problem under the promise that all input automata accept a language in some level of the dot-depth hierarchy, or some level of the Straubing-Thérien hierarchy. Automata accepting languages from the lowest levels of these hierarchies arise naturally in the context of model checking. We identify a dichotomy in the dot-depth hierarchy by showing that the problem is already NP-complete when all input automata accept languages of the levels $\mathcal{B}_0$ or $\mathcal{B}_{1/2}$ and already PSPACE-hard when all automata accept a language from the level $\mathcal{B}_1$. Conversely, we identify a tetrachotomy in the Straubing-Thérien hierarchy. More precisely, we show that the problem is in AC^0 when restricted to level $\mathcal{L}_0$; complete for L or NL, depending on the input representation, when restricted to languages in the level $\mathcal{L}_{1/2}$; NP-complete when the input is given as DFAs accepting a language in $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$; and finally, PSPACE-complete when the input automata accept languages in level $\mathcal{L}_2$ or higher. Moreover, we show that the proof technique used to show containment in NP for DFAs accepting languages in $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$ does not generalize to the context of NFAs. To prove this, we identify a family of languages that provide an exponential separation between the state complexity of general NFAs and that of partially ordered NFAs. To the best of our knowledge, this is the first superpolynomial separation between these two models of computation.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Problems, reductions and completeness

Keywords and phrases Intersection Non-emptiness Problem, Star-Free Languages, Straubing-Thérien Hierarchy, dot-depth Hierarchy, Commutative Languages, Complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.34

Related Version *Full Version*: `http://arxiv.org/abs/2110.01279` [3]

Funding *Emmanuel Arrighi*: Research Council of Norway (no. 274526), IS-DAAD (no. 309319).
Henning Fernau: DAAD PPP (no. 57525246).
Ismaël Jecker: Marie Skłodowska-Curie Grant Agreement no. 754411.
Mateus de Oliveira Oliveira: Research Council of Norway (no. 288761), IS-DAAD (no. 309319).
Petra Wolf: DFG project FE 560/9-1, DAAD PPP (no. 57525246).

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 34; pp. 34:1–34:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We like to thank Lukas Fleischer and Michael Wehar for our discussions. This work started at the Schloss Dagstuhl Event 20483 *Moderne Aspekte der Komplexitätstheorie in der Automatentheorie* `https://www.dagstuhl.de/20483`.

1 Introduction

The Intersection Non-emptiness problem for finite automata is one of the most fundamental and well studied problems in the interplay between algorithms, complexity theory, and automata theory [12, 20, 21, 24, 26, 43, 44, 45]. Given a list $A_1, A_2, \ldots, A_m$ of finite automata over a common alphabet Σ, the goal is to determine whether there is a string $w \in \Sigma^*$ that is accepted by each of the automata in the list. This problem is PSPACE-complete when no restrictions are imposed [24], and becomes NP-complete when the input automata accept unary languages (implicitly contained already in [38]) or finite languages [34].

In this work, we analyze the complexity of the Intersection Non-emptiness problem under the assumption that the languages accepted by the input automata belong to a given level of the Straubing-Thérien hierarchy [33, 39, 40, 42] or to some level of the Cohen-Brzozowski dot-depth hierarchy [6, 11, 33]. Somehow, these languages are severely restricted, in the sense that both hierarchies, which are infinite, are entirely contained in the class of star-free languages, a class of languages that can be represented by expressions that use union, concatenation, and complementation, but *no* Kleene star operation [6, 8, 33]. Yet, languages belonging to fixed levels of either hierarchy may already be very difficult to characterize, in the sense that the very problem of deciding whether the language accepted by a given finite automaton belongs to a given full level or half-level k of either hierarchy is open, except for a few values of k [2, 15, 16, 33]. It is worth noting that while the problem of determining whether a given automaton accepts a language in a certain level of either the dot-depth or of the Straubing-Thérien hierarchy is computationally hard (Theorem 1), automata accepting languages in lower levels of these hierarchies arise naturally in a variety of applications such as model checking where the Intersection Non-emptiness problem is of fundamental relevance [1, 4, 5].

An interesting question to consider is how the complexity of the Intersection Non-emptiness problem changes as we move up in the levels of the Straubing-Thérien hierarchy or in the levels of the dot-depth hierarchy. In particular, does the complexity of this problem changes gradually, as we increase the complexity of the input languages? In this work, we show that this is actually not the case, and that the complexity landscape for the Intersection Non-emptiness problem is already determined by the very first levels of either hierarchy (see Figure 1). Our first main result states that the Intersection Non-emptiness problem for NFAs and DFAs accepting languages from the level 1/2 of the Straubing-Thérien hierarchy are NL-complete and L-complete, respectively, under AC^0 reductions (Theorem 3). Additionally, this completeness result holds even in the case of unary languages. To prove hardness for NL and L, respectively, we will use a simple reduction from the reachability problem for DAGs and for directed trees, respectively. Nevertheless, the proof of containment in NL and in L, respectively, will require a new insight that may be of independent interest. More precisely, we will use a characterization of languages in the level 1/2 of the Straubing-Thérien hierarchy as shuffle ideals to show that the Intersection Non-emptiness problem can be reduced to concatenation non-emptiness (Lemma 5). This allows us to decide Intersection Non-emptiness by analyzing each finite automaton given at the input individually. It is worth mentioning that this result is optimal in the sense that the problem becomes NP-hard even if we allow a single DFA to accept a language from $\mathcal{L}_1$, and require all the others to accept languages from $\mathcal{L}_{1/2}$ (Theorem 8).

Subsequently, we analyze the complexity of Intersection Non-emptiness when all input automata are assumed to accept languages from one of the levels of $\mathcal{B}_0$ or $\mathcal{B}_{1/2}$ of the dot-depth hierarchy, or from the levels $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$ of the Straubing-Thérien hierarchy. It is worth noting that NP-hardness follows straightforwardly from the fact that Intersection Non-emptiness for DFAs accepting finite languages is already NP-hard [34]. Containment in NP, on the other hand, is a more delicate issue, and here the representation of the input automaton plays an important role. A characterization of languages in $\mathcal{L}_{3/2}$ in terms of languages accepted by partially ordered NFAs [37] is crucial for us, combined with the fact that Intersection Non-emptiness when the input is given by such automata is NP-complete [29]. Intuitively, the proof in [29] follows by showing that the minimum length of a word in the intersection of languages in the level 3/2 of the Straubing-Thérien hierarchy is bounded by a polynomial on the sizes of the minimum partially ordered NFAs accepting these languages. To prove that Intersection Non-emptiness is in NP when the input automata are given as DFAs, we prove a new result establishing that the number of Myhill-Nerode equivalence classes in a language in the level $\mathcal{L}_{3/2}$ is at least as large as the number of states in a minimum partially ordered automaton representing the same language (Lemma 12).

Interestingly, we show that the proof technique used to prove this last result does not generalize to the context of NFAs. To prove this, we carefully design a sequence $(L_n)_{n\in\mathbb{N}_{\geq 1}}$ of languages over a binary alphabet such that for every $n \in \mathbb{N}_{\geq 1}$, the language L_n can be accepted by an NFA of size n, but any partially ordered NFA accepting L_n has size $2^{\Omega(\sqrt{n})}$. This lower bound is ensured by the fact that the syntactic monoid of L_n has many $\mathcal{J}$-factors. Our construction is inspired by a technique introduced by Klein and Zimmermann, in a completely different context, to prove lower bounds on the amount of look-ahead necessary to win infinite games with delay [22]. To the best of our knowledge, this is the first exponential separation between the state complexity of general NFAs and that of partially ordered NFAs. While this result does not exclude the possibility that Intersection Non-emptiness for languages in $\mathcal{L}_{3/2}$ represented by general NFAs is in NP, it gives some indication that proving such a containment requires substantially new techniques.

Finally, we show that Intersection Non-emptiness for both DFAs and for NFAs is already PSPACE-complete if all accepting languages are from the level $\mathcal{B}_1$ of the dot-depth hierarchy or from the level $\mathcal{L}_2$ of the Straubing-Thérien hierarchy. We can adapt Kozen's classical PSPACE-completeness proof by using the complement of languages introduced in [28] in the study of partially ordered automata. Since the languages in [28] belong to $\mathcal{L}_{3/2}$, their complement belong to $\mathcal{L}_2$ (and to $\mathcal{B}_1$), and therefore, the proof follows.

Due to space constraints, many details of the paper can be found in the long version [3].

2 Preliminaries

We let $\mathbb{N}_{\geq k}$ denote the set of natural numbers greater or equal than k.

We assume the reader to be familiar with the basics in computational complexity theory [31]. In particular, we recall the inclusion chain: $\mathsf{AC}^0 \subset \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$. Let AC^0 (NC^1, respectively) refer to the class of problems accepted by Turing machines with a bounded (unbounded, respectively) number of alternations in logarithmic time; alternatively one can define these classes by uniform Boolean circuits. Here, L (NL, respectively) refers to the class of problems that are accepted by deterministic (nondeterministic, respectively) Turing machines with logarithmic space, P (NP, respectively) denotes the class of problems solvable by deterministic (nondeterministic, respectively) Turing machines in polynomial time, and PSPACE refers to the class of languages accepted by deterministic or

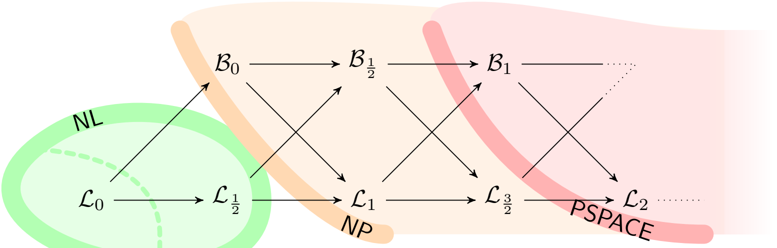


Figure 1 Straubing-Thérien and dot-depth hierarchies: the Intersection Non-emptiness status.

nondeterministic Turing machines in polynomial space [35]. Completeness and hardness are always meant with respect to deterministic logspace many-one reductions unless otherwise stated. We will also consider the parameterized class XP of problems that can be solved in time $n^{f(k)}$, where n is the size of the input, k is a parameter, and f is a computable function [13].

We mostly consider *nondeterministic finite automata* (NFAs). An NFA A is a tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite *state set* with the *start state* $q_0 \in Q$, the *alphabet* Σ is a finite set of input symbols, and $F \subseteq Q$ is the *final state set*. The *transition function* $\delta : Q \times \Sigma \to 2^Q$ extends to words from Σ^* as usual. Here, 2^Q denotes the powerset of Q. By $L(A) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset \,\}$, we denote the *language accepted by* A. The NFA A is a *deterministic finite automaton* (DFA) if $|\delta(q, a)| = 1$ for every $q \in Q$ and $a \in \Sigma$. Then, we simply write $\delta(q, a) = p$ instead of $\delta(q, a) = \{p\}$. If $|\Sigma| = 1$, we call A a *unary* automaton.

We study Intersection Non-emptiness problems and their complexity. For finite automata, this problem is defined as follows:

- *Input*: Finite automata $A_i = (Q_i, \Sigma, \delta_i, q_{(0,i)}, F_i)$, for $1 \leq i \leq m$.
- *Question*: Is there a word w that is accepted by all A_i, i.e., is $\bigcap_{i=1}^{m} L(A_i) \neq \emptyset$?

Observe that the automata have a common input alphabet. Note that the complexity of the non-emptiness problem for finite automata of a certain type is a lower bound for the Intersection Non-emptiness for this particular type of automata. Throughout the paper we are mostly interested in the complexity of the Intersection Non-emptiness problem for finite state devices whose languages are contained in a particular language class.

We study the computational complexity of the intersection non-emptiness for languages from the classes of the Straubing-Thérien [39, 42] and Cohen-Brzozowski's dot-depth hierarchy [11]. Both hierarchies are concatenation hierarchies that are defined by alternating the use of polynomial and Boolean closures. Let's be more specific. Let Σ be a finite alphabet. A language $L \subseteq \Sigma^*$ is a *marked product* of the languages $L_0, L_1, \ldots, L_k$, if $L = L_0 a_1 L_1 \cdots a_k L_k$, where the a_i's are letters. For a class of languages $\mathcal{M}$, the *polynomial closure* of $\mathcal{M}$ is the set of languages that are finite unions of marked product of languages from $\mathcal{M}$.

The concatenation hierarchy of basis $\mathcal{M}$ (a class of languages) is defined as follows (also refer to [32]): Level 0 is $\mathcal{M}$, i.e., $\mathcal{M}_0 = \mathcal{M}$ and, for each $n \geq 0$,

1. $\mathcal{M}_{n+1/2}$, that is, level $n + 1/2$, is the polynomial closure of level n and
2. $\mathcal{M}_{n+1}$, that is, level $n + 1$, is the Boolean closure of level $n + 1/2$.

The basis of the dot-depth hierarchy is the class of all finite and co-finite languages[1] and their classes are referred to as $\mathcal{B}_n$ ($\mathcal{B}_{n+1/2}$, respectively), while the basis of the Straubing-Thérien hierarchy is the class of languages that contains only the empty set and Σ^* and their classes are denoted by $\mathcal{L}_n$ ($\mathcal{L}_{n+1/2}$, respectively). Their inclusion relation is given by

[1] The dot-depth hierarchy, apart level $\mathcal{B}_0$, coincides with the concatenation hierarchy starting with the language class $\{\emptyset, \{\lambda\}, \Sigma^+, \Sigma^*\}$.

$$\mathcal{B}_{n+1/2} \subseteq \mathcal{B}_{n+1} \subseteq \mathcal{B}_{n+3/2} \quad \text{and} \quad \mathcal{L}_{n+1/2} \subseteq \mathcal{L}_{n+1} \subseteq \mathcal{L}_{n+3/2},$$

for $n \geq 0$, and

$$\mathcal{L}_{n-1/2} \subseteq \mathcal{B}_{n-1/2} \subseteq \mathcal{L}_{n+1/2} \quad \text{and} \quad \mathcal{L}_n \subseteq \mathcal{B}_n \subseteq \mathcal{L}_{n+1},$$

for $n \geq 1$. In particular, $\mathcal{L}_0 \subseteq \mathcal{B}_0$, $\mathcal{B}_0 \subseteq \mathcal{B}_{1/2}$, and $\mathcal{L}_0 \subseteq \mathcal{L}_{1/2}$. Both hierarchies are infinite for alphabets of at least two letters and completely exhaust the class of star-free languages, which can be described by expressions that use union, concatenation, and complementation, but *no* Kleene star operation. For singleton letter alphabets, both hierarchies collapse to $\mathcal{B}_0$ and $\mathcal{L}_1$, respectively. Next, we describe the first few levels of each of these hierarchies:

Straubing-Thérien hierarchy: A language of Σ^* is of level 0 if and only if it is empty or equal to Σ^*. The languages of level 1/2 are exactly those languages that are a finite (possibly empty) union of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \cdots a_k \Sigma^*$, where the a_i's are letters from Σ. The languages of level 1 are finite Boolean combinations of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \cdots a_k \Sigma^*$, where the a_i's are letters. These languages are also called *piecewise* testable languages. In particular, all finite and co-finite languages are of level 1. Finally, the languages of level 3/2 of Σ^* are the finite unions of languages of the form $\Sigma_0^* a_1 \Sigma_1^* a_2 \cdots a_k \Sigma_k^*$, where the a_i's are letters from Σ and the Σ_i are subsets of Σ.

Dot-depth hierarchy: A language of Σ^* is of dot-depth (level) 0 if and only if it is finite or co-finite. The languages of dot-depth 1/2 are exactly those languages that are a finite union of languages of the form $u_0 \Sigma^* u_1 \Sigma^* u_2 \cdots u_{k-1} \Sigma^* u_k$, where $k \geq 0$ and the u_i's are words from Σ^*. The languages of dot-depth 1 are finite Boolean combinations of languages of the form $u_0 \Sigma^* u_1 \Sigma^* u_2 \cdots u_{k-1} \Sigma^* u_k$, where $k \geq 0$ and the u_i's are words from Σ^*.

It is worth mentioning that in [37] it was shown that partially ordered NFAs (with multiple initial states) characterize the class $\mathcal{L}_{3/2}$, while partially ordered DFAs characterize the class of $\mathcal{R}$-trivial languages [7], a class that is strictly in between $\mathcal{L}_1$ and $\mathcal{L}_{3/2}$. For an automaton A with input alphabet Σ, a state q is reachable from a state p, written $p \leq q$, if there is a word $w \in \Sigma^*$ such that $q \in \delta(p, w)$. An automaton is partially ordered if $\leq$ is a partial order. Partially ordered automata are sometimes also called acyclic or weakly acyclic automata. We refer to a partially ordered NFA (DFA, respectively) as poNFA (poDFA, respectively).

The fact that some of our results have a promise looks a bit technical, but the following result implies that we cannot get rid of this condition in general. To this end, we study, for a language class $\mathcal{L}$, the following question of $\mathcal{L}$-Membership.

- *Input*: A finite automaton A.
- *Question*: Is $L(A) \in \mathcal{L}$?

▶ **Theorem 1.** *For each level $\mathcal{L}$ of the Straubing-Thérien or the dot-depth hierarchies, the $\mathcal{L}$-Membership problem for NFAs is* PSPACE*-hard, even when restricted to binary alphabets.*

Proof. For the PSPACE-hardness, note that each of the classes contains $\{0,1\}^*$ and is closed under quotients, since each class is a positive variety. As Non-universality is PSPACE-hard for NFAs, we can apply Theorem 3.1.1 of [19], first reducing regular expressions to NFAs. ◀

For some of the lower levels of the hierarchies, we also have containment in PSPACE, but in general, this is unknown, as it connects to the famous open problem if, for instance, $\mathcal{L}$-Membership is decidable for $\mathcal{L} = \mathcal{L}_3$; see [27, 33] for an overview on the decidability status of these questions. Checking for $\mathcal{L}_0$ up to $\mathcal{L}_2$ and $\mathcal{B}_0$ up to $\mathcal{B}_1$ containment for DFAs can be done in NL and is also complete for this class by ideas similar to the ones used in [9].

3 Inside Logspace

A language of Σ^* belongs to level 0 of the Straubing-Thérien hierarchy if and only if it is empty or Σ^*. The INTERSECTION NON-EMPTINESS problem for language from this language family is not entirely trivial, because we have to check for emptiness. Since by our problem definition the property of a language being a member of level 0 is a promise, we can do the emptiness check within AC^0, since we only have to verify whether the empty word belongs to the language L specified by the automaton. In case $\varepsilon \in L$, then $L = \Sigma^*$; otherwise $L = \emptyset$. Since in the definition of finite state devices we do not allow for ε-transitions, we thus only have to check whether the initial state is also an accepting one. Therefore, we obtain:

▶ **Theorem 2.** *The INTERSECTION NON-EMPTINESS problem for DFAs or NFAs accepting languages from $\mathcal{L}_0$ belongs to AC^0.*

For the languages of level $\mathcal{L}_{1/2}$ we find the following completeness result.

▶ **Theorem 3.** *The INTERSECTION NON-EMPTINESS problem for NFAs accepting languages from $\mathcal{L}_{1/2}$ is NL-complete. Moreover, the problem remains NL-hard even if we restrict the input to NFAs over a unary alphabet. If the input instance contains only DFAs, the problem becomes L-complete (under weak reductions[2]).*

Hardness is shown by standard reductions from variants of graph accessibility [17, 41].

▶ **Lemma 4.** *The INTERSECTION NON-EMPTINESS problem for NFAs over unary alphabet accepting languages from $\mathcal{L}_{1/2}$ is NL-hard. If the input instance contains only DFAs, the problem becomes L-hard under weak reductions.*

It remains to show containment in logspace. To this end, we utilize an alternative characterization of the languages of level 1/2 of the Straubing-Thérien hierarchy as exactly those languages that are shuffle ideals. A language L is a *shuffle ideal* if, for every word $w \in L$ and $v \in \Sigma^*$, the set $w \sqcup\!\sqcup v$ is contained in L, where $w \sqcup\!\sqcup v := \{ w_0v_0w_1v_1 \dots w_kv_k \mid w = w_0w_1 \dots w_k \text{ and } v = v_0v_1 \dots v_k \text{ with } w_i, v_i \in \Sigma^*, \text{ for } 0 \leq i \leq k \}$. The operation $\sqcup\!\sqcup$ naturally generalizes to sets. For the level $\mathcal{L}_{1/2}$, we find the following situation.

▶ **Lemma 5.** *Let $m \geq 1$ and languages $L_i \subseteq \Sigma^*$, for $1 \leq i \leq m$, be shuffle ideals, i.e., they belong to $\mathcal{L}_{1/2}$. Then, $\bigcap_{i=1}^m L_i \neq \emptyset$ iff the shuffle ideal $L_1L_2 \cdots L_m \neq \emptyset$ iff $L_i \neq \emptyset$ for every i with $1 \leq i \leq m$. Finally, $L_i \neq \emptyset$, for $1 \leq i \leq m$, iff $(a_1a_2 \dots a_k)^{\ell_i} \in L_i$, where $\Sigma = \{a_1, a_2, \dots a_k\}$ and the shortest word in L_i is of length ℓ_i.*

Now, we are ready to prove containment in logspace.

▶ **Lemma 6.** *The INTERSECTION NON-EMPTINESS problem for NFAs accepting languages from $\mathcal{L}_{1/2}$ belongs to NL. If the input instance contains only DFAs, the problem is solvable in L.*

Proof. In order to solve the INTERSECTION NON-EMPTINESS problem for given finite automata $A_1, A_2, \dots, A_m$ with a common input alphabet Σ, regardless of whether they are deterministic or nondeterministic, it suffices to check non-emptiness for all languages $L(A_i)$, for $1 \leq i \leq m$, in sequence, because of Lemma 5. To this end, membership of the words $(a_1a_2 \dots a_k)^{\ell_i}$ in L_i has to be tested, where ℓ_i is the length of the shortest word in L_i. Obviously, all ℓ_i are linearly bounded in the number of states of the appropriate finite automaton

[2] Some form of AC^0 reducibility can be employed.

that accepts L_i. Hence, for NFAs as input instance, the test can be done on a nondeterministic logspace-bounded Turing machine, guessing the computations in the individual NFAs on the input word $(a_1a_2\ldots a_k)^{\ell_i}$. For DFAs as input instance, nondeterminism is not needed, so that the procedure can be implemented on a deterministic Turing machine. ◀

4 NP-Completeness

In contrast to the Straubing-Thérien hierarchy, the Intersection Non-emptiness problem for languages from the dot-depth hierarchy is already NP-hard in the lowest level $\mathcal{B}_0$. More precisely, Intersection Non-emptiness for finite languages is NP-hard [34, Theorem 1] and $\mathcal{B}_0$ already contains all finite languages. Hence, the Intersection Non-emptiness problem for languages from the Straubing-Thérien hierarchy of level $\mathcal{L}_1$ and above is NP-hard, too. For the levels $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, or $\mathcal{L}_{3/2}$, we give matching complexity upper bounds if the input are DFAs, yielding the first main result of this section proven in Subsection 4.1.

▶ **Theorem 7.** *The Intersection Non-emptiness problem for DFAs accepting languages from either $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, or $\mathcal{L}_{3/2}$ is NP-complete. The same holds for poNFAs instead of DFAs. The results hold even for a binary alphabet.*

For the level $\mathcal{L}_1$ of the Straubing-Thérien hierarchy, we obtain with the next main theorem a stronger result. Recall that if all input DFAs accept languages from $\mathcal{L}_{1/2}$, the Intersection Non-emptiness problem is L-complete due to Lemmata 4 and 6.

▶ **Theorem 8.** *The Intersection Non-emptiness problem for DFAs is NP-complete even if only one DFA accepts a language from $\mathcal{L}_1$ and all other DFAs accept languages from $\mathcal{L}_{1/2}$ and the alphabet is binary.*

The proof of this theorem will be given in Subsection 4.2.

For the level $\mathcal{B}_0$, we obtain a complete picture of the complexity of the Intersection Non-emptiness problem, independent of structural properties of the input finite automata, i.e., we show that here the problem is NP-complete for general NFAs.

For the level $\mathcal{L}_{3/2}$, if the input NFA are from the class of poNFA, which characterize level $\mathcal{L}_{3/2}$, then the Intersection Non-emptiness problem is known to be NP-complete [28]. Recall that $\mathcal{L}_{3/2}$ contains the levels $\mathcal{B}_{1/2}$, and $\mathcal{L}_1$ and hence also languages from these classes can be represented by poNFAs. But if the input automata are given as NFAs without any structural property, then the precise complexity of Intersection Non-emptiness for $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, and $\mathcal{L}_{3/2}$ is an open problem and narrowed by NP-hardness and membership in PSPACE. We present a “No-Go-Theorem” by proving that for an NFA accepting a co-finite language, the smallest equivalent poNFA is exponentially larger in Subsection 4.3.

▶ **Theorem 9.** *For every $n \in \mathbb{N}_{\geq 1}$, there exists a language $L_n \in \mathcal{B}_0$ on a binary alphabet such that L_n is recognized by an NFA of size $O(n^2)$, but the minimal poNFA recognizing L_n has more than 2^{n-1} states.*

While for NFAs the precise complexity for Intersection Non-emptiness of languages from $\mathcal{L}_1$ remains open, we can tackle this gap by narrowing the considered language class to *commutative* languages in level $\mathcal{L}_1$; recall that a language $L \subseteq \Sigma^*$ is *commutative* if, for any $a, b \in \Sigma$ and words $u, v \in \Sigma^*$, we have that $uabv \in L$ implies $ubav \in L$. We show that for DFAs, this restricted Intersection Non-emptiness problem remains NP-hard, in case the alphabet is unbounded. Concerning membership in NP, we show that even for NFAs, the Intersection Non-emptiness problem for *commutative* languages is contained in NP in general and in particular for commutative languages on each level. This generalizes the case of unary NFAs. Note that for commutative languages, the Straubing-Thérien hierarchy collapses at level $\mathcal{L}_{3/2}$. See Subsection 4.4 for the proofs.

▶ **Theorem 10.** *The* Intersection Non-emptiness *problem*

- *is* NP*-hard for DFAs accepting* commutative *languages in* $\mathcal{L}_1$*, but*
- *is contained in* NP *for NFAs accepting* commutative *languages that might not be star-free.*

The proof of NP-hardness for commutative star-free languages in $\mathcal{L}_1$ requires an arbitrary alphabet. However, we show that Intersection Non-emptiness is contained in XP for specific forms of NFAs such as poNFAs or DFAs accepting commutative languages, with the size of the alphabet as the parameter, i.e., for fixed input alphabets, our problem is solvable in polynomial time.

4.1 NP-Membership

Next, we focus on the NP-membership part of Theorem 7 and begin by proving that for $\mathcal{B}_0$, regardless of whether the input automata are NFAs or DFAs, the Intersection Non-emptiness problem is contained in NP and therefore NP-complete in combination with [34].

▶ **Lemma 11.** *The* Intersection Non-emptiness *problem for DFAs or NFAs all accepting languages from* $\mathcal{B}_0$ *is contained in* NP*.*

Proof. Let $A_1, A_2, \ldots, A_m$ be NFAs accepting languages from $\mathcal{B}_0$. If all NFAs accept co-finite languages, which can be verified in deterministic polynomial time, the intersection $\bigcap_{i=1}^{m} L(A_i)$ is non-empty. Otherwise, there is at least one NFA accepting a finite language, where the longest word is bounded by the number of states of this device. Hence, if $\bigcap_{i=1}^{m} L(A_i) \neq \emptyset$, there is a word w of length polynomial in the length of the input that witnesses this fact. Such a w can be nondeterministically guessed by a Turing machine checking membership of w in $L(A_i)$, for all NFAs A_i, in sequence. This shows containment in NP as desired. ◀

Notice that Masopust and Krötzsch have shown in [28] that Intersection Non-emptiness for poDFAs and for poNFAs is NP-complete. Also the unary case is discussed there, which can be solved in polynomial time. We cannot directly make use of these results, as we consider arbitrary NFAs or DFAs as inputs, only with the promise that they accept languages from a certain level of the studied hierarchies. In order to prove that for the levels $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, and $\mathcal{L}_{3/2}$, the Intersection Non-emptiness problem for DFAs is contained in NP, it is sufficient to prove the claim for $\mathcal{L}_{3/2}$ as all other stated levels are contained in $\mathcal{L}_{3/2}$. We prove the latter statement by obtaining a bound, polynomial in the size of the largest DFA, on the length of a shortest word accepted by all DFAs. Therefore, we show that for a minimal poNFA A, the size of an equivalent DFA is lower-bounded by the size of A and use a result of [28] for poNFAs. They have shown that given poNFAs $A_1, A_2, \ldots, A_m$, if the intersection of these automata is non-empty, then there exists a word of size at most $\sum_{i \in \{1,\ldots,m\}} d_i$, where d_i is the *depth* of A_i [28, Theorem 3.3]. Here, the depth of A_i is the length of the longest path (without self-loops) in the state graph of A_i. This result implies that the Intersection Non-emptiness problem for poNFAs accepting languages from $\mathcal{L}_{3/2}$ is contained in NP. We will further use this result to show that the Intersection Non-emptiness problem for DFAs accepting languages from $\mathcal{L}_{3/2}$ is NP-complete. First, we show that the number of states in a minimal poNFA is at most the number of classes in the Myhill-Nerode equivalence relation.

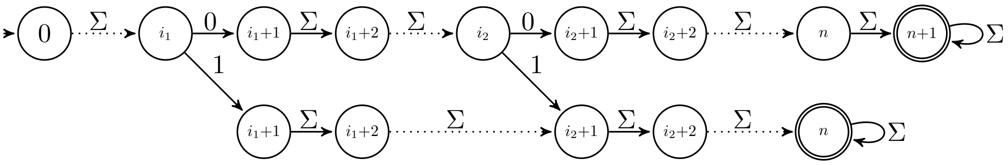


Figure 2 DFA A_{e_i} with $L(A_{e_i}) = \Sigma^{i_1} \cdot 1 \cdot \Sigma^{n-i_1-1} \cup \Sigma^{i_2} \cdot 1 \cdot \Sigma^{n-i_2-1} \cup \Sigma^{\geq n+1}$. A dotted arrow between some states j and j' represents a chain of length $j' - j$ with the same transition labels.

▶ **Lemma 12.** *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a minimal poNFA. Then, $L({}_{q_1}A) \neq L({}_{q_2}A)$ for all states $q_1, q_2 \in Q$, where ${}_qA$ is defined as $(Q, \Sigma, \delta, q, F)$.*

Now, we can use the result from Masopust and Krötzsch to prove that the Intersection Non-emptiness problem for DFAs accepting languages in $\mathcal{L}_{3/2}$ is in NP.

▶ **Lemma 13.** *The Intersection Non-emptiness problem for DFAs accepting languages from $\mathcal{L}_{3/2}$ belongs to* NP*.*

Proof. By Lemma 12, we have that the number of states in a minimal poNFA is at most the number of classes of the Myhill-Nerode equivalence relation. Hence, given a DFA accepting a language $L \in \mathcal{L}_{3/2}$, there exists a smaller poNFA that recognizes L. By [28, Theorem 3.3], if the intersection is not empty, then there is a certificate of polynomial size. ◀

4.2 NP-Hardness

Recall that by [34, Theorem 1] Intersection Non-emptiness for finite languages accepted by DFAs is already NP-complete. As the level $\mathcal{B}_0$ of the dot-depth hierarchy contains all finite language, the NP-hardness part of Theorem 7 follows directly from inclusion of language classes. Combining Lemma 13, and [28, Theorem 3.3] with the inclusion between levels in the Straubing-Thérien and the dot-depth hierarchy, we conclude the proof of Theorem 7.

▶ Remark 14. Recall that the dot-depth hierarchy, apart form $\mathcal{B}_0$, coincides with the concatenation hierarchy starting with the language class $\{\emptyset, \{\lambda\}, \Sigma^+, \Sigma^*\}$. The Intersection Non-emptiness problem for DFAs or NFAs accepting only languages from $\{\emptyset, \{\lambda\}, \Sigma^+, \Sigma^*\}$ belongs to AC^0, by similar arguments as in the proof of Theorem 2.

We showed in Section 3 that Intersection Non-emptiness for DFAs, all accepting languages from $\mathcal{L}_{1/2}$, belongs to L. If we allow only one DFA to accept a language from $\mathcal{L}_1$, the problem becomes NP-hard. The statement also holds if the common alphabet is binary.

▶ **Theorem 8.** *The Intersection Non-emptiness problem for DFAs is* NP*-complete even if only one DFA accepts a language from $\mathcal{L}_1$ and all other DFAs accept languages from $\mathcal{L}_{1/2}$ and the alphabet is binary.*

Proof sketch. The reduction is from Vertex Cover. Let $k \in \mathbb{N}_{\geq 0}$ and let $G = (V, E)$ be a graph with vertex set $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and edge set $E = \{e_0, e_1, \ldots, e_{m-1}\}$. The only words $w = a_0 a_1 \ldots a_\ell$ accepted by all DFAs will be of length exactly $n = \ell + 1$ and encode a vertex cover by: v_j is in the vertex cover if and only if $a_j = 1$. Therefore, we construct for each edge $e_i = \{v_{i_1}, v_{i_2}\} \in E$, with $i_1 < i_2$, a DFA A_{e_i}, as depicted in Figure 2, that accepts the language $L(A_{e_i}) = \Sigma^{i_1} \cdot 1 \cdot \Sigma^{n-i_1-1} \cup \Sigma^{i_2} \cdot 1 \cdot \Sigma^{n-i_2-1} \cup \Sigma^{\geq n+1}$. We show that $L(A_{e_i})$ is from $\mathcal{L}_{1/2}$, as it also accepts all words of length at least $n+1$. We further construct a DFA $A_{=n,\leq k}$ that accepts all words of length exactly n that contain at most k letters 1. The finite language $L(A_{=n,\leq k})$ is the only language from $\mathcal{L}_1$ in the instance. ◀

4.3 Large Partially Ordered NFAs

The results obtained in the last subsection left the precise complexity membership of Intersection Non-emptiness in the case of input automata being NFAs without any structural properties for the levels $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, and $\mathcal{L}_{3/2}$ open. We devote this subsection to the proof of Theorem 9, showing that already for languages of $\mathcal{B}_0$ being accepted by an NFA, the size of an equivalent minimal poNFA can be exponential in the size of the NFA.

▶ **Theorem 9.** *For every $n \in \mathbb{N}_{\geq 1}$, there exists a language $L_n \in \mathcal{B}_0$ on a binary alphabet such that L_n is recognized by an NFA of size $O(n^2)$, but the minimal poNFA recognizing L_n has more than 2^{n-1} states.*

Proof. While the statement requires languages over a binary alphabet, we begin by constructing an auxiliary family $(M_n)_{n\in\mathbb{N}_{\geq 1}}$ of languages over an unbounded alphabet. For all $n \in \mathbb{N}_{\geq 1}$ we then define L_n by encoding M_n with a binary alphabet, and we prove three properties of these languages that directly imply the statement of the Theorem.

For every $n \in \mathbb{N}_{\geq 1}$, we define the languages M'_n and M''_n over the alphabet $\{1, 2, \ldots, n\}$ as follows. The language M'_n contains all the words of odd length, and M''_n contains all the words in which there are two occurrences of some letter $i \in \{1, 2, \ldots, n\}$ with only letters smaller than i appearing in between.[3] Formally,

$$M'_n = \{\, x \in \{1, 2, \ldots, n\}^* \mid |x| \text{ is odd} \,\},$$
$$M''_n = \{\, xiyiz \in \{1, 2, \ldots, n\}^* \mid i \in \{1, 2, \ldots, n\}, y \in \{1, 2, \ldots, i-1\}^* \,\}.$$

We then define M_n as the union $M'_n \cup M''_n$. Moreover, we define L_n by encoding M_n with the binary alphabet $\{a, b\}$: Let us consider the function $\phi_n : \{1, 2, \ldots, n\}^* \to \{a, b\}^*$ defined by $\phi(i_1 i_2 \ldots i_m) = a^{i_1} b^{n-i_1} a^{i_2} b^{n-i_2} \ldots a^{i_m} b^{n-i_m}$. We set $L_n \subseteq \{a, b\}^*$ as the union of $\phi_n(M_n)$ with the language $\{a, b\}^* \setminus \phi(\{1, 2, \ldots, n\}^*)$ containing all the words that are not a proper encoding of some word in $\{1, 2, \ldots, n\}^*$.

The statement of the theorem immediately follows from the following claim

▷ Claim 15.
1. The languages M_n and L_n are cofinite, thus they are in $\mathcal{B}_0$.
2. The languages M_n and L_n are recognized by NFAs of size $n + 4$, resp. $O(n^2)$.
3. Every poNFA recognizing either M_n or L_n has a size greater than 2^{n-1}.

The formal proof of this claim is presented in the long version [3]. ◀

4.4 Commutative Star-Free Languages

In the case of commutative languages, we have a complete picture of the complexities for both hierarchies, even for arbitrary input NFAs. Observe, that commutative languages generalize unary languages, where it is known that for unary star-free languages both hierarchies collapse. For commutative star-free languages, a similar result holds, employing [18, Prop. 30].

▶ **Theorem 16.** *For* commutative star-free languages *the levels $\mathcal{L}_n$ of the Straubing-Thérien and $\mathcal{B}_n$ of the dot-depth hierarchy coincide for all full and half levels, except for $\mathcal{L}_0$ and $\mathcal{B}_0$. Moreover, the hierarchy collapses at level one.*

Next we will give the results, summarized in Theorem 10, for the case of the commutative (star-free) languages. The NP-hardness follows by a reduction from 3-CNF-SAT.

[3] The languages $(M''_n)_{n\in\mathbb{N}_{\geq 1}}$ were previously studied in [22] with a game-theoretic background. We also refer to [30] for similar "fractal languages."

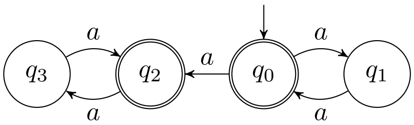


Figure 3 An example of a non-totally star-free NFA that accepts a star-free language.

▶ **Lemma 17.** *The* Intersection Non-emptiness *problem is* NP*-hard for DFAs accepting* commutative *languages in $\mathcal{L}_1$.*

The upper bound shown next also holds for arbitrary commutative languages.

▶ **Theorem 18.** *The* Intersection Non-emptiness *problem for NFAs accepting arbitrary, i.e., not necessarily star-free,* commutative *languages is in* NP*.*

Proof. It was shown in [38] that Intersection Non-emptiness is NP-complete for unary NFAs as input. Fix some order $\Sigma = \{a_1, a_2, \ldots, a_r\}$ of the input alphabet. Let $A_1, A_2, \ldots, A_m$ be the NFAs accepting commutative languages with $A_i = (Q_i, \Sigma, \delta_i, q_{0,i}, F_i)$ for $1 \leq i \leq m$. Without loss of generality, we may assume that every F_i is a singleton set, namely $F_i = \{q_{f,i}\}$. For each $1 \leq i \leq m$ and $1 \leq j \leq r$, let $B_{i,j}$ be the automaton over the unary alphabet $\{a_j\}$ obtained from A_i by deleting all transitions labeled with letters different from a_j and only retaining those labeled with a_j. Each $B_{i,j}$ will have one initial and one final state. Let $\vec{q_0} = (q_{0,1}, q_{0,2}, \ldots, q_{0,m})$ be the tuple of initial states of the NFAs; they are the initial states of $B_{1,1}, B_{2,1}, \ldots, B_{m,1}$, respectively. Then, nondeterministically guess further tuples $\vec{q_j}$ from $Q_1 \times Q_2 \times \ldots \times Q_m$ for $1 \leq j \leq r-1$. The jth tuple is considered as collecting the final states of the $B_{i,j}$ but also as the start states for the $B_{i,j+1}$. Finally, let $\vec{q_f} = (q_{f,1}, q_{f,2}, \ldots, q_{f,m})$ and consider this as the final states of $B_{1,r}, B_{2,r}, \ldots, B_{m,r}$. Then, for each $1 \leq j \leq r$ solve Intersection Non-emptiness for the unary automata $B_{1,j}, B_{2,j}, \ldots, B_{m,j}$. If there exist words w_j in the intersection of $L(B_{1,j}), L(B_{2,j}), \ldots, L(B_{m,j})$, for each $1 \leq j \leq r$, then, by commutativity, there exists one in $a_1^* a_2^* \cdots a_r^*$, namely, $w_1 w_2 \cdots w_m$, and so the above procedure finds it. Conversely, if the above procedure finds a word, this is contained in the intersection of the languages induced by the A_i's. ◀

For fixed alphabets, we have a polynomial-time algorithm, showing that the problem is in XP for alphabet size as a parameter, for a class of NFAs generalizing, among others, poNFAs and DFAs (accepting star-free languages). This is in contrast to the other results on the Intersection Non-emptiness problem in this paper. We say that an NFA $A = (Q, \Sigma, \delta, q_0, F)$ is *totally star-free*, if the language accepted by ${}_qA_p = (Q, \Sigma, \delta, q, \{p\})$ is star-free for any states $q, p \in Q$. For instance, partially ordered NFAs are totally star-free.

An example of a non-totally star-free NFA accepting a star-free language is given next. Consider the following NFA $A = (\{q_0, q_1, q_2, q_3\}, \delta, q_0, \{q_0, q_2\})$ with $\delta(q_0, a) = \{q_1, q_2\}$, $\delta(q_1, a) = \{q_0\}$, $\delta(q_2, a) = \{q_3\}$, and $\delta(q_3, a) = \{q_2\}$ that accepts the language $\{a\}^*$. The automaton is depicted in Figure 3. Yet, neither $L({}_{q_0}A_{q_0}) = \{aa\}^*$ nor $L({}_{q_0}A_{q_2}) = \{a\}\{aa\}^* \cup \{\varepsilon\}$ are star-free.
The proof of the following theorem uses classical results of Chrobak and Schützenberger [10,36].

▶ **Theorem 19.** *The* Intersection Non-emptiness *problem for totally star-free NFAs accepting* star-free commutative *languages, i.e., commutative languages in $\mathcal{L}_{3/2}$, is contained in* XP *(with the size of the alphabet as the parameter).*

▶ Remark 20. Note that Theorem 19 does not hold for arbitrary commutative languages concerning a fixed alphabet, but only for star-free commutative languages, since in the general case, the problem is NP-complete even for languages over a common unary alphabet [38].

5 PSPACE-Completeness

Here, we prove that even when restricted to languages from $\mathcal{B}_1$ or $\mathcal{L}_2$, Intersection Non-emptiness is PSPACE-complete, as it is for unrestricted DFAs or NFAs. We will profit from the close relations of Intersection Non-emptiness to the Non-universality problem for NFAs: Given an NFA A with input alphabet Σ, decide if $L(A) \neq \Sigma^*$. Conversely, we can also observe that Non-universality for NFAs is PSPACE-complete for languages from $\mathcal{B}_1$.

▶ **Theorem 21.** *The Intersection Non-emptiness problem for DFAs or NFAs accepting languages from $\mathcal{B}_1$ or $\mathcal{L}_2$ is PSPACE-complete, even for binary input alphabets.*

As $\mathcal{B}_1 \subseteq \mathcal{L}_2$, it is sufficient to show that the problem is PSPACE-hard for $\mathcal{B}_1$. While without paying attention to the size of the input alphabet, this result can be readily obtained by re-analyzing Kozen's original proof in [24], the restriction to binary input alphabets needs some more care. Details can be found in the long version [3]. We modify the proof of Theorem 3 in [25] that showed PSPACE-completeness for Non-universality for poNFAs (that characterize the level 3/2 of the Straubing-Thérien hierarchy). Also, it can be observed that the languages involved in the intersection are actually locally testable languages. Without giving details of definitions, we can therefore formulate:

▶ **Corollary 22.** *The Intersection Non-emptiness problem for DFAs or NFAs accepting locally testable languages is PSPACE-complete, even for binary input alphabets.*

By the proof of Theorem 3 in [25], also $\bigcup_i L_i$ belongs to $\mathcal{B}_1$, so that we can conclude:

▶ **Corollary 23.** *The Non-universality problem for NFAs accepting languages from $\mathcal{B}_1$ is PSPACE-complete, even for binary input alphabets.*

6 Conclusion and Open Problems

We have investigated how the increase in complexity within the dot-depth and the Straubing-Thérien hierarchies is reflected in the complexity of the Intersection Non-emptiness problem. We have shown the complexity of this problem is already completely determined by the very first levels of either hierarchy.

Our work leaves open some very interesting questions and directions of research. First, we were not able to prove containment in NP for the Intersection Non-emptiness problem when the input automata are allowed to be NFAs accepting a language in the level 3/2 or in the level 1 of the Straubing-Thérien hierarchy. Interestingly, we have shown that such containment holds in the case of DFAs, but have shown that the technique we have used to prove this containment does not carry over to the context of NFAs. In particular, to show this we have provided the first exponential separation between the state complexity of general NFAs and partially ordered NFAs. The most immediate open question is if Intersection Non-emptiness for NFAs accepting languages in $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, or $\mathcal{L}_{3/2}$ is complete for some level higher up in the polynomial-time hierarchy (PH), or if this case is already PSPACE-complete. Another tantalizing open question is whether one can capture the levels of PH in terms of the Intersection Non-emptiness problem when the input automata are assumed to accept languages belonging to levels of a sub-hierarchy of $\mathcal{L}_2$. Such sub-hierarchies have been considered for instance in [23].

It would also be interesting to have a systematic study of these two well-known subregular hierarchies for related problems like Non-universality for NFAs or Union Non-universality for DFAs. Notice the technicality that Union Non-universality (similar to Intersection Non-emptiness) has an implicit Boolean operation (now union instead of intersection) within the problem statement, while Non-universality lacks this implicit Boolean operation. This might lead to a small “shift” in the discussions of the hierarchy levels that involve Boolean closure. Another interesting hierarchy is the group hierarchy [32], where we start with the group languages, i.e., languages acceptable by automata in which every letter induces a permutation of the state set, at level 0. Note that for group languages, Intersection Non-emptiness is NP-complete even for a unary alphabet [38]. As Σ^* is a group language, the Straubing-Thérien hierarchy is contained in the corresponding levels of the group hierarchy, and hence, we get PSPACE-hardness for level 2 and above in this hierarchy. However, we do not know what happens in the levels in between.

References

1 Parosh Aziz Abdulla. Regular model checking. *International Journal on Software Tools for Technology Transfer*, 14(2):109–118, 2012.

2 Jorge Almeida and Ondrej Klíma. New decidable upper bound of the second level in the Straubing-Thérien concatenation hierarchy of star-free languages. *Discrete Mathematics & Theoretical Computer Science*, 12(4):41–58, 2010.

3 Emmanuel Arrighi, Henning Fernau, Stefan Hoffmann, Markus Holzer, Ismaël Jecker, Mateus de Oliveira Oliveira, and Petra Wolf. On the Complexity of Intersection Non-emptiness for Star-Free Language Classes. *CoRR*, abs/2110.01279, 2021. URL: `http://arxiv.org/abs/2110.01279`, `arXiv:2110.01279`.

4 Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.

5 Ahmed Bouajjani, Anca Muscholl, and Tayssir Touili. Permutation rewriting and algorithmic verification. *Information and Computation*, 205:199–224, 2007.

6 Janusz A. Brzozowski. Hierarchies of aperiodic languages. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 10(2):33–49, 1976.

7 Janusz A. Brzozowski and Faith E. Fich. Languages of $\mathcal{R}$-trivial monoids. *Journal of Computer and System Sciences*, 20(1):32–49, February 1980.

8 Janusz A. Brzozowski and Robert Knast. The dot-depth hierarchy of star-free languages is infinite. *Journal of Computer and System Sciences*, 16(1):37–55, 1978.

9 Sang Cho and Dung T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99–116, September 1991.

10 Marek Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47(3):149–158, 1986.

11 Rina S. Cohen and Janusz A. Brzozowski. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5(1):1–16, 1971.

12 Henning Fernau and Andreas Krebs. Problems on finite automata and the exponential time hypothesis. *Algorithms*, 10(1):24, 2017.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

14 Pawel Gawrychowski. Chrobak normal form revisited, with applications. In Béatrice Bouchou-Markhoff, Pascal Caron, Jean-Marc Champarnaud, and Denis Maurel, editors, *Implementation and Application of Automata - 16th International Conference, CIAA*, volume 6807 of *Lecture Notes in Computer Science*, pages 142–153. Springer, 2011.

15 Christian Glaßer and Heinz Schmitz. Decidable hierarchies of starfree languages. In Sanjiv Kapoor and Sanjiva Prasad, editors, *Foundations of Software Technology and Theoretical Computer Science, 20th Conference, FST TCS*, volume 1974 of *Lecture Notes in Computer Science*, pages 503–515. Springer, 2000.

16 Christian Glaßer and Heinz Schmitz. Level 5/2 of the Straubing-Thérien hierarchy for two-letter alphabets. In Werner Kuich, Grzegorz Rozenberg, and Arto Salomaa, editors, *Developments in Language Theory, 5th International Conference, DLT*, volume 2295 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

17 Juris Hartmanis, Neil Immerman, and Stephen R. Mahaney. One-way log-tape reductions. In *19th Annual Symposium on Foundations of Computer Science, FOCS*, pages 65–72. IEEE Computer Society, 1978.

18 Stefan Hoffmann. Regularity conditions for iterated shuffle on commutative regular languages. *accepted at CIAA*, 2021.

19 Harry B. Hunt III and Daniel J. Rosenkrantz. Computational parallels between the regular and context-free languages. *SIAM Journal on Computing*, 7(1):99–114, 1978.

20 George Karakostas, Richard J. Lipton, and Anastasios Viglas. On the complexity of intersecting finite state automata and NL versus NP. *Theoretical Computer Science*, 302(1):257–274, 2003.

21 Takumi Kasai and Shigeki Iwata. Gradually intractable problems and nondeterministic log-space lower bounds. *Mathematical Systems Theory*, 18(1):153–170, 1985.

22 Felix Klein and Martin Zimmermann. How much lookahead is needed to win infinite games? *Logical Methods in Computer Science*, 12(3), 2016. doi:10.2168/LMCS-12(3:4)2016.

23 Ondrej Klíma and Libor Polák. Subhierarchies of the second level in the straubing-thérien hierarchy. *International Journal of Algebra and Computation*, 21(7):1195–1215, 2011.

24 Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 254–266. IEEE Computer Society, 1977.

25 Markus Krötsch, Tomás Masopust, and Michaël Thomazo. Complexity of universality and related problems for partially ordered NFAs. *Information and Computation*, 255:177–192, 2017.

26 Klaus-Jörn Lange and Peter Rossmanith. The emptiness problem for intersections of regular languages. In Ivan M. Havel and Václav Koubek, editors, *Mathematical Foundations of Computer Science 1992, 17th International Symposium, MFCS*, volume 629 of *Lecture Notes in Computer Science*, pages 346–354. Springer, 1992.

27 Tomás Masopust. Separability by piecewise testable languages is PTime-complete. *Theoretical Computer Science*, 711:109–114, 2018.

28 Tomás Masopust and Markus Krötzsch. Partially ordered automata and piecewise testability. *CoRR*, abs/1907.13115, 2019. arXiv:1907.13115.

29 Tomás Masopust and Michaël Thomazo. On the complexity of k-piecewise testability and the depth of automata. In Igor Potapov, editor, *Developments in Language Theory - 19th International Conference, DLT*, number 9168 in Lecture Notes in Computer Science, pages 364–376. Springer, 2015.

30 Mike Naylor. Abacaba! – using a mathematical pattern to connect art, music, poetry and literature. *Bridges*, pages 89–96, 2011.

31 Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

32 Jean-Éric Pin. Bridges for concatenation hierarchies. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 431–442. Springer, 1998.

33 Thomas Place and Marc Zeitoun. Generic results for concatenation hierarchies. *Theory of Computing Systems*, 63(4):849–901, 2019.

34 Narad Rampersad and Jeffrey Shallit. Detecting patterns in finite regular and context-free languages. *Information Processing Letters*, 110(3):108–112, 2010.

35 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

36 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

37 Thomas Schwentick, Denis Thérien, and Heribert Vollmer. Partially-ordered two-way automata: A new characterization of DA. In Werner Kuich, Grzegorz Rozenberg, and Arto Salomaa, editors, *Developments in Language Theory, 5th International Conference, DLT*, volume 2295 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2001.

38 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *5th Annual Symposium on Theory of Computing, STOC*, pages 1–9. ACM, 1973.

39 Howard Straubing. A generalization of the Schützenberger product of finite monoids. *Theoretical Computer Science*, 13:137–150, 1981.

40 Howard Straubing. Finite semigroup varieties of the form V^*D. *Journal of Pure and Applied Algebra*, 36:53–94, 1985.

41 Ivan Hal Sudborough. On tape-bounded complexity classes and multihead finite automata. *Journal of Computer and System Sciences*, 10(1):62–76, February 1975.

42 Denis Thérien. Classification of finite monoids: the language approach. *Theoretical Computer Science*, 14(2):195–208, 1981.

43 Todd Wareham. The parameterized complexity of intersection and composition operations on sets of finite-state automata. In Sheng Yu and Andrei Paun, editors, *Implementation and Application of Automata, 5th International Conference, CIAA*, volume 2088 of *Lecture Notes in Computer Science*, pages 302–310. Springer, 2000.

44 Michael Wehar. Hardness results for intersection non-emptiness. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 354–362. Springer, 2014.

45 Michael Wehar. *On the Complexity of Intersection Non-Emptiness Problems*. PhD thesis, University at Buffalo, 2016.

Complexity of Coverability in Bounded Path Broadcast Networks

A. R. Balasubramanian
Technische Universität München, Germany

Abstract

Broadcast networks are a formalism of distributed computation that allow one to model networks of identical nodes communicating through message broadcasts over a communication topology that does not change over the course of executions. The parameterized verification problem for these networks amounts to proving correctness of a property for any number of nodes, and on all executions. Dually speaking, this problem asks for the existence of an execution of the broadcast network that violates a given property. One specific instance of parameterized verification is the coverability problem which asks whether there is an execution of the network in which some node reaches a given state of the broadcast protocol. This problem was proven to be undecidable by Delzanno, Sangnier and Zavattaro (CONCUR 2010). In the same paper, the authors also prove that, if we additionally assume that the underlying communication topology has a bound on the longest path, then the coverability problem becomes decidable.

In this paper, we provide complexity results for the above problem and prove that the coverability problem for bounded-path topologies is $\mathbf{F}_{\epsilon_0}$-complete, where $\mathbf{F}_{\epsilon_0}$ is a class in the fast-growing hierarchy of complexity classes. This solves an open problem of Hasse, Schmitz and Schnoebelen (LMCS, Vol 10, Issue 4).

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Distributed computing models

Keywords and phrases Parameterized verification, Bounded path networks, Fast-growing complexity classes

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.35

Funding *A. R. Balasubramanian*: Supported by funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS).

1 Introduction

In recent years, significant effort has been put into understanding the precise computational complexity of problems which are *non-elementary*, i.e., problems whose running times cannot be upper bounded by any fixed tower of exponentials of the input size [13, 6, 20, 19, 1, 18, 8]. A well-known such problem is the satisfiability problem of the weak monadic theory of one successor (WS1S) [17]. A more recent addition to this collection is the reachability problem for Petri nets [7]. We refer the reader to the excellent survey by Schmitz [19] for a collection of various non-elementary problems from logic, automata theory and verification which have been proven to be complete for appropriate complexity classes in the *fast-growing hierarchy*. This hierarchy allows for a finer classification of problems lying beyond the elementary regime.

From a tractability perspective, these results are of course negative. However, there are non-elementary problems for which tools have been developed, for e.g. MONA for WS1S [11]; and considerable effort has been put into the development of fast heuristics to solve some non-elementary problems on realistic inputs, for e.g., there is a huge wealth of heuristics and special cases which have been studied for the Petri net reachability problem [3, 12, 14, 4, 5, 15]. Hence, understanding the precise complexity of a non-elementary problem can help us to solve it in practice by reducing it to various other well-studied non-elementary problems.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 35; pp. 35:1–35:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The fast-growing hierarchy mentioned above can help us in this goal of understanding the computational complexity of non-elementary problems. Proving a problem to be hard for one of these classes implies that that problem cannot have an efficient encoding into any of the non-elementary problems which lie strictly below this class. In their invited paper for CONCUR 2013 [21], Schmitz and Schnoebelen explicity state the program of populating the catalog of hard problems for classes in the fast-growing hierarchy, so that hardness proofs do not have to begin from Turing machines, but can instead rely on simpler reductions.

In this paper, we contribute to this program by considering a problem from the parameterized verification of broadcast networks and proving that it is $\mathbf{F}_{\epsilon_0}$-complete, where $\mathbf{F}_{\epsilon_0}$ is a complexity class in the fast-growing hierarchy. We now offer a brief overview of broadcast networks [9, 2]. Broadcast networks are a formalism of distributed computation that allow one to model networks of *identical nodes* communicating through message broadcasts. Each node runs the same protocol and an underlying communication topology specifies for each node, the set of neighbors that it can broadcast messages to. This topology remains invariant over the course of executions of the network. At any point, a node can broadcast a message which is received by all of its neighbors.

The parameterized verification problem for these networks amounts to proving correctness of a property for any number of nodes and over any communication topology. Dually, we ask for the existence of an execution of the network that violates a given property. One specific instance of parameterized verification is the coverability problem which asks whether there is an execution of the network in which some node reaches a given state of the broadcast protocol. This problem was proven to be undecidable by Delzanno, Sangnier and Zavattaro (Theorem 1 of [9]). In the same paper, the authors also prove that, if we additionally assume that the underlying communication topology has a bound on the longest path (bounded-path topologies), then the coverability problem becomes decidable (Theorem 5 of [9]). Our main result in this paper is that the coverability problem for bounded-path topologies is $\mathbf{F}_{\epsilon_0}$-complete, where $\mathbf{F}_{\epsilon_0}$ is a class in the aforementioned fast-growing hierarchy of complexity classes.

Our result settles a conjecture raised by Hasse, Schmitz and Schnoebelen (Section 8.3 of [16]) and also settles the complexity of the last remaining question from the original paper that initiated the study of parameterized verification problems for broadcast networks [9]. Moreover, we provide a new and rather natural problem to the list of $\mathbf{F}_{\epsilon_0}$-complete problems, which when compared to the list of $\mathbf{F}_{\omega}$-complete and $\mathbf{F}_{\omega^\omega}$-complete problems, is rather small currently (Section 6.4 of [19]). (Both $\mathbf{F}_{\omega}$ and $\mathbf{F}_{\omega^\omega}$ are classes in the fast-growing hierarchy which are much smaller than $\mathbf{F}_{\epsilon_0}$). Hence, in this sense, we contribute to the above-mentioned program of finding hard problems for classes in the fast-growing hierarchy. Further, we hope that the present work might prove to be useful in settling the complexity of other problems conjectured to be $\mathbf{F}_{\epsilon_0}$-complete (Section 8.3 of [16]), since all the problems mentioned there are concerned with infinite-state systems regarding bounded-path trees and graphs, and so those problems are in some sense "close" to the problem that we consider here.

2 Preliminaries

In this section, we recall the model of broadcast networks as defined in [2]. Intuitively, a broadcast network consists of several nodes, each executing the same finite-state *broadcast protocol.* A communication topology assigns to each node, a finite set of neighbors, to which it can communicate. At any point, some node can broadcast a message which is received by all of its neighbors. We now proceed to formalize this intuition.

Broadcast networks

▶ **Definition 1.** *A broadcast protocol is a tuple $\mathcal{P} = (Q, I, \Sigma, \Delta)$ where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, Σ is a finite set of messages and $\Delta \subseteq Q \times \{!a, ?a, : a \in \Sigma\} \times Q$ is the transition relation.*

For ease of notation, we will write $q \xrightarrow{!a} q'$ (resp. $q \xrightarrow{?a} q'$) for $(q, !a, q') \in \Delta$ (resp. $(q, ?a, q') \in \Delta$). A transition $q \xrightarrow{!a} q'$ (resp. $q \xrightarrow{?a} q'$) intuitively corresponds to broadcasting (resp. receiving) the message a. We will assume that broadcast protocols are complete, i.e. for every state q and every message a there exists q' such that $q \xrightarrow{?a} q'$.

As mentioned before, a broadcast network consists of several identical *nodes* running a broadcast protocol and each node has a finite set of neighbors. To formalize this, given a broadcast protocol $\mathcal{P} = (Q, I, \Sigma, \Delta)$, a *configuration* of $\mathcal{P}$ is a labelled graph $\gamma = (\mathsf{N}, \mathsf{E}, \mathsf{L})$ where N is a finite set of nodes, $\mathsf{E} \subseteq \mathsf{N} \times \mathsf{N}$ is a finite set of (undirected) edges specifying for every pair of nodes whether or not there is a communication link between them and $\mathsf{L} : \mathsf{N} \to Q$ is a labelling function that specifies the current state of each node. We let $\mathsf{L}(\gamma) = \{\mathsf{L}(\mathsf{n}) : \mathsf{n} \in \mathsf{N}\}$ be the set of labels appearing in the nodes of γ. We say that γ is *initial* if $\mathsf{L}(\gamma) \subseteq I$.

The semantics of the broadcast network of a protocol $\mathcal{P}$ is given by means of an infinite-state transition system $\mathcal{T}(\mathcal{P})$ which consists of all the configurations of the protocol $\mathcal{P}$. There is a step from the configuration $\gamma = (\mathsf{N}, \mathsf{E}, \mathsf{L})$ to the configuration $\gamma' = (\mathsf{N}', \mathsf{E}', \mathsf{L}')$ if $\mathsf{N}' = \mathsf{N}$, $\mathsf{E}' = \mathsf{E}$ and there exists a node n and a message $a \in \Sigma$ such that $(\mathsf{L}(\mathsf{n}), !a, \mathsf{L}'(\mathsf{n})) \in \Delta$, and for every other node n', if $(\mathsf{n}, \mathsf{n}') \in \mathsf{E}$, then $(\mathsf{L}(\mathsf{n}), ?a, \mathsf{L}'(\mathsf{n}')) \in \Delta$; otherwise $\mathsf{L}(\mathsf{n}') = \mathsf{L}'(\mathsf{n}')$. In this case, we write $\gamma \xrightarrow{\mathsf{n},a} \gamma'$ or simply $\gamma \to \gamma'$. Intuitively, a step consists of a node n broadcasting some message a which is then received by *all* of its neighbors; all the other nodes do nothing. Notice that between steps, the set of nodes and edges do not change.

A *run* from the configuration γ to the configuation γ' is a sequence of steps $\gamma \to \gamma_1 \to \gamma_2 \to \dots \gamma_{k-1} \to \gamma'$. If a run exists between configurations γ and γ' we denote it by $\gamma \xrightarrow{*} \gamma'$. An *execution* is a run starting from an initial configuration.

Given a state f and a configuration γ we say that γ covers f if $f \in \mathsf{L}(\gamma)$, i.e., if the state of some node in γ is f. We say that an execution $\gamma_0 \xrightarrow{*} \gamma$ covers f, if γ covers f. The *coverability* problem for broadcast protocols is to decide, given a broadcast protocol $\mathcal{P}$ and a state f, whether there is an execution from some initial configuration that covers f. It is known that the coverability problem is undecidable (Theorem 1 of [9]).

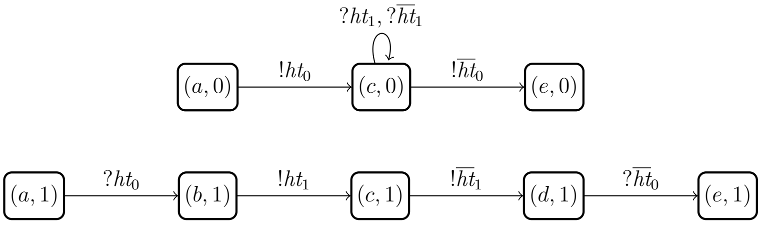


Figure 1 Example of a broadcast protocol where we set $I = \{(a,0),(a,1)\}$ and $\Sigma = \{ht_i, \overline{ht}_i : 0 \leq i \leq 1\}$. If for a state (f,i), we have not depicted what happens when message m is received at (f,i), we assume that $(f,i) \xrightarrow{?m} (\bot, i)$. Here $(\bot, 0)$ and $(\bot, 1)$ are new *sink* states, i.e., states with no outgoing transition.

▶ **Example 2.** We consider the broadcast protocol given in Figure 1. Figure 2 shows an execution in this protocol covering the state $(e,0)$. Moreover, let $\gamma = (\mathsf{N}, \mathsf{E}, \mathsf{L})$ be *any* initial configuration and $\gamma' = (\mathsf{N}', \mathsf{E}', \mathsf{L}')$ be *any* configuration covering $(e,0)$ such that $\gamma \xrightarrow{*} \gamma'$.

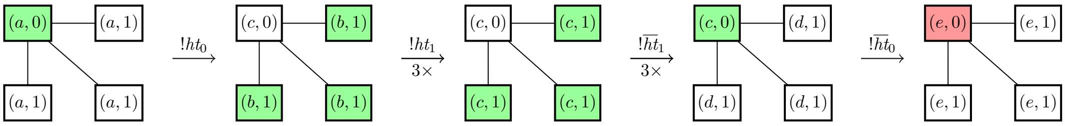


Figure 2 Example of an execution covering $(e,0)$ in the broadcast protocol given in Figure 1. The nodes marked in green make the broadcasts, i.e., first the node on the topmost left broadcasts ht_0, then all the other nodes broadcast ht_1 in some order, and then $\overline{ht}_1$ in some order, and then the node on the topmost left broadcasts $\overline{ht}_0$.

Hence, there is a node n such that $\mathsf{L}'(\mathsf{n}) = (e,0)$. Note that $\mathsf{L}(\mathsf{n})$ must be $(a,0)$. Hence n must have broadcasted both ht_0 and $\overline{ht}_0$ to move into the states $(c,0)$ and $(e,0)$ at different points during the run. This means that all of the neighbors of n received $\overline{ht}_0$ at some point, and so the labels of all of its neighbors in γ' must be either $(e,1)$ or $(\bot,0)$ or $(\bot,1)$.

Suppose n' is a neighbor of n such that $\mathsf{L}'(\mathsf{n}') = (e,1)$. Notice that if there is a neighbor $\mathsf{n}'' \neq \mathsf{n}$ of n' which was at $(c,0)$ during some point in the run, then n'' must have broadcasted ht_0 during the run. However, then n' would have received two ht_0 messages, which would have caused it to move into either $(\bot,0)$ or $(\bot,1)$. Hence, there is exactly one neighbor of n' which was labelled by $(c,0)$ at some point during the run.

This protocol along with the above discussion will prove useful later on for the lower bound reductions in section 5.

Bounded-path broadcast networks

Motivated by the undecidability of the coverability problem, the authors of [9] also study a different variant of the problem, which we now describe.

Let $\mathcal{P}$ be a broadcast protocol and let $k \geq 1$ be some number. Let γ be a configuration of $\mathcal{P}$. We say that γ is k-path bounded if the length of the *longest* simple path in γ is at most k. Now, let $\mathcal{T}_k(\mathcal{P})$ be the restriction of the transition system $\mathcal{T}(\mathcal{P})$ to only k-path bounded configurations. Notice that since the set of nodes and edges do not change during a run, $\mathcal{T}_k(\mathcal{P})$ is closed under the step relation. The *path bounded* coverability problem (Bounded-Path-Cover) is then defined as follows:

Given: A broadcast protocol $\mathcal{P} = (Q, I, \Sigma, \Delta)$, a state $f \in Q$ and a number k.
Decide: If there is an execution in $\mathcal{T}_k(\mathcal{P})$ which covers f.

The authors of [9] prove that this problem is decidable (Theorem 5 of [9]). The main result that we prove in this paper is that

▶ **Theorem 3.** *Bounded-Path-Cover is $\mathbf{F}_{\epsilon_0}$-complete.*

Here $\mathbf{F}_{\epsilon_0}$ is a member of the *fast-growing* complexity class hierarchy. We refer the reader to Section 2.3 of [19] for a description of the fast-growing hierarchy and the class $\mathbf{F}_{\epsilon_0}$. To prove the upper bound for our problem, we will consider the algorithm given in [9] and analyze its running time by means of *controlled-bad* sequences of a suitable well-quasi order, whose upper bounds will allow us to place Bounded-Path-Cover in the complexity class $\mathbf{F}_{\epsilon_0}$. The lower bound is proved by giving a logspace reduction from a known $\mathbf{F}_{\epsilon_0}$-hard problem, which we now proceed to describe.

3 Nested counter systems (NCS)

A nested counter system is a generalisation of a usual counter system with *higher-order counters*, i.e., counters which can themselves contain other (lower-order) counters. Intuitively, an one-dimensional counter is a usual counter, which can either add or subtract 1. A two-dimensional counter can either add or remove an one-dimensional counter, a three-dimensional counter can either add or remove a two-dimensional counter and so on. Here, we slightly alter the definition of nested counter systems as given in [8] so that it better suits our purposes. It can be easily verified that our altered definition does not affect the semantics of the system as given in [8].

A k-nested counter system (k-NCS) is a tuple $\mathcal{N} = (Q, \delta)$ where Q is a finite set of *states* and $\delta \subseteq \bigcup_{1 \leq i,j \leq k+1} (Q^i \times Q^j)$ is a set of *rules*. The set $\mathcal{C}_\mathcal{N}$ of *configurations* of $\mathcal{N}$ is defined to be the set of all labelled rooted trees of height atmost k, with labels from the set Q.

The operational semantics of $\mathcal{N}$ is defined in terms of the following transition relation $\rightarrow \subseteq \mathcal{C}_\mathcal{N} \times \mathcal{C}_\mathcal{N}$ on configurations: Let $r := ((q_0, \dots, q_i), (q'_0, \dots, q'_j)) \in \delta$ be a rule with $i \leq j \leq k$. We say that a configuration C can move to the configuration C' using the rule r (denoted by $C \xrightarrow{r} C'$), if there is a path $v_0, v_1 \dots, v_i$ in C starting at the root such that for every $0 \leq l \leq i$, the label of v_l is q_l and, C' is obtained from C by 1) for every $0 \leq l \leq i$, changing the label of each v_l to q'_l and 2) for every $i+1 \leq l \leq j$, creating a new vertex v_l with label q'_l and adding it as a child to v_{l-1}.

Similarly, suppose $r := ((q_0, \dots, q_i), (q'_0, \dots, q'_j)) \in \delta$ is a rule with $j < i \leq k$. Then $C \xrightarrow{r} C'$ if there is a path $v_0, v_1, \dots, v_i$ in C starting at the root such that for every $0 \leq l \leq i$, the label of v_l is q_l and, C' is obtained from C by 1) for every $0 \leq l \leq j$, changing the label of each v_l to q'_l and 2) removing the subtree rooted at the node v_{j+1}.

▶ **Example 4.** Let us consider the NCS $\mathcal{N}$ given by the states $Q = \{p_i, p'_i, q_i, q'_i : 0 \leq i \leq 4\}$ and consisting of the following rules: $r_1 = ((q_0, q_1), (q'_0, q'_1, q'_2)), r_2 = ((q'_0, q_3, q_2), (p_0)), r_3 = ((p_0), (p'_0))$. In Figure 3, we illustrate the application of these rules to a configuration of $\mathcal{N}$.

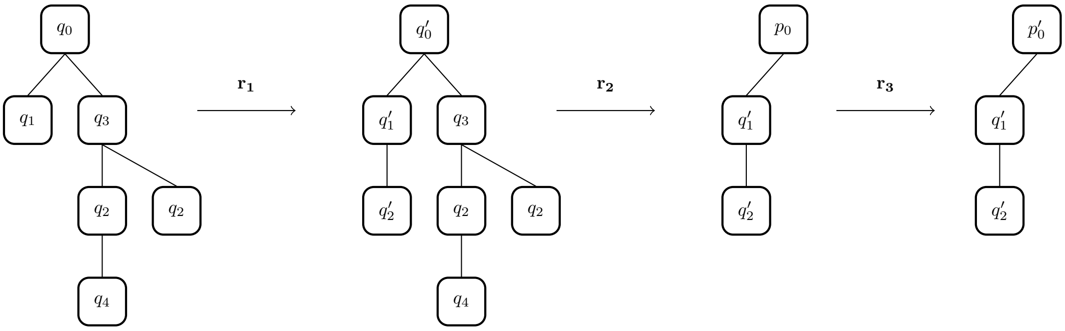


Figure 3 Application of the rules r_1, r_2 and r_3 to a configuration of $\mathcal{N}$, which is described in Example 4.

We say that $C \rightarrow C'$ if $C \xrightarrow{r} C'$ for some rule r. We let $\xrightarrow{*}$ denote the reflexive and transitive closure of $\rightarrow$ and we say that a configuration C reaches C' if $C \xrightarrow{*} C'$. Given two states $q_{in}, q_f \in Q$, we say that q_{in} can cover q_f if the (unique) configuration consisting of the single root vertex labelled with q_{in} can reach *some* configuration where the root is labelled by q_f. The coverability problem for an NCS is then the following: Given an NCS $\mathcal{N}$ and two states q_{in}, q_f, can q_{in} cover q_f? It is known that the coverability problem is $\mathbf{F}_{\epsilon_0}$-hard (Theorem 7 of [8]).

Lossy semantics. In addition to the "usual" semantics of an NCS that we have described in the previous section, we also need a *lossy semantics* which we now define here. Let $\mathcal{N} = (Q, \delta)$ be a k-NCS and let $q_{in}, q_f \in Q$. We say that there is a *lossy step* between configurations C and C', if C' can be obtained from C by deleting the subtree rooted at some vertex v in C. We let $C \dashrightarrow C'$ if either there is a lossy step between C and C' or $C \xrightarrow{r} C'$ for some rule r. As usual, we let $\overset{*}{\dashrightarrow}$ denote the reflexive and transitive closure of $\dashrightarrow$ and we say that C can reach C' in a lossy manner if $C \overset{*}{\dashrightarrow} C'$. We can then define the notion of the state q_{in} covering the state q_f in a straightforward manner.

For configurations C, C', we say that $C \geq C'$ iff C' can be obtained from C by a sequence of lossy steps. Since NCS do not have any zero tests, from the definition of the transition relation, we can easily infer the following proposition.

▶ **Proposition 5.** *If $C_1 \geq C_1'$ and $C_1' \overset{*}{\dashrightarrow} C_2'$ then there exists $C_2 \geq C_2'$ such that $C_1 \xrightarrow{*} C_2$.*

Hence, we get the following corollary.

▶ **Corollary 6.** *q_f can be covered from q_{in} in a lossy manner iff q_f can be covered from q_{in} under the usual semantics.*

This corollary will be useful later on in order to prove our hardness result.

4 A simulator protocol $\mathcal{P}_{\text{sim}}$

Throughout this section, let $\mathcal{N} = (Q, \delta)$ be a fixed k-NCS with two fixed states q_{in} and q_f. In this section, we will construct a broadcast protocol $\mathcal{P}_{\text{sim}} = (Q_{\text{sim}}, I_{\text{sim}}, \Sigma_{\text{sim}}, \delta_{\text{sim}})$, a state p of $\mathcal{P}_{\text{sim}}$, and define a notion of *good initial configurations* of $\mathcal{P}_{\text{sim}}$ such that the following property is satisfied: q_f can be covered from q_{in} in the NCS $\mathcal{N}$ iff p can be covered in $\mathcal{T}_{2k}(\mathcal{P}_{\text{sim}})$ by some execution starting at a *good initial configuration.* Intuitively, the protocol $\mathcal{P}_{\text{sim}}$ will *simulate* the NCS $\mathcal{N}$, *provided* that the initial configuration that it begins with is a good initial configuration.

States, alphabet and good configurations. For each $0 \leq i \leq k$, $\mathcal{P}_{\text{sim}}$ will have two states $(\textit{start}, i), (\textit{finish}, i)$. For each $0 \leq i \leq k$ and each $r \in \delta$, we will have five states $(\texttt{req-rec}[r], i), (\texttt{req-fwd}[r], i), (\texttt{wait}[r], i), (\texttt{ack-rec}[r], i), (\texttt{ack-fwd}[r], i)$. Finally, for each $0 \leq i \leq k$ and each $q \in Q$, $\mathcal{P}_{\text{sim}}$ will have a state (q, i). Notice that each state of $\mathcal{P}_{\text{sim}}$ is of the form (a, b) where $a \in Q \cup \{\textit{start}, \textit{finish}\} \cup \{\texttt{req-rec}[r], \texttt{req-fwd}[r], \texttt{wait}[r], \texttt{ack-rec}[r], \texttt{ack-fwd}[r]\}$ and $0 \leq b \leq k$. The first part "a" will be called the *base* of the state and the second part "b" will be called the *grade*. Sometimes we will abuse notation and refer to the base (resp. grade) of a node in a configuration to mean the base (resp. grade) of the label of that node.

The initial set of states I_{sim} will be the set $\{(q_{in}, 0)\} \cup \{(\textit{start}, i) : 1 \leq i \leq k\}$. (The asymmetry in the initial set of states between the case of 0 and others will be discussed in the following paragraphs). The alphabet Σ_{sim} will be the set $\{\textit{begin}_i^r, \textit{end}_i^r : r \in \delta, 0 \leq i \leq k\}$.

A configuration γ of $\mathcal{P}_{\text{sim}}$ is called *good* if γ is a tree of height at most k such that 1) the base of the label of every node is in the set $Q \cup \{\textit{start}, \textit{finish}\}$, 2) there is exactly one node n labelled by a state of grade 0, which will be called the root of γ and, 3) every node at distance i from n is labelled by a state of grade i. Notice that if γ is a good initial configuration then $\gamma \in \mathcal{T}_{2k}(\mathcal{P})$. Further, notice that in a good initial configuration, the root must be labelled by $(q_{in}, 0)$ and every node at distance i from the root is labelled by $(\textit{start}, i)$.

Intuition behind good configurations of $\mathcal{P}_{\text{sim}}$. Before we describe the transition relation of $\mathcal{P}_{\text{sim}}$, we describe some intuition behind the notion of a good configuration.

Let γ be a good configuration of $\mathcal{P}_{\text{sim}}$. Notice that there is a way to map γ to a configuration of $\mathcal{N}$: First, forget all the grades from the labels of each node in γ and just keep the bases. Next, remove all nodes whose label is either *start* or *finish* and from the resulting forest, pick the tree T containing the root. In this way, to every good configuration γ of $\mathcal{P}_{\text{sim}}$ we can define a configuration $\mathbb{E}(\gamma)$ of $\mathcal{N}$. Hence, we can use good configurations of $\mathcal{P}_{\text{sim}}$ to encode configurations of $\mathcal{N}$ and this is the reason behind defining good configurations of $\mathcal{P}_{\text{sim}}$. An example of this mapping is given in Figure 4.

Further, notice that if γ is any good initial configuration, then $\mathbb{E}(\gamma)$ is the initial configuration of $\mathcal{N}$. This is the reason behind the asymmetry in the definition of the initial set of states between the case of 0 and others.

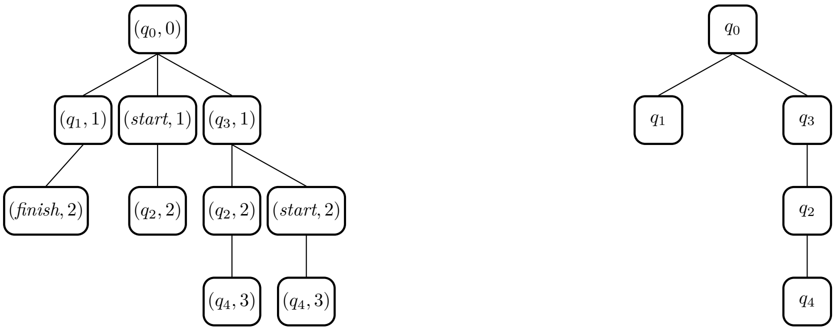


Figure 4 An example of the map $\mathbb{E}$ between good configurations of $\mathcal{P}$ and configurations of $\mathcal{N}$. On the left is a good configuration γ of $\mathcal{P}$ and on the right is its corresponding mapped configuration $\mathbb{E}(\gamma)$ of $\mathcal{N}$.

4.1 Transitions involving the letters $begin_i^r$ and end_i^r

For the rest of this section, let us fix a rule $r = ((q_0, \ldots, q_i), (q'_0, \ldots, q'_j)) \in \delta$ where $i, j \leq k$ and let $w = \max(i, j)$. For the sake of uniformity, if $i < j$, then let $q_l = start$ for every $i < l \leq j$. If $i > j$, then let $q'_l = finish$ for every $j < l \leq i$.

Intuitively, the gadget that we will demonstrate will use the messages $begin_i^r$ and end_i^r to find a path $\mathsf{n}_0, \ldots, \mathsf{n}_w$ labelled by $(q_0, 0), (q_1, 1), \ldots, (q_w, w)$ and then change the labels along this path to $(q'_0, 0), (q'_1, 1), \ldots, (q'_w, w)$. Notice that if $i \leq j$, this means that a path of the form $(q_0, 0), \ldots, (q_i, i), (start, i+1), \ldots, (start, j)$ becomes $(q'_0, 0), \ldots, (q'_i, i), (q'_{i+1}, i+1), \ldots, (q'_j, j)$. Similarly, if $i > j$ then a path of the form $(q_0, 0), \ldots, (q_j, j), \ldots (q_i, i)$ becomes $(q'_0, 0), \ldots, (q'_j, j), (finish, j+1), \ldots, (finish, i)$. This would then allow us to simulate the rule r on good configurations of $\mathcal{P}_{\text{sim}}$.

Formally, we now describe the transitions involving the letters $\{begin_i^r, end_i^r : 0 \leq i \leq k\}$. First, we make a small remark:

▶ Remark 7. In the following, if we do not specify what happens upon receiving a message m from a state with base a and grade b, then it is to be assumed that $(a, b) \xrightarrow{?m} (finish, b)$.

The "gadget" for "simulating" the rule r. We now present the main transitions involving the messages $begin_i^r$ and end_i^r.

- First, we have four transitions

$$(q_0, 0) \xrightarrow{!begin^r_0} (\texttt{req-fwd}[r], 0) \xrightarrow{?begin^r_1} (\texttt{wait}[r], 0) \xrightarrow{?end^r_1} (\texttt{ack-rec}[r], 0) \xrightarrow{!end^r_0} (q'_0, 0)$$

- Then, for every $1 \leq l \leq w - 1$, we have

$$(q_l, l) \xrightarrow{?begin^r_{l-1}} (\texttt{req-rec}[r], l) \xrightarrow{!begin^r_l} (\texttt{req-fwd}[r], l) \xrightarrow{?begin^r_{l+1}} (\texttt{wait}[r], l) \xrightarrow{?end^r_{l+1}}$$

$$(\texttt{ack-rec}[r], l) \xrightarrow{!end^r_l} (\texttt{ack-fwd}[r], l) \xrightarrow{?end^r_{l-1}} (q'_l, l)$$

- Finally, we have four transitions

$$(q_w, w) \xrightarrow{?begin^r_{w-1}} (\texttt{req-rec}[r], w) \xrightarrow{!begin^r_w} (\texttt{wait}[r], w) \xrightarrow{!end^r_w} (\texttt{ack-fwd}[r], w) \xrightarrow{?end^r_{w-1}} (q'_w, w)$$

Self-loops. While the previous gadget comprised the main transitions involving $begin^r_i$ and end^r_i, for technical reasons we need the following self-loop transitions as well: For every state with base $a \in Q \cup \{start, finish\}$ and grade $1 \leq i \leq k$, there are two transitions $(a, i) \xrightarrow{?begin^r_{i-1}} (a, i)$ and $(a, i) \xrightarrow{?end^r_{i-1}} (a, i)$.

This finishes our description of the transition relation of $\mathcal{P}_{\text{sim}}$.

Intuition behind the transitions. We now give a brief intuition behind the gadget in the case of $w = 2$. Notice that only the root n_0 in a good configuration can be labelled by $(q_0, 0)$. Hence if n_0 broadcasts $begin^r_0$, it is *forwarding* its request of wanting to simulate the rule r to its children. The children have two choices: either stay where they are by means of the self-loops or *receive* the request and move to $(\texttt{req-rec}[r], 1)$. Atleast one child n_1 has to receive the request and move, otherwise the configuration enters into a deadlock. From $(\texttt{req-rec}[r], 1)$ n_1 can *forward* this request to its children by broadcasting $begin^r_1$ (and also let n_0 know that is has received its request, whereby it enters a waiting mode). Notice that if two children of n_0 forward the request, then n_0 will enter $(finish, 0)$ and the simulation of the rule r cannot happen. Similarly, some child n_2 of n_1 must receive the request of n_1, move to $(\texttt{req-rec}[r], 2)$, then broadcast $begin^r_2$. At this point, the base of each n_i is $\texttt{wait}[r]$.

Now n_2 can broadcast end^r_2, forwarding an *acknowledgment* to the request made by n_1. n_1 can receive this acknowledgment and broadcast end^r_1, forwarding an acknowledgment to n_0 which can broadcast end^r_0 and move to $(q'_0, 0)$. At this point, the labels of $\mathsf{n}_0, \mathsf{n}_1$ and n_2 are $(q'_0, 0), (q'_1, 1)$ and $(q'_2, 2)$ respectively, which means that we have changed the labels along a path from $(q_0, 0), (q_1, 1)$ and $(q_2, 2)$ to $(q'_0, 0), (q'_1, 1)$ and $(q'_2, 2)$.

4.2 Proof of correctness

The following lemma tells us that we can use good configurations of $\mathcal{P}_{\text{sim}}$ along with the gadget for the rule r described in the previous section to simulate steps of $\mathcal{N}$.

▶ **Lemma 8** ($\mathcal{P}_{\mathsf{sim}}$ simulates $\mathcal{N}$). *Suppose $C \xrightarrow{r} C'$ is a step in the NCS $\mathcal{N}$. Suppose γ is a good configuration such that 1) $\mathbb{E}(\gamma) = C$ and, 2) there is a path $\mathsf{n}_0, \ldots, \mathsf{n}_w$ in γ where the label of each n_l is (q_l, l). Then there is a good configuration γ' with $\gamma \xrightarrow{*} \gamma'$ such that 1) $\mathbb{E}(\gamma') = C'$ and, 2) γ' is the same as γ except the label of each n_l is (q'_l, l).*

Proof sketch. For ease of presentation, we provide the proof in the case of $w = 2$. This proof can be generalized to any w in a straightforward manner.

The proof for $w = 2$ is essentially the same argument that is given in the intuition paragraph. Throughout the run that we are going to describe, if a node $\mathsf{n} \notin \{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2\}$ receives a message, then it will always take the self-loop transitions that we have constructed in the gadget for the rule r.

From γ, n_0 broadcasts $begin_0^r$ and moves to $(\texttt{req-fwd}[r], 0)$ and n_1 receives it and moves to $(\texttt{req-rec}[r], 1)$. Then, n_1 broadcasts $begin_1^r$ and moves to $(\texttt{req-fwd}[r], 1)$ and n_0 and n_2 receive it and move to $(\texttt{wait}[r], 0)$ and $(\texttt{req-rec}[r], 2)$ respectively. Then, n_2 broadcasts $begin_2^r$ and moves to $(\texttt{wait}[r], 2)$ and n_1 receives it and moves to $(\texttt{wait}[r], 1)$. Notice that at this point, the base of each n_i is $\texttt{wait}[r]$ and the labels of all the other nodes are unchanged, i.e., the same as the labels in γ.

Now, we proceed in the reverse direction. n_2 broadcasts end_2^r and moves to $(\texttt{ack-fwd}[r], 2)$ and n_1 receives it and moves to $(\texttt{ack-rec}[r], 1)$. Then, n_1 broadcasts end_1^r and moves to $(\texttt{ack-fwd}[r], 1)$ and n_0 and n_2 receive it and move to $(\texttt{ack-rec}[r], 0)$ and $(q_2', 2)$ respectively. Then, n_0 broadcasts end_0^r and moves to $(q_0', 0)$ and n_1 receives it and moves to $(q_1', 1)$. It is clear that the configuration reached at the end of this run satisfies the required properties. ◀

We now show a partial converse to the above lemma. It says that if there is a run of good configurations which uses only the transitions given in the gadget for the rule r and begins and ends with the root being in $(q_0, 0)$ and $(q_0', 0)$, then it is possible to "lift" that run back to the corresponding configurations in the NCS $\mathcal{N}$.

▶ **Lemma 9** ($\mathcal{N}$ simulates $\mathcal{P}_{\mathsf{sim}}$). *Suppose $\gamma \xrightarrow{*} \gamma'$ where 1) γ is a good configuration, 2) the labels of the root in γ and γ' are $(q_0, 0)$ and $(q_0', 0)$ and 3) in all the configurations between γ and γ', the base of the root is in the set $\{\texttt{req-fwd}[r], \texttt{wait}[r], \texttt{ack-rec}[r]\}$. Then, 1) γ' is a good configuration and, 2) $\mathbb{E}(\gamma) \overset{*}{\dashrightarrow} \mathbb{E}(\gamma')$.*

Proof sketch. Let the run $\gamma \xrightarrow{*} \gamma'$ be of the form $\gamma = \gamma_0 \rightarrow \gamma_1 \rightarrow \dots \gamma_{m-1} \rightarrow \gamma_m = \gamma'$. By means of induction and some extensive case analysis on the gadget that we have constructed, we can first prove that there exists a path $\mathsf{n}_0, \mathsf{n}_1, \dots, \mathsf{n}_w$ in γ with the following properties:

- For each $0 \leq l \leq w$, the label of n_l is (q_l, l) in γ and (q_l', l) in γ'.
- For each $0 \leq l \leq w$, n_l broadcasts exactly two messages: $begin_l^r$ and end_l^r.
- For each $0 \leq l < w$, the only child of n_l that broadcasts a message in the run is n_{l+1}.

We then let $Ch(\mathsf{n}_l)$ denote the set of children of n_l. Notice that the only node which could broadcast a message in γ_0 is n_0 and so it must be the case that $\gamma_0 \xrightarrow{\mathsf{n}_0, begin_0^r} \gamma_1$. Now, suppose, for some $0 \leq l < w$, we have shown that it must be the case that $\gamma_0 \xrightarrow{\mathsf{n}_0, begin_0^r} \gamma_1 \dots \gamma_l \xrightarrow{\mathsf{n}_l, begin_l^r} \gamma_{l+1}$. Then, notice that the only nodes whose labels in γ_{l+1} could have an outgoing broadcast transition are the nodes in $\bigcup_{0 \leq l' < l} (Ch(\mathsf{n}_{l'}) \setminus \{\mathsf{n}_{l'+1}\}) \cup Ch(\mathsf{n}_l)$. By our claim, among these only n_{l+1} broadcasts a message and so we must have that $\gamma_{l+1} \xrightarrow{\mathsf{n}_{l+1}, begin_{l+1}^r} \gamma_{l+2}$. Hence, in this way we get that $\gamma_0 \xrightarrow{\mathsf{n}_0, begin_0^r} \dots \gamma_w \xrightarrow{\mathsf{n}_w, begin_w^r} \gamma_{w+1}$. In exactly the same way, we can show that it must be the case that $\gamma_{w+1} \xrightarrow{\mathsf{n}_w, end_w^r} \gamma_{w+2} \xrightarrow{\mathsf{n}_{w-1}, end_{w-1}^r} \gamma_{w+3} \cdots \gamma_{2w+1} \xrightarrow{\mathsf{n}_0, end_0^r} \gamma_{2w+2} = \gamma_m$.

Let S be the set of all nodes whose base in γ belonged to $Q \cup \{start\}$ and whose base in γ' is *finish*. (Notice that $S \subseteq \bigcup_{0 \leq l < w} Ch(\mathsf{n}_l)$ and $S \cup \{\mathsf{n}_0, \dots, \mathsf{n}_w\}$ are exactly the set of nodes whose labels have changed during the run). It is then easy to see that, by firing the rule r from $\mathbb{E}(\gamma)$ and then deleting all the subtrees whose roots are in S, we get $\mathbb{E}(\gamma) \overset{*}{\dashrightarrow} \mathbb{E}(\gamma')$. ◀

With these two "simulation" lemmas, we have the following main result.

▶ **Theorem 10.** *The state q_{in} can cover the state q_f in the NCS $\mathcal{N}$ iff $(q_f, 0)$ can be covered by some execution in $\mathcal{P}$ starting at a good initial configuration.*

5 A seeker protocol $\mathcal{P}_{\text{seek}}$

In the previous section, we have shown that given a k-NCS $\mathcal{N} = (Q, \delta)$ along with two states $q_{in}, q_f \in Q$, we can construct a simulator protocol $\mathcal{P}_{\text{sim}}$, such that q_{in} can cover q_f in $\mathcal{N}$ iff $(q_f, 0)$ can be covered in $\mathcal{P}_{\text{sim}}$ by an execution starting at a *good initial configuration*. In this section, we will construct a *seeker* protocol $\mathcal{P}_{\text{seek}}$ and "attach" it to $\mathcal{P}_{\text{sim}}$ which will let us get rid of the *goodness* assumption. The seeker protocol $\mathcal{P}_{\text{seek}}$ will begin at an arbitrary initial communication topology and *seek* for a subgraph to act as a good initial configuration for $\mathcal{P}_{\text{sim}}$. Hence, once we have deployed $\mathcal{P}_{\text{seek}}$ to find such a subgraph, we can then use $\mathcal{P}_{\text{sim}}$ to simulate the k-NCS $\mathcal{N}$ on this subgraph.

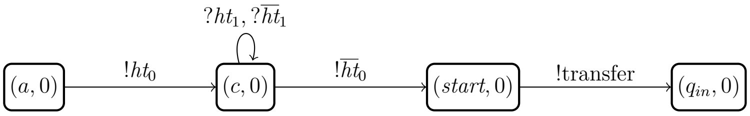
Formally, the seeker protocol $\mathcal{P}_{\text{seek}} = (Q_{\text{seek}}, I_{\text{seek}}, \Sigma_{\text{seek}}, \delta_{\text{seek}})$ will be a generalization of the protocol given in Figure 1 (with the exception that the (e, i) and $(\perp, i)$ states will be replaced by (start, i) and (finish, i) respectively).

States and alphabet. For each $0 \leq i \leq k$, $\mathcal{P}_{\text{seek}}$ will have six states of the form $(a, i), (b, i), (c, i), (d, i), (\mathit{start}, i)$ and (finish, i). Notice that (start, i) and (finish, i) are also present in $\mathcal{P}_{\text{sim}}$. $\mathcal{P}_{\text{seek}}$ will also have the state $(q_{in}, 0)$, which is a part of $\mathcal{P}_{\text{sim}}$ as well. Similar to $\mathcal{P}_{\text{sim}}$, we can define base and grade of a state.

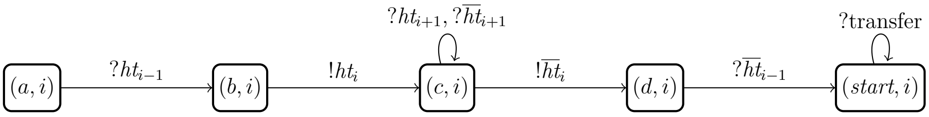
The initial set of states will be $\{(a, i) : 0 \leq i \leq k\}$. For each $0 \leq i \leq k$, Σ_{seek} will have two letters: ht_i and $\overline{ht}_i$. Σ_{seek} will also have another additional letter: *transfer*.

Transitions. Before we define the set of transitions, we make the same convention for $\mathcal{P}_{\text{seek}}$ that we had made in Remark 7 for $\mathcal{P}_{\text{sim}}$. Having stated this, we now describe the transitions:

- For the case of $i = 0$, we have the following transitions:



- For the case of $1 \leq i \leq k$, we have the following transitions: (The self-loops over the state (c, i) are not included when $i = k$).



Intuition behind the transitions. Let us give a brief intuition behind the transitions in the case of $k = 2$. A node n_0 which is at $(a, 0)$ aims to become the root of the good initial configuration that the seeker protocol should find, and so broadcasts ht_0, letting its neighbors know that it wants to be the root of the good subgraph. If any neighbor of n_0 is not in $(a, 1)$ then it immediately moves to a state with base *finish*. Otherwise, the set of all neighbors in $(a, 1)$ move to $(b, 1)$. From here, all of these nodes can broadcast ht_1, letting their neighbors know that they now want to become a child of the root. All these messages will also be received n_0 which will use the self-loop at $(c, 0)$ to ignore these messages. All the nodes which receive a ht_1 message can either move to a state with base *finish* or move to $(b, 2)$, from where they can broadcast ht_2 and thereby move to $(c, 2)$. At this point, we must have a tree subgraph in which n_0 is labelled by $(c, 0)$, its children are labelled by $(c, 1)$ and its children are labelled by $(c, 2)$.

Now the nodes labelled by $(c, 2)$ can all broadcast $\overline{ht}_2$, then the nodes labelled by $(c, 1)$ can all broadcast $\overline{ht}_1$ and then the node n_0 can broadcast $\overline{ht}_0$. This leads to a tree subgraph where n_0 is labelled by $(\mathit{start}, 0)$, its children are labelled by $(\mathit{start}, 1)$ and its children are labelled by $(\mathit{start}, 2)$. Now, n_0 can broadcast the letter "transfer" and move into $(q_{in}, 0)$, thereby *transferring* the control over to the simulator protocol $\mathcal{P}_{\text{sim}}$. In this manner, $\mathcal{P}_{\text{seek}}$ has found a good initial subgraph in which to run $\mathcal{P}_{\text{sim}}$.

Proof of correctness. Let $\mathcal{P} = (Q_{\text{seek}} \cup Q_{\text{sim}}, I_{\text{seek}}, \Sigma_{\text{seek}} \cup \Sigma_{\text{sim}}, \delta_{\text{seek}} \cup \delta_{\text{sim}})$ be the protocol obtained by taking the union of the seeker and the simulator protocols, such that the initial set of states is the initial set of states of the seeker protocol. Similar to the intuition given above, the protocol $\mathcal{P}$ first runs the seeker protocol till a node with label $(q_{in}, 0)$ is reached, at which point it runs the simulator protocol. The following lemma tells us that if a node gets labelled by $(q_{in}, 0)$ while running $\mathcal{P}$, then with that node as the root, there is a good initial configuration for the simulator protocol $\mathcal{P}_{\text{sim}}$. This then allows us the protocol $\mathcal{P}$ to run the simulator protocol on this good initial configuration.

▶ **Lemma 11.** *Suppose $\gamma \xrightarrow{*} \gamma' \xrightarrow{\mathsf{n}, \mathit{transfer}} \eta$ is an execution of $\mathcal{P}$. After removing all nodes whose label's base is finish in η, the connected component containing the node n is a good initial configuration for the simulator protocol $\mathcal{P}_{sim}$.*

Proof sketch. First, let us focus on the execution $\gamma \xrightarrow{*} \gamma'$. By definition of an execution, γ is an initial configuration for the protocol $\mathcal{P}_{\text{seek}}$ and so all the nodes in γ have their labels in the set $\{a_i : 0 \leq i \leq k\}$.

Let T be the connected component containing the node n in γ' after removing all nodes whose base is *finish*. Let $F := \{(\mathit{start}, i) : 0 \leq i \leq k\}$. First, we show that all nodes in T must have labels from the set F. Suppose there is a node n' in T whose label is not in F. Pick such an n' which is at the shortest distance from n and let $\mathsf{n} = \mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2, \ldots, \mathsf{n}_l, \mathsf{n}'$ be a shortest path from n to n'.

By a generalization of the argument given in Example 2, we can prove by induction that for each $1 \leq i \leq l$, the label of each n_i in T is (start, i) and the only neighbor of n_i which was labelled by $(c, i-1)$ at some point during the run is n_{i-1}. Using this, we can then show that n' must have moved to $(\mathit{start}, l+1)$ at some point during the run.

By assumption, the label of n' is not $(\mathit{start}, l+1)$ in T, and so it must moved out of $(\mathit{start}, l+1)$ to some state of the simulator protocol. By analysing the constructed protocol $\mathcal{P}$, we can then prove that n' must have received two ht_l messages. But any node that receives two ht_l messages must necessarily move to a state with base *finish*, contradicting the fact that $\mathsf{n}' \in T$.

Having proved that every node in T has its label in F, we can then show by examining the structure of the transitions, that T must be a tree of height atmost k such that n_0 is labelled by $(\mathit{start}, 0)$ and all nodes at distance i from n_0 are labelled by (start, i). This then implies that after removing all nodes with base *finish* in η, the connected component containing the node n is a good initial configuration for the simulator protocol. ◀

▶ **Theorem 12.** *The state q_{in} can cover q_f in the NCS $\mathcal{N}$ iff the state $(q_f, 0)$ can be covered from any initial configuration in $\mathcal{T}_{2k}(\mathcal{P})$.*

Proof sketch. Due to lack of space, we focus only on the right to left implication. Suppose $\gamma \xrightarrow{*} \gamma'$ is an execution of $\mathcal{P}$ such that some node n in γ' is labelled by $(q_f, 0)$. Let γ_0 be the configuration along this run when the node n first got the label $(q_{in}, 0)$. (Notice that such a configuration must exist because of the construction of $\mathcal{P}$). By Lemma 11, in γ_0, if we remove all nodes whose base is *finish*, then we get a good initial configuration T for $\mathcal{P}_{\text{sim}}$ with n as the root. Notice that no node with base *finish* can ever broadcast a message. Hence, in the run $\gamma_0 \xrightarrow{*} \gamma'$, none of the nodes in T ever receive a message from any node outside of T. It follows that we can restrict the run $\gamma_0 \xrightarrow{*} \gamma'$ to only the subtree T, to get a run of $\mathcal{P}_{\text{sim}}$ starting at a good initial configuration and covering $(q_f, 0)$. By Theorem 10, we get that q_f can be covered from q_{in} in $\mathcal{N}$. ◀

Hence, we get,

▶ **Corollary 13.** *Bounded-Path-Cover is* $\mathbf{F}_{\epsilon_0}$*-hard.*

6 Upper bound for Bounded-Path-Cover

In this section, we give a sketch of the proof that Bounded-Path-Cover is in $\mathbf{F}_{\epsilon_0}$. Let $\mathcal{P} = (Q, I, \Sigma, \Delta)$ be a fixed protocol.

▶ **Definition 14.** *Let $\gamma_1 = (\mathsf{N}_1, \mathsf{E}_1, \mathsf{L}_1)$ and $\gamma_2 = (\mathsf{N}_2, \mathsf{E}_2, \mathsf{L}_2)$ be two configurations of $\mathcal{P}$. We say that γ_1 is an induced subgraph of γ_2 (denoted by $\gamma_1 \preceq_{is} \gamma_2$) if there is a label preserving injection h from N_1 to N_2 such that $(\mathsf{n}, \mathsf{n}') \in \mathsf{E}_1$ if and only if $(h(\mathsf{n}), h(\mathsf{n}')) \in \mathsf{E}_2$.*

It is known that, for any $k \geq 1$, the set of all k-path bounded configurations of $\mathcal{P}$ is a well-quasi ordering under the induced subgraph relation $\preceq_{\text{is}}$ (Theorem 2.2 of [10]). Using this fact, the authors of [9] show that for every k, the transition system $\mathcal{T}_k(\mathcal{P})$ is a *well-structured transition system* (WSTS) and then apply the generic backward exploration algorithm for WSTS (See [20, 13]) to prove that Bounded-Path-Cover is decidable. By using the standard and generic complexity arguments for WSTS (See [20, 13, 21]), an upper bound on the running time of their procedure simply boils down to estimating the length of *controlled bad sequences* of k-path bounded configurations under the induced subgraph relation.

Let $H : \mathbb{N} \to \mathbb{N}$ be the successor function and let $n \in \mathbb{N}$. For each $i \in \mathbb{N}$, let H^i denote the i-fold application of H to itself i times, with H^0 being the identity function.

▶ **Definition 15.** *A sequence $\gamma_0, \gamma_1, \ldots,$ of k-path bounded configurations is (H, n)-controlled bad if the number of nodes in each γ_i is at most $H^i(n)$ and $\gamma_i \not\preceq_{is} \gamma_j$ for any $i < j$.*

Our main result is an upper bound on the length of (H, n)-controlled bad sequences of k-path bounded configurations, by embedding these configurations into the well-quasi ordering of *generalized priority alphabets* (See [16]). This encoding is inspired by a similar encoding given for bounded depth trees in Section 8.1 of [16]. This result then allows us to prove that

▶ **Theorem 16.** *Bounded-Path-Cover is in $\mathbf{F}_{\epsilon_0}$ and hence $\mathbf{F}_{\epsilon_0}$-complete.*

References

1 Sergio Abriola, Santiago Figueira, and Gabriel Senno. Linearizing well quasi-orders and bounding the length of bad sequences. *Theor. Comput. Sci.*, 603:3–22, 2015. doi:10.1016/j.tcs.2015.07.012.

2 Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar. Reconfiguration and message losses in parameterized broadcast networks. *Log. Methods Comput. Sci.*, 17(1), 2021. URL: https://lmcs.episciences.org/7280.

3 Michael Blondin. The abcs of petri net reachability relaxations. *ACM SIGLOG News*, 7(3):29–43, 2020. doi:10.1145/3436980.3436984.

4 Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. The logical view on continuous petri nets. *ACM Trans. Comput. Log.*, 18(3):24:1–24:28, 2017. doi:10.1145/3105908.

5 Michael Blondin and Christoph Haase. Logics for continuous reachability in petri nets and vector addition systems with states. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005068.

6 Pierre Chambart and Philippe Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 205–216, 2008. doi:10.1109/LICS.2008.47.

7 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. 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 24–33. ACM, 2019. doi:10.1145/3313276.3316369.

8 Normann Decker and Daniel Thoma. On freeze LTL with ordered attributes. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9634 of *Lecture Notes in Computer Science*, pages 269–284. Springer, 2016. doi:10.1007/978-3-662-49630-5_16.

9 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR 2010 - Concurrency Theory, 21th International Conference,*, pages 313–327, 2010. doi:10.1007/978-3-642-15375-4_22.

10 Guoli Ding. Subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 16(5):489–502, 1992. doi:10.1002/jgt.3190160509.

11 Jacob Elgaard, Nils Klarlund, and Anders Møller. Mona 1.x: New techniques for ws1s and ws2s. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification*, pages 516–520, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

12 Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp J. Meyer, and Filip Niksic. An smt-based approach to coverability analysis. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 603–619. Springer, 2014. doi:10.1007/978-3-319-08867-9_40.

13 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with dickson's lemma. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science*, pages 269–278, 2011. doi:10.1109/LICS.2011.39.

14 Estíbaliz Fraca and Serge Haddad. Complexity analysis of continuous petri nets. *Fundam. Informaticae*, 137(1):1–28, 2015. doi:10.3233/FI-2015-1168.

15 Christoph Haase and Simon Halfon. Integer vector addition systems with states. In Joël Ouaknine, Igor Potapov, and James Worrell, editors, *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, volume 8762 of *Lecture Notes in Computer Science*, pages 112–124. Springer, 2014. doi:10.1007/978-3-319-11439-2_9.

16 Christoph Haase, Sylvain Schmitz, and Philippe Schnoebelen. The power of priority channel systems. *Log. Methods Comput. Sci.*, 10(4), 2014. doi:10.2168/LMCS-10(4:4)2014.

17 Albert R. Meyer. Weak monadic second order theory of succesor is not elementary-recursive. In Rohit Parikh, editor, *Logic Colloquium*, pages 132–154, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

18 Sylvain Schmitz. Complexity bounds for ordinal-based termination - (invited talk). In *Reachability Problems - 8th International Workshop, RP 2014*, pages 1–19, 2014. doi:10.1007/978-3-319-11439-2_1.

19 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.

20 Sylvain Schmitz and Philippe Schnoebelen. Multiply-recursive upper bounds with higman's lemma. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011*, pages 441–452, 2011. doi:10.1007/978-3-642-22012-8_35.

21 Sylvain Schmitz and Philippe Schnoebelen. The power of well-structured systems. In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, volume 8052 of *Lecture Notes in Computer Science*, pages 5–24. Springer, 2013. doi:10.1007/978-3-642-40184-8_2.

A Appendix

A.1 Proofs for Section 6

First, let us describe the backward exploration algorithm for solving Bounded-Path-Cover that is given in Section 5 of [9]. Given a protocol $\mathcal{P} = (Q, I, \Sigma, \delta)$, a state f and a number k, we consider the set of all configurations in $\mathcal{T}_k(\mathcal{P})$ with the induced subgraph ordering $\preceq_{\text{is}}$. Given a set S of $\mathcal{T}_k(\mathcal{P})$ we let $\uparrow S := \{\gamma' : \exists \gamma \in S, \gamma \preceq_{\text{is}} \gamma'\}$. A set S is called *upward-closed* if $S = \uparrow S$.

In Section 5 of [9], the following results are proved about $\mathcal{T}_k(\mathcal{P})$:

- If S is upward-closed, then there exists a finite set B such that $\uparrow B = S$. Such a B will be called the basis of S.
- If S is upward-closed and if $Pre(S)$ is the set of all configurations $\gamma' \in \mathcal{T}_k(\mathcal{P})$ such that there is a configuration $\gamma \in S$ with $\gamma' \to \gamma$, then $S \cup Pre(S)$ is upward-closed. Moreover, given a basis B of S, we can compute a basis B' of $S \cup Pre(S)$ such that the number of nodes of each configuration in B' is at most one more than the maximum number of nodes in any configuration of B.

In Theorem 5 of [9] it is shown that the following algorithm terminates and is correct for Bounded-Path-Cover : Construct a sequence of finite sets $B_0, B_1, \dots,$ such that each $B_i \subseteq \mathcal{T}_k(\mathcal{P})$, B_0 is the single node configuration labelled by f and B_{i+1} is a basis for $\uparrow B_i \cup Pre(\uparrow B_i)$. The algorithm then finds the first m such that $\uparrow B_m = \uparrow B_{m+1}$ and checks if there is an initial configuration in $\uparrow B_m$.

The running time complexity of the algorithm is mainly dominated by the length of the sequence $B_0, B_1, \dots, B_m$. Since m is the first index such that $\uparrow B_m = \uparrow B_{m+1}$, we can find a minimal element $\gamma_i \in \uparrow B_{i+1} \setminus \uparrow B_i$ for each $i < m$.

Consider the sequence $\gamma_0, \dots, \gamma_{m-1}$. Notice that $\gamma_i \not\preceq_{\text{is}} \gamma_j$ for any $j > i$ and further the number of nodes in each γ_i is at most $H^i(1)$, where H is the successor function. It follows that $\gamma_0, \dots, \gamma_{m-1}$ is a controlled bad sequence. Our main result is that

▶ **Lemma 17.** *The length of (H, n)-controlled bad sequences over k-path bounded configurations of $\mathcal{P}$ is upper bounded by the function $F_{\epsilon_0}(p(|Q|, k, n))$.*

Here F_{ϵ_0} is the *fast-growing* function at level ϵ_0 and p is some fixed primitive recursive function. For our purposes, we do not need the actual definition of F_{ϵ_0}, but we only need to know that $\mathbf{F}_{\epsilon_0}$ contains the set of problems whose running time is upper bounded by the function F_{ϵ_0} composed with any primitive recursive function (See [19]). By the lemma above and the fact that the running time complexity of the algorithm for Bounded-Path-Cover is primarily dominated by the length of $(H, 1)$-controlled bad sequences we get,

▶ **Theorem 18.** *Bounded-Path-Cover is in* $\mathbf{F}_{\epsilon_0}$*.*

All that suffices is to prove Lemma 17. To do so, we will reduce the problem of estimating the length of controlled bad sequences over k-path bounded configurations to the problem of estimating the length of controlled bad sequences over another well-quasi order for which we already know upper bounds. We now proceed to recall this well-quasi order as it is defined in [16].

Generalized priority alphabets

Given a number $k \in \mathbb{N}$ called the priority level and a finite set Γ, a *generalised priority alphabet* is the set $\Sigma_{\Gamma,k} := \{(a,i) : a \in \Gamma, 0 \le i \le k\}$. Given $m = (a,i) \in \Sigma_{\Gamma,k}$, we say that i is the priority of m. Then for $x, y \in \Sigma^*_{\Gamma,k}$, we say that $x \sqsubseteq_{\Gamma,k} y$ if $x = (a_1, i_1), (a_2, i_2), \dots, (a_l, i_l)$ where each $(a_j, i_j) \in \Sigma_{\Gamma,k}$ and $y = y_1(a_1, i_1)y_2(a_2, i_2)y_3 \dots y_l(a_l, i_l)$ such that $\forall 1 \le j \le l$, we have $y_j \in \Sigma^*_{\Gamma,i_j}$, i.e., x can be obtained from y by removing subwords in such a manner so that the priority of each removed subword is not bigger than the first preserved letter to its right. It is known that for every k and Γ, the ordering $\sqsubseteq_{\Gamma,k}$ is a well-quasi ordering. (Theorem 3.6 of [16]). Now, similar to controlled bad sequences for k-path bounded configurations, we can define (a slightly different notion of) controlled bad sequences for words over $\Sigma_{\Gamma,k}$. Let $Sq : \mathbb{N} \to \mathbb{N}$ be the squaring function and let Sq^i denote the squaring function composed with itself i times.

▶ **Definition 19.** *A sequence $w_0, w_1, \dots,$ of words over $\Sigma_{\Gamma,k}$ is (Sq, n)-controlled bad if the length of each w_i is at most $Sq^i(n)$ and $w_i \not\sqsubseteq_{\Gamma,k} w_j$ for any $i < j$.*

Encoding k-path bounded graphs using generalized priority alphabets

A labelled k-path bounded graph is any graph $G = (\mathsf{N}, \mathsf{E}, \mathsf{L})$ such that there is a labelling function $\mathsf{L} : \mathsf{N} \to A$ for some some finite set A. (Notice that the set of k-path bounded configurations of a protocol is a labelled k-path bounded graph where A is the set of states of the protocol). We have the following theorem regarding labelled k-path bounded graphs.

▶ **Theorem 20** (Lemma 2.1 of [10]). *Suppose G is a labelled k-path bounded graph for $k \ge 1$. Then there is a node n such that every connected component of $G \setminus \{\mathsf{n}\}$ is a labelled $(k-1)$-path bounded graph.*

This theorem suggests the following inductive encoding of labelled k-path bounded graphs as strings over a priority alphabet: Let $G = (\mathsf{N}, \mathsf{E}, \mathsf{L})$ be any labelled graph with labelling function $\mathsf{L} : \mathsf{N} \to A$ where A is some finite set. Let $\mathsf{e}, \bar{\mathsf{e}}$ be two symbols not in the finite set A and let $A^k := \cup_{0 \le i \le k} A \times \{\mathsf{e}, \bar{\mathsf{e}}\}^i$. Notice that $A^0 := A$. By induction on k, we will now define a string $\langle G \rangle \in \Sigma_{A^k,k}$.

Base case: If G is a 0-path bounded configuration, then G is a single node n and can be encoded as $(\mathsf{L}(\mathsf{n}), 0) \in \Sigma^*_{A^0,0}$.

Induction step: Suppose G is a k-path bounded configuration for some $k \ge 1$ such that G is not $(k-1)$-path bounded. Let n be a vertex such that all the connected components $C_1, \dots, C_l$ of $G \setminus \{\mathsf{n}\}$ are $(k-1)$-path bounded configurations. (Such a node exists by Theorem 20). For every node n' in every C_i, first change its label from $\mathsf{L}(\mathsf{n}')$ to $(\mathsf{L}(\mathsf{n}'), \mathsf{e})$ if n' is a neighbor of n in G and otherwise change its label to $(\mathsf{L}(\mathsf{n}'), \bar{\mathsf{e}})$. Call these new labelled graphs as $C_1^{\mathsf{n}}, \dots, C_l^{\mathsf{n}}$.

By induction hypothesis, for each C_i^{n}, we have a string $\langle C_i^{\mathsf{n}} \rangle \in \Sigma^*_{((A \times \{\mathsf{e}, \bar{\mathsf{e}}\})^{k-1}, k-1)} \subseteq \Sigma^*_{A^k,k-1}$. We now let $\langle G \rangle := \langle C_1^{\mathsf{n}} \rangle (\mathsf{L}(\mathsf{n}), k) \langle C_2^{\mathsf{n}} \rangle (\mathsf{L}(\mathsf{n}), k) \dots \langle C_l^{\mathsf{n}} \rangle (\mathsf{L}(\mathsf{n}), k)$.

Notice that if G is a labelled k-path bounded graph which is not $(k-1)$-path bounded, then $\langle G \rangle$ is of the form $\langle C_1^{\mathsf{n}} \rangle (a,k) \langle C_2^{\mathsf{n}} \rangle (a,k) \dots \langle C_l^{\mathsf{n}} \rangle (a,k)$ where 1) a is the label of some node n in G, 2) $C_1, \dots, C_l$ are connected components of $G \setminus \{\mathsf{n}\}$ which are labelled $(k-1)$-path bounded subgraphs of G. This will be called the *decomposition* of $\langle G \rangle$ and the node n will be called its *crown*.

We then have the following lemma:

▶ **Lemma 21.** *If G and H are such that $\langle G \rangle \sqsubseteq_{A^k,k} \langle H \rangle$ then $G \preceq_{is} H$.*

Proof. Notice that if $\langle G \rangle \sqsubseteq_{A^k,k} \langle H \rangle$, then the highest priority appearing in $\langle G \rangle$ and $\langle H \rangle$ must be the same, which, without loss of generality, we can assume to be k.

We prove the lemma by induction on k. The base case of 0 is clear.

For the induction step, let $\langle C_1^{\mathsf{n}} \rangle (a,k) \langle C_2^{\mathsf{n}} \rangle (a,k) \dots \langle C_m^{\mathsf{n}} \rangle (a,k)$ be the decomposition of $\langle G \rangle$ with crown n and let $\langle D_1^{\mathsf{n}'} \rangle (a',k) \langle D_2^{\mathsf{n}'} \rangle (a',k) \dots \langle D_n^{\mathsf{n}'} \rangle (a',k)$ be the decomposition of $\langle H \rangle$ with crown n'. Since $\langle G \rangle \sqsubseteq_{A^k,k} \langle H \rangle$, it must be the case that $a = a'$.

By definition of the $\sqsubseteq_{A^k,k}$ relation, it must be the case that for every C_j^{n}, there exists i_j such that $\langle C_j^{\mathsf{n}} \rangle \sqsubseteq_{A^k,k-1} \langle D_{i_j}^{\mathsf{n}'} \rangle$. Notice that the priority has reduced and we can apply the induction hypothesis to conclude that for each j, $C_j^{\mathsf{n}} \preceq_{\text{is}} D_{i_j}^{\mathsf{n}'}$ and so there exists a label preserving injection h_j from the nodes of C_j^{n} to the nodes of $D_{i_j}^{\mathsf{n}'}$ such that (u,v) is an edge in C_j^{n} iff $(h_j(u), h_j(v))$ is an edge in $D_{i_j}^{\mathsf{n}'}$.

Now, consider the following label preserving injection h from G to H: Map the crown n to the other crown n' and if n'' is any other node in any one of the connected components C_j, then map n'' to $h_j(\mathsf{n}'')$. Notice that if u and v are nodes in G which belong to the same connected component of $G \setminus \{\mathsf{n}\}$ then (u,v) is an edge in G iff $(h(u), h(v))$ is an edge in H. Similarly, if u and v are nodes in G which belong to different connected components of $G \setminus \{\mathsf{n}\}$ then $h(u)$ and $h(v)$ also belong to different connected components of $H \setminus \{\mathsf{n}'\}$ and so the statement "(u,v) is an edge in G iff $(h(u), h(v))$ is an edge in H" is vacously true.

Finally suppose $u = \mathsf{n}$ and v is some other node of G. Notice that the last field in the label of v is e if (u,v) is an edge in G and $\bar{\mathsf{e}}$ otherwise. By definition of h we have that $h(u) = \mathsf{n}'$ and also that the label of $h(v)$ is the same as v. But by definition of decomposition of $\langle H \rangle$, the last field in the label of $h(v)$ is e if $(\mathsf{n}', h(v))$ is an edge in H and $\bar{\mathsf{e}}$ otherwise. Hence, in this case as well, we have shown that (u,v) is an edge in G iff $(h(u), h(v))$ is an edge in H. This concludes the proof. ◀

Upper bound on the length of controlled bad sequences for k-path bounded configurations

Fix a protocol $\mathcal{P}$ with states Q and a number k and consider the set of configurations in $\mathcal{T}_k(\mathcal{P})$. By the previous lemma, we can infer that the length of the longest (H,n)-controlled bad sequence over the set of configurations of $\mathcal{T}_k(\mathcal{P})$ is at most the length of the longest (Sq,n)-controlled bad sequence over the generalized priority alphabet $\Sigma_{Q^k,k}$, which we know is at most $F_{\epsilon_0}(p(|Q|,k,n))$ where p is some primitive recursive function (Proposition 4.1 and Sections 4.1.1 and 4.1.2 of [16]). This then implies Lemma 17, which is what we wanted to prove.

On Classical Decidable Logics Extended with Percentage Quantifiers and Arithmetics

Bartosz Bednarczyk
Computational Logic Group, Technische Universität Dresden, Germany
Institute of Computer Science, University of Wrocław, Poland

Maja Orłowska
Institute of Computer Science, University of Wrocław, Poland

Anna Pacanowska
Institute of Computer Science, University of Wrocław, Poland

Tony Tan
Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan

Abstract

During the last decades, a lot of effort was put into identifying decidable fragments of first-order logic. Such efforts gave birth, among the others, to the two-variable fragment and the guarded fragment, depending on the type of restriction imposed on formulae from the language. Despite the success of the mentioned logics in areas like formal verification and knowledge representation, such first-order fragments are too weak to express even the simplest statistical constraints, required for modelling of influence networks or in statistical reasoning.

In this work we investigate the extensions of these classical decidable logics with percentage quantifiers, specifying how frequently a formula is satisfied in the indented model. We show, surprisingly, that all the mentioned decidable fragments become undecidable under such extension, sharpening the existing results in the literature. Our negative results are supplemented by decidability of the two-variable guarded fragment with even more expressive counting, namely Presburger constraints. Our results can be applied to infer decidability of various modal and description logics, e.g. Presburger Modal Logics with Converse or $\mathcal{ALCI}$, with expressive cardinality constraints.

2012 ACM Subject Classification Theory of computation → Logic and databases

Keywords and phrases statistical reasoning, knowledge representation, satisfiability, fragments of first-order logic, guarded fragment, two-variable fragment, (un)decidability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.36

Related Version *Full Version*: https://arxiv.org/abs/2106.15250

Funding *Bartosz Bednarczyk*: supported by "Diamentowy. Grant" no. DI2017 006447.
Tony Tan: supported by Taiwan Ministry of Science and Technology under grant no. 109-2221-E-002-143-MY3 and National Taiwan University under grant no. 109L891808.

1 Introduction

Since the works of Church, Turing and Trakhtenbrot, it is well-known that the (finite) satisfiability and validity problems for the First-Order Logic (FO) are undecidable [29]. Such results motivated researchers to study restricted classes of FO that come with decidable satisfiability problem, such as the prefix classes [9], fragments with fixed number of variables [28], restricted forms of quantification [1, 30] and the restricted use of negation [6]. These fragments have found many applications in the areas of knowledge representation, automated reasoning and program verification, just to name a few. To the best of our knowledge, none of the known decidable logics incorporate a feature that allows for stating

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 36; pp. 36:1–36:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

even a very modest statistical property. For example, one may want to state that "to qualify to be a major, one must have at least 51% of the total votes", which may be useful to formalise, e.g. the voting systems.

Our results. In this paper, we revisit the satisfiability problem for some of the most prominent fragments of FO, namely the two-variable fragment FO^2 and the guarded fragment GF. We extend them with the so-called *percentage* quantifiers, in two versions: *local* and *global*. Global percentage quantifiers are quantifiers of the form $\exists^{=q\%}x\ \varphi(x)$, which states that the formula $\varphi(x)$ holds on exactly $q\%$ of the domain elements. Their local counterparts are quantifiers of the form $\exists_R^{=q\%}y\ \varphi(x,y)$, which intuitively means that exactly $q\%$ of the R-successors of an element x satisfy φ.

In this paper, we show that both FO^2 and GF become undecidable when extended with percentage quantifiers of any type. In fact, the undecidability of GF already holds for its three variable fragment GF^3. Our results strengthen the existing undecidability proofs of $\mathcal{ALCISCC}^{++}$ from [4] and of FO^2 with equicardinality statements (implemented via the Härtig quantifier) from [18] and contrast with the decidability of FO^2 with counting quantifiers (C^2) [17, 23, 26] and modulo and ultimately-periodic counting quantifiers [8].

Additionally, we show that the decidability status of GF can be regained if we consider GF^2, i.e. the intersection of GF and FO^2, which is still a relevant fragment of FO that captures standard description logics up to $\mathcal{ALCIHb}^{\mathsf{self}}$ [5, 14]. We in fact show a stronger result here: GF^2 remains decidable when extended with *local Presburger quantifiers*, which are essentially Presburger constraints on the neighbouring elements, e.g. we can say that the number of red outgoing edges plus twice the number of blue outgoing edges is at least three times as many as the number of green incoming edges.

We stress here that the semantics of global percentage quantifiers makes sense only over finite domains and hence, we study the satisfiability problem over finite models only. Similarly, the semantics of local percentage quantifiers only makes sense if the models are finitely-branching. While we stick again to the finite structures, our results on local percentage quantifiers also can be transferred to the case of (possibly infinite) finitely-branching structures.

Related works. Some restricted fragments of GF^2 extended with arithmetics, namely the (multi) modal logics, were already studied in the literature [11, 20, 2, 4], where the decidability results for their finite and unrestricted satisfiability were obtained. However, the logics considered there do not allow the use of the inverse of relations. Since GF^2 captures the extensions of all the aforementioned logics with the inverse relations, our decidability results subsume those in [11, 20, 2, 4]. We note that prior to our paper, it was an open question whether any of these decidability results still hold when inverse relations are allowed [4]. In our approach, despite the obvious difference in expressive power, we show that GF^2 with Presburger quantifiers can be encoded directly into the two-variable logic with counting quantifiers [17, 23, 26], which we believe is relatively simple and avoids cumbersome reductions of the satisfiability problem into integer programming.

2 Preliminaries

We employ the standard terminology from finite model theory [21]. We refer to structures/models with calligraphic letters $\mathcal{A}, \mathcal{B}, \mathcal{M}$ and to their universes with the corresponding capital letters A, B, M. We work only on structures with *finite* universes over purely relational (i.e.

constant- and function-free) signatures of arity ≤ 2 containing the equality predicate $=$. We usually use $a, b, \ldots$ to denote elements of structures, $\bar{a}, \bar{b}, \ldots$ for tuples of elements, $x, y, \ldots$ for variables and $\bar{x}, \bar{y}, \ldots$ for tuples of variables (all of these possibly with some decorations). We write $\varphi(\bar{x})$ to indicate that all free variables of φ are in $\bar{x}$. We write $\mathcal{M}, x/a \models \varphi(x)$ to denote that $\varphi(x)$ holds in the structure $\mathcal{M}$ when the free variable x is assigned with element a. Its generalization to arbitrary number of free variables is defined similarly. The (finite) satisfiability problem is to decide whether an input formula has a (finite) model.

2.1 Percentage quantifiers

For a formula $\varphi(x)$ with a single free-variable x, we write $|\varphi(x)|_{\mathcal{M}}$ to denote the total number of elements of $\mathcal{M}$ satisfying $\varphi(x)$. Likewise, for an element $a \in M$ and a formula $\varphi(x, y)$ with free variables x and y, we write $|\varphi(x, y)|_{\mathcal{M}}^{x/a}$ to denote the total number of elements $b \in M$ such that (a, b) satisfies $\varphi(x, y)$.

The *percentage quantifiers* are quantifiers of the form $\exists^{=q\%}x\ \varphi(x, y)$, where q is a rational number between 0 and 100, stating that exactly $q\%$ of domain elements satisfy $\varphi(x, y)$ with y known upfront. Formally:

$$\mathcal{M}, y/a \models \exists^{=q\%}x\ \varphi(x, y) \qquad \text{iff} \qquad |\varphi(x, y)|_{\mathcal{M}}^{y/a} = \frac{q}{100} \cdot |M|.$$

Percentage quantifiers for other thresholds (e.g. for $<$) are defined analogously. We stress here that the above quantifiers count *globally*, i.e. they take the whole universe of $\mathcal{M}$ into account. This motivates us to define their local counterpart, as follows: for a binary[1] relation R and a rational q between 0 and 100, we define the quantifier $\exists_R^{=q\%}y\ \varphi(x, y)$, which evaluates to true whenever exactly $q\%$ of R-successors y of x satisfy $\varphi(x, y)$. Formally,

$$\mathcal{M}, x/a\ \models\ \exists_R^{=q\%}y\ \varphi(x, y) \qquad \text{iff} \qquad |R(x, y) \wedge \varphi(x, y)|_{\mathcal{M}}^{x/a} = \frac{q}{100} \cdot |R(x, y)|_{\mathcal{M}}^{x/a}.$$

We define the percentage quantifiers w.r.t. R^{-} (i.e. the inverse of R) and for other thresholds analogously.

2.2 Local Presburger quantifiers

The *local Presburger quantifiers* are expressions of the following form:

$$\sum_{i=1}^{n} \lambda_i \cdot \#_y^{r_i}[\varphi_i(x, y)] \quad \circledast \quad \delta$$

where λ_i, δ are integers; r_i is either R or R^{-} for some binary relation R; $\varphi_i(x, y)$ is a formula with free variables x and y; and $\circledast$ is one of $=, \neq, \leq, \geq, <, >, \equiv_d$ or $\not\equiv_d$, where $d \in \mathbb{N}_+$. Here $\equiv_d$ denotes the congruence modulo d. Note that the above formula has one free variable x.

Intuitively, the expression $\#_y^{r_i}[\varphi_i(x, y)]$ denotes the number of y's that satisfy $r_i(x, y) \wedge \varphi_i(x, y)$ and evaluates to true on x, if the (in)equality $\circledast$ holds. Formally,

$$\mathcal{M}, x/a \quad \models \quad \sum_{i=1}^{n} \lambda_i \cdot \#_y^{r_i}[\varphi_i(x, y)] \ \circledast\ \delta \quad \text{iff} \quad \sum_{i=1}^{n} \lambda_i \cdot |r_i(x, y) \wedge \varphi_i(x, y)|_{\mathcal{M}}^{x/a} \quad \circledast \quad \delta$$

Note that local percentage quantifiers can be expressed with Presburger quantifiers, e..g. $\exists_R^{50\%} y\varphi(x, y)$ can be expressed as local Presburger quantifier: $\#_y^R[\varphi(x, y)] - \frac{1}{2}\#_y^R[\top] = 0$.

[1] Local percentage quantifiers for predicates of arity higher than two can also be defined but we will never use them. Hence, for simplicity, we define such quantifiers only for binary relations.

2.3 Logics

In this paper we mostly consider two fragments of first-order logic, namely *the two-variable fragment* FO^2 and *the guarded fragment* GF. The former logic is a fragment of FO in which we can only use the variables x and y. By allowing local and global percentage quantifiers in addition to the standard universal and existential quantifiers, we obtain the logics $\mathrm{FO}^2_{loc\%}$ and $\mathrm{FO}^2_{gl\%}$. The latter logic is defined by relativising quantifiers with relations. More formally, GF is the smallest set of first-order formulae such that the following holds.

- GF contains all atomic formulae $R(\bar{x})$ and equalities between variables.
- GF is closed under boolean connectives.
- If $\psi(\bar{x}, \bar{y})$ is in GF and $\gamma(\bar{x}, \bar{y})$ is a relational atom containing all free variables of ψ, then both $\forall \bar{y}\ \gamma(\bar{x}, \bar{y}) \to \psi(\bar{x}, \bar{y})$ and $\exists \bar{y}\ \gamma(\bar{x}, \bar{y}) \wedge \psi(\bar{x}, \bar{y})$ are in GF.

By allowing global percentage quantifiers additionally in place of existential ones, we obtain the logic $\mathrm{GF}_{gl\%}$. We obtain the logic $\mathrm{GF}_{loc\%}$ by extending GF's definition with the rule:[2]

- $\exists^{=q\%}_{R} y\ \varphi(x, y)$ is in $\mathrm{GF}_{loc\%}$ iff $\varphi(x, y)$ in GF with free variables x, y.

Similarly, we obtain GF_{pres} by extending GF's definition with the rule:

- $\sum_{i=1}^{n}\ \lambda_i \cdot \#_y^{r_i}[\varphi_i(x, y)]\ \circledast\ \delta$ is in GF_{pres} iff $\varphi_i(x, y)$ are in GF with free variables x, y.

Finally, we use $\mathrm{GF}^k_{gl\%}$, $\mathrm{GF}^k_{loc\%}$ and GF^k_{pres} to denote the k-variable fragments of the mentioned logics. Specifically, we use $\mathrm{GF}^2_{gl\%}$, $\mathrm{GF}^2_{loc\%}$ and GF^2_{pres} for the two-variable fragments.

2.4 Semi-linear sets

Since we will exploit the semi-linear characterization of Presburger constraints, we introduce some terminology. The term *vector* always means *row vectors*. For vectors $\bar{v}_0, \bar{v}_1, \ldots, \bar{v}_k \in \mathbb{N}^\ell$, we write $L(\bar{v}_0; \bar{v}_1, \ldots, \bar{v}_k)$ to denote the set:

$$L(\bar{v}_0; \bar{v}_1, \ldots, \bar{v}_k) \quad := \quad \left\{ \ \bar{u} \in \mathbb{N}^\ell \ \middle| \ \bar{u} = \bar{v}_0 + \textstyle\sum_{i=1}^{k} n_i \bar{v}_i \text{ for some } n_1, \ldots, n_k \in \mathbb{N} \ \right\}$$

A set $S \subseteq \mathbb{N}^\ell$ is a *linear* set, if $S = L(\bar{v}_0; \bar{v}_1, \ldots, \bar{v}_k)$, for some $\bar{v}_0, \bar{v}_1, \ldots, \bar{v}_k \in \mathbb{N}^\ell$. In this case, the vector $\bar{v}_0$ is called the *offset* vector of S, and $\bar{v}_1, \ldots, \bar{v}_k$ are called the *period* vectors of S. We denote by $\mathrm{offset}(S)$ the offset vector of S, i.e. $\bar{v}_0$ and $\mathrm{prd}(S)$ the set of period vectors of S, i.e. $\{\bar{v}_1, \ldots, \bar{v}_k\}$. A *semilinear* set is a finite union of linear sets.

The following theorem is a well-known result by Ginsburg and Spanier [13] which states that every set $S \subseteq \mathbb{N}^\ell$ definable by Presburger formula is a semilinear set. See [13] for the formal definition of Presburger formula.

▶ **Theorem 1** ([13]). *For every Presburger formula $\varphi(x_1, \ldots, x_\ell)$ with free variables $x_1, \ldots, x_\ell$, the set $\{\bar{u} \in \mathbb{N}^\ell \mid \varphi(\bar{u})$ holds in $\mathbb{N}\}$ is semilinear. Moreover, given the formula $\varphi(x_1, \ldots, x_\ell)$, one can effectively compute a set of tuples of vectors $\{(\bar{v}_{1,0}, \ldots, \bar{v}_{1,k_1}), \ldots, (\bar{v}_{p,0}, \bar{v}_{p,1}, \ldots, \bar{v}_{p,k_p})\}$ such that $\{\bar{u} \in \mathbb{N}^\ell \mid \varphi(\bar{u})$ holds in $\mathbb{N}\}$ is equal to $\bigcup_{i=1}^{p} L(\bar{v}_{i,0}; \bar{v}_{i,1}, \ldots, \bar{v}_{i,k_i})$.*

[2] Note that R in the subscript of a quantifier serves the role of a “guard”.

2.5 Types and neighbourhoods

A 1-*type* over a signature Σ is a maximally consistent set of unary predicates from Σ or their negations, where each atom uses only one variable x. Similarly, a 2-*type* over Σ is a maximally consistent set of binary predicates from Σ or their negations containing the atom $x \neq y$, where each atom or its negation uses two variables x and y.[3]

Note that 1-types and 2-types can be viewed as quantifier-free formulae that are the conjunction of their elements. We will use the symbols π and η (possibly indexed) to denote 1-type and 2-type, respectively. When viewed as formula, we write $\pi(x)$ and $\eta(x, y)$, respectively. We write $\pi(y)$ to denote formula $\pi(x)$ with x being substituted with y. The 2-type that contains only the negations of atomic predicates is called the *null* type, denoted by η_{null}. Otherwise, it is called a *non-null* type.

For a Σ-structure $\mathcal{M}$, the *type of an element* $a \in M$ is the unique 1-type π that a satisfies in $\mathcal{M}$. Similarly, the type of a pair $(a, b) \in M \times M$, where $a \neq b$, is the unique 2-type that (a, b) satisfies in $\mathcal{M}$. For an element $a \in M$, the η-*neighbourhood* of a, denoted by $\mathcal{N}_{\mathcal{M},\eta}(a)$, is the set of elements b such that η is the 2-type of (a, b). Formally,

$$\mathcal{N}_{\mathcal{M},\eta}(a) := \left\{ b \in M \;\middle|\; \mathcal{M}, x/a, y/b \models \eta(x, y) \right\}.$$

The η-*degree* of a, denoted by $\deg_{\mathcal{M},\eta}(a)$, is the cardinality of $\mathcal{N}_{\mathcal{M},\eta}(a)$.

Let $\eta_1, \ldots, \eta_\ell$ be an enumeration of all the non-null types. The *degree of* a *in* $\mathcal{M}$ is defined as the vector $\deg_{\mathcal{M}}(a) := (\deg_{\mathcal{M},\eta_1}(a), \cdots, \deg_{\mathcal{M},\eta_\ell}(a))$. Intuitively, $\deg_{\mathcal{M}}(a)$ counts the number of elements adjacent to a with non-null type. We note that our logic can be easily extended with atomic predicates of the form of a linear constraint C over the variables $\deg_\eta(x)$'s or $\deg(x) \in S$, where S is a semilinear set. Semantically, $\mathcal{M}, x/a \models C$ iff the linear constraint C evaluates to true when each $\deg_\eta(x)$ is substituted with $\deg_{\mathcal{M},\eta}(a)$ and $\mathcal{M}, x/a \models \deg(x) \in S$ iff $\deg_{\mathcal{M}}(a) \in S$. We stress that these atomic predicates will only be used to facilitate the proof of our decidability result.

3 Negative results

In this section we turn our attention to the negative results announced in the introduction.

3.1 Two-Variable Fragment

We start by proving that the two-variable fragment of FO extended with percentage quantification has undecidable finite satisfiability problem. Actually, in our proof, we will only use the $\exists^{=50\%}$ quantifier. Our results strengthen the existing undecidability proofs of $\mathcal{ALCISCC}^{++}$ from [4] and of FO^2 with equicardinality statements (implemented via the Härtig quantifier) from [18]. Roughly speaking, our counting mechanism is weaker: we cannot write arbitrary Presburger constraints (as it is done in [4]) nor compare sizes of any two sets (as it is done in [18]). Nevertheless, we will see that in our framework we can express “functionality” of a binary relation and “compare” cardinalities of sets, but under some technical assumptions of dividing the intended models into halves. Due to such technicality, we cannot simply encode the undecidability proofs of [4, 18] and we need to prepare our proof “from scratch”.

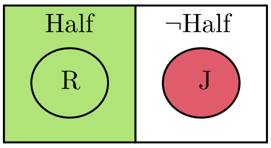
[3] We should remark here that the standard definition of 2-type, such as in [16, 26], a 2-type also contains unary predicates or its negation involving variable x or y. However, for our purpose, it is more convenient to define a 2-type as consisting of only binary predicates that strictly use both variables x and y. Note also that we view a binary predicate such as $R(x, x)$ as a unary predicate.

Our proof relies on encoding of Hilbert's tenth problem, whose simplified version is introduced below. In the classical version of *Hilbert's tenth problem* we ask whether a *diophantine equation*, i.e. a polynomial equation with integer coefficients, has a solution over $\mathbb{N}$. It is well-known that such problem is undecidable [22]. By employing some routine transformations (e.g. by rearranging terms with negative coefficients, by replacing exponentiation by multiplication and by introducing fresh variable for partial results of multiplications or addition), one can reduce any diophantine equation to an equi-solvable system of equations, where the only allowed operations are addition or multiplication of two variables or assigning the value one to some of them. We refer to the problem of checking solvability (over $\mathbb{N}$) of such systems of equations as SHTP (*simpler Hilbert's tenth problem*) and present its precise definition next. Note that, by the described reduction, SHTP is undecidable.

▶ **Definition 2** (SHTP). *An input of* SHTP *is a system of equations ε, where each of its entries ε_i is in one of the following forms: (i) $u_i = 1$, (ii) $u_i = v_i + w_i$, (iii) $u_i = v_i \cdot w_i$, where u_i, v_i, w_i are pairwise distinct* variables *from some countably infinite set* Var*. In* SHTP *we ask whether an input system of equations ε, as described before, has a solution over $\mathbb{N}$.*

3.1.1 Playing with percentage quantifiers

Before reducing SHTP to $\mathrm{FO}^2_{gl\%}$, let us gain more intuitions of $\mathrm{FO}^2_{gl\%}$ and introduce a useful trick employing percentage quantifiers to express equi-cardinality statements. Let $\mathcal{M}$ be a finite structure and let Half, R, J be unary predicates. We say that $\mathcal{M}$ is (*Half, R, J*)*-separated* whenever it satisfies the following conditions: (a) exactly half of the domain elements from $\mathcal{M}$ satisfy Half (b) the satisfaction of R implies the satisfaction of Half (c) the satisfaction of J implies the non-satisfaction of Half. Roughly speaking, the above conditions entail that the elements satisfying R and those satisfying J are in different halves of the model. We show that under these assumptions one can enforce the equality $|\mathrm{R}(x)|_{\mathcal{M}} = |\mathrm{J}(x)|_{\mathcal{M}}$. Indeed, such a property can be expressed in $\mathrm{FO}^2_{gl\%}$ with the following formula $\varphi_{\mathrm{eq}}(\mathrm{Half}, \mathrm{R}, \mathrm{J})$:

$\mathcal{A} :=$  $\models \varphi_{\mathrm{eq}}(\mathrm{Half}, \mathrm{R}, \mathrm{J}) := \exists^{=50\%}x\ (\mathrm{Half}(x) \wedge \neg\mathrm{R}(x)) \vee \mathrm{J}(x)$

For intuitions on $\varphi_{\mathrm{eq}}(\mathrm{Half}, \mathrm{R}, \mathrm{J})$, consult the above picture. We basically take all the elements satisfying Half (so exactly half of the domain elements, indicated by the green area). Next, we discard the elements labelled with R (so we get the green area without the circle inside) and replace them with the elements satisfying J (the red circle, note that $\mathrm{J}^{\mathcal{A}}$ and $\mathrm{R}^{\mathcal{A}}$ are disjoint!). The total number of selected elements is equal to half of the domain, thus $|\mathrm{J}^{\mathcal{M}}| = |\mathrm{R}^{\mathcal{M}}|$. The following fact is a direct consequence of the semantics of $\mathrm{FO}^2_{gl\%}$.

▶ **Fact 1.** *For* (*Half, R, J*)*-separated* $\mathcal{M}$ *we have* $\mathcal{M} \models \varphi_{eq}(Half, R, J)$ *iff* $|R(x)|_{\mathcal{M}} = |J(x)|_{\mathcal{M}}$.

3.1.2 Undecidability proof

Until the end of this section, let us fix ε, a valid input of SHTP. By $\mathrm{Var}(\varepsilon) = \{u, v, w, \ldots\}$ we denote the set of all variables appearing in ε, and with $|\varepsilon|$ we denote the total number of entries in ε. Let $\mathcal{M}$ be a finite structure.

The main idea of the encoding is fairly simple: in the intended model $\mathcal{M}$ some elements will be labelled with A_u predicates, ranging over variables $u \in \mathrm{Var}(\varepsilon)$, and the number of such elements will indicate the value of u in an example solution to ε. The only tricky part here is to encode multiplication of variables. Once ε contains an entry $w = u \cdot v$, we need to ensure that $|A_w(x)|_{\mathcal{M}} = |A_u(x)|_{\mathcal{M}} \cdot |A_v(x)|_{\mathcal{M}}$ holds. It is achieved by linking, via a binary relation $\mathrm{Mult}^{\mathcal{M}}$, each element from $A_u^{\mathcal{M}}$ with exactly $|A_v(x)|_{\mathcal{M}}$ elements satisfying A_w, which relies on imposing equicardinality statements. To ensure that the performed multiplication is correct, each element labelled with $A_w^{\mathcal{M}}$ has exactly one predecessor from $A_u^{\mathcal{M}}$ and hence the relation $\mathrm{Mult}^{\mathcal{M}}$ is backward-functional.

We start with a formula inducing a labelling of elements with variable predicates and ensuring that all elements of $\mathcal{M}$ satisfy at most one variable predicate. Note that it can happen that there will be auxiliary elements that are not labelled with any of the variable predicates.

($\varphi^{\varepsilon}_{\mathsf{var}}$) $\forall x \bigwedge_{u \neq v \in \mathrm{Var}(\varepsilon)} \neg(A_u(x) \wedge A_v(x))$.

We now focus on encoding the entries of ε. For an entry ε_i of the form $u_i = 1$ we write:

($\varphi_{u_i=1}$) $\exists x\, A_{u_i}(x) \wedge \forall x \forall y\, (A_{u_i}(x) \wedge A_{u_i}(y)) \to x = y$

▶ **Fact 2.** *$\mathcal{M} \models \varphi_{u_i=1}$ holds iff there is exactly one element in $\mathcal{M}$ satisfying $A_u(x)$.*
To deal with entries ε_i of the form $w_i = u_i + v_i$ or $w_i = u_i \cdot v_i$ we need to "prepare an area" for the encoding, similarly to Section 3.1.1. First, we cover domain elements of $\mathcal{M}$ by *layers*. The i-th layer is divided into halves with $\mathrm{FHalf}^{[i]}$ and $\mathrm{SHalf}^{[i]}$ predicates with:

($\varphi^{i}_{\mathsf{halves}}$) $\forall x \left(\mathrm{FHalf}^{[i]}(x) \leftrightarrow \neg\mathrm{SHalf}^{[i]}(x)\right) \wedge \exists^{=50\%} x\ \mathrm{FHalf}^{[i]}(x)$

▶ **Fact 3.** *$\mathcal{M} \models \varphi^{i}_{halves}$ holds iff exactly half of the domain elements from $\mathcal{M}$ are labelled with $\mathit{FHalf}^{[i]}$ and the other half of elements are labelled with $\mathit{SHalf}^{[i]}$.*

Second, we need to ensure that in the i-th layer of $\mathcal{M}$, the elements satisfying A_{u_i} or A_{v_i} are in the first half, whereas elements satisfying A_{w_i} are in the second half. We do it with:

($\varphi^{i}_{\mathsf{parti}}(u_i, v_i, w_i)$) $\forall x \left([(A_{u_i}(x) \vee A_{v_i}(x)) \to \mathrm{FHalf}^{[i]}(x)] \wedge [A_{w_i}(x) \to \mathrm{SHalf}^{[i]}(x)]\right)$

▶ **Fact 4.** *$\mathcal{M} \models \varphi^{i}_{parti}(u_i, v_i, w_i)$ holds iff for all elements $a \in M$, if a satisfies $A_{u_i}(x) \vee A_{v_i}(x)$ then a also satisfies $\mathit{FHalf}^{[i]}(x)$ and if a satisfies $A_{w_i}(x)$ then a also satisfies $\mathit{SHalf}^{[i]}(x)$.*

Gathering the presented formulae, we call a structure $\mathcal{M}$ *well-prepared*, if it satisfies the conjunction of all previous formulae over $1 \leq i \leq |\varepsilon|$ and over all entries ε_i from the system ε. The forthcoming encodings will be given under the assumption of *well-preparedness*.

Now, for the encoding of addition, assume that ε_i is of the form $u_i + v_i = w_i$. Thus in our encoding, we would like to express that $|A_{u_i}(x)|_{\mathcal{M}} + |A_{v_i}(x)|_{\mathcal{M}} = |A_{w_i}(x)|_{\mathcal{M}}$, which is clearly equivalent to $|A_{w_i}(x)|_{\mathcal{M}} - |A_{u_i}(x)|_{\mathcal{M}} - |A_{v_i}(x)|_{\mathcal{M}} = 0$ and also to $|A_{w_i}(x)|_{\mathcal{M}} + |\mathrm{FHalf}^{[i]}(x)|_{\mathcal{M}} - |A_{u_i}(x)|_{\mathcal{M}} - |A_{v_i}(x)|_{\mathcal{M}} = |\mathrm{FHalf}^{[i]}(x)|_{\mathcal{M}}$. Knowing that exactly 50% of domain elements of an intended model satisfy $\mathrm{FHalf}^{[i]}$ and that A_{u_i}, A_{v_i} and A_{w_i} label disjoint parts of the model, we can write the obtained equation as an $\mathrm{FO}^2_{gl\%}$ formula:

($\varphi^{i}_{\mathsf{add}}(u_i, v_i, w_i)$) $\exists^{=50\%} x \left(A_{w_i}(x) \vee (\mathrm{FHalf}^{[i]}(x) \wedge \neg A_{u_i}(x) \wedge \neg A_{v_i}(x))\right)$
Note that the above formula is exactly the $\varphi_{\mathrm{eq}}(\mathrm{Half}, \mathrm{R}, \mathrm{J})$ formula from Section 3.1.1, with $\mathrm{Half} = \mathrm{FHalf}^{[i]}(x)$, $\mathrm{J} = A_{w_i}$ and R defined as a union of A_{u_i} and A_{v_i}. Hence, we conclude:

▶ **Lemma 3.** *A well-prepared $\mathcal{M}$ satisfies $\varphi^{i}_{add}(u_i, v_i, w_i)$ iff $|A_{u_i}(x)|_{\mathcal{M}} + |A_{v_i}(x)|_{\mathcal{M}} = |A_{w_i}(x)|_{\mathcal{M}}$.*

The only missing part is the encoding of multiplication. Take ε_i of the form $u_i \cdot v_i = w_i$. As already described in the overview, our definition of multiplication requires three steps:

(link) A binary relation $\mathrm{Mult}_i{}^{\mathcal{M}}$ links each element from $A_{w_i}^{\mathcal{M}}$ to some element from $A_{u_i}^{\mathcal{M}}$.
(count) Each element from M satisfying $A_{u_i}(x)$ has exactly $|A_{v_i}(x)|_{\mathcal{M}}$ $\mathrm{Mult}_i{}^{\mathcal{M}}$-successors.
(bfunc) The binary relation $\mathrm{Mult}_i{}^{\mathcal{M}}$ is backward-functional.

Such properties can be expressed with the help of $\exists^{=50\%}$ quantifier, as presented below:

($\varphi^i_{\text{link}}(u_i, w_i)$) $\forall y\, A_{w_i}(y) \to \exists x\, \mathrm{Mult}_i(x, y) \wedge \forall x \forall y\, \mathrm{Mult}_i(x, y) \to (A_{u_i}(x) \wedge A_{w_i}(y))$
($\varphi^i_{\text{count}}(u_i, v_i, w_i)$) $\forall x\, A_{u_i}(x) \to \exists^{=50\%} y \left([\mathrm{SHalf}^{[i]}(y) \wedge \neg \mathrm{Mult}_i(x, y)] \vee A_{v_i}(y) \right)$
($\varphi^i_{\text{bfunc}}(u_i, v_i, w_i)$) $\forall x\, A_{w_i}(x) \to \exists^{=50\%} y \left([\mathrm{SHalf}^{[i]}(y) \wedge x \neq y] \vee \mathrm{Mult}_i(y, x) \right)$
While the first formula, namely $\varphi^i_{\text{link}}(u_i, w_i)$, is immediate to write, the next two are more involved. A careful reader can notice that they are actually instances of $\varphi_{\text{eq}}(\mathrm{Half}, \mathrm{R}, \mathrm{J})$ formula from Section 3.1.1. In the case of $\varphi^i_{\text{count}}(u_i, v_i, w_i)$ we have $\mathrm{Half} = \mathrm{SHalf}^{[i]}$, $\mathrm{J} = A_{v_i}$ and the Mult_i-successors of x play the role of elements labelled by R. For the last formula one can see that we remove exactly one element from $\mathrm{SHalf}^{[i]}$ (y that is equal to x) and we replace it with the Mult_i-predecessors of x, which implies that there is the unique such predecessor. We summarise the mentioned facts as follows:

▶ **Lemma 4.** *Let $\mathcal{M}$ be a well-prepared structure satisfying $\varphi^i_{link}(u_i, w_i)$. We have that (i) $\mathcal{M}$ satisfies $\varphi^i_{count}(u_i, v_i, w_i)$ iff every $a \in M$ satisfying A_{u_i} is connected via $Mult_i$ to exactly $|A_{v_i}|$ elements satisfying A_{w_i} and (ii) $\mathcal{M}$ satisfies $\varphi^i_{bfunc}(u_i, v_i, w_i)$ iff the binary relation $Mult_i{}^{\mathcal{M}}$ linking elements satisfying $A_{u_i}(x)$ with those satisfying $A_{w_i}(x)$ is backward-functional.*

Putting the last three properties together, we encode multiplication as their conjunction:

($\varphi^i_{\text{mult}}(u_i, v_i, w_i)$) $\quad \varphi^i_{\text{link}}(u_i, v_i, w_i) \wedge \varphi^i_{\text{count}}(u_i, v_i, w_i) \wedge \varphi^i_{\text{bfunc}}(u_i, v_i, w_i)$

▶ **Lemma 5.** *If a well-prepared $\mathcal{M}$ satisfies $\varphi^i_{mult}(u_i, v_i, w_i)$, then $|A_{u_i}(x)|_{\mathcal{M}} \cdot |A_{v_i}(x)|_{\mathcal{M}} = |A_{w_i}(x)|_{\mathcal{M}}$.*

Let $\varphi^{\varepsilon}_{\text{red}}$ be $\varphi^{\varepsilon}_{\text{var}}$ supplemented with a conjunction of formulae $\varphi^{\varepsilon_i}_{\text{entry}}$, where $\varphi^{\varepsilon_i}_{\text{entry}}$ is respectively: (i) $\varphi_{u_i=1}$ if ε_i is equal to u_i=1, (ii) $\varphi^i_{\text{halves}} \wedge \varphi^i_{\text{parti}}(u_i, v_i, w_i) \wedge \varphi^i_{\text{add}}(u_i, v_i, w_i)$ for ε_i of the form $u_i + v_i = w_i$ and (iii) $\varphi^i_{\text{halves}} \wedge \varphi^i_{\text{parti}}(u_i, v_i, w_i) \wedge \varphi^i_{\text{mult}}(u_i, v_i, w_i)$ for ε_i of the form $u_i \cdot v_i = w_i$. As the last piece in the proof we show that each solution of the system ε corresponds to some model of $\varphi^{\varepsilon}_{\text{red}}$. Its proof is routine and relies on the correctness of all previously announced facts (consult [7, Appendix B] for more details). Hence, by the undecidability of SHTP, we immediately conclude:

▶ **Theorem 6.** *The finite satisfiability problem for $\mathrm{FO}^2_{gl\%}$ is undecidable, even when the only percentage quantifier allowed is $\exists^{=50\%}$.*

Note that in our proof above, all the presented formulas can be easily transformed to formulae under the local semantics of percentage quantifiers as follows. First, we introduce a fresh binary symbol U and enforce it to be interpreted as the universal relation with $\forall x \forall y\, U(x, y)$. Then, we replace every occurrence of $\exists^{=50\%} x\, \varphi$ by $\exists_U^{=50\%} x\, \varphi$. Obviously, the resulting formula is FO^2 formula with local percentage quantifiers. Thus we conclude:

▶ **Corollary 7.** *The finite satisfiability problem for $\mathrm{FO}^2_{loc\%}$ is undecidable.*

3.2 Guarded Fragment

We now focus on the second seminal fragment of FO considered in this paper, namely on the guarded-fragment GF. We start from the global semantics of percentage quantifiers. Consider a unary predicate H, whose interpretation is constrained to label exactly half of the domain with $\exists^{=50\%}x\,\mathrm{H}(x)$. We then employ the formula

$$\forall x\; x = x \to \exists^{=50\%}y\,[\mathrm{U}(x,y) \wedge \mathrm{H}(y)] \wedge \exists^{=50\%}y\,[\mathrm{U}(x,y) \wedge \neg\mathrm{H}(y)]\,,$$

whose satisfaction by $\mathcal{M}$ entails that $U^{\mathcal{M}}$ is the universal relation. Hence, by putting U as a dummy guard in every formula in the undecidability proof of $\mathrm{FO}^2_{loc\%}$, we conclude:

▶ **Corollary 8.** *The finite satisfiability problem for* $\mathrm{GF}_{gl\%}$ *is undecidable, even when restricted to its two-variable fragment* $\mathrm{GF}^2_{gl\%}$*.*

It turns out that the undecidability still holds for GF once we switch from the global to the local semantics of percentage counting. In order to show it, we present a reduction from $\mathrm{GF}^3[F]$ (i.e. the three-variable fragment of GF with a distinguished binary F interpreted as a functional relation), whose finite satisfiability was shown to be undecidable in [15].

▶ **Theorem 9.** *The finite satisfiability problem for* $\mathrm{GF}_{loc\%}$ *(and even* $\mathrm{GF}^3_{loc\%}$*) is undecidable.*

Proof sketch. By reduction from $\mathrm{GF}^3[F]$ it suffices to express that F is functional. Let H, R be fresh binary relational symbols. We use a similar trick to the one from Section 3.1.1, where $H(x,\cdot)$ plays the role of Half (note that H may induce different partitions for different x), $R(\cdot,y)$ plays the role of R and y in $x = y$ plays the role of J.

The functionality of F can be expressed with:

$$\varphi_{func} := \forall x\; x = x \to [(\forall y\; F(x,y) \to R(x,y)) \wedge (\exists^{=50\%}_{R} y\; H(x,y)) \wedge$$
$$(\forall y\; F(x,y) \to (\neg H(x,y) \vee x = y)) \wedge (\exists^{=50\%}_{R} y\; ((H(x,y) \wedge x \neq y) \vee F(x,y)))]$$

In the appendix we will show that if $\mathcal{M} \models \varphi_{func}$ then F is indeed functional and every structure $\mathcal{M}$ with functional F can be extended by H and R, such that φ_{func} holds. ◀

The similar proof techniques do not work for GF^2, since GF^2 with counting is decidable [27]. Thus, in the forthcoming section we show that decidability status transfers not only to GF^2 with percentage counting, but also with Presburger arithmetics. This can be then applied to infer decidability of several modal and description logics, see [7, Appendix A].

4 Positive results

We next show that the finite satisfiability problem for GF^2_{pres} is decidable, as stated below.

▶ **Theorem 10.** *The finite satisfiability problem for* GF^2_{pres} *is decidable.*

It is also worth pointing out that Theorem 10 together with a minor modification of existing techniques [3] yields decidability of conjunctive query entailment problem for GF^2_{pres}, i.e. a problem of checking if an existentially quantified conjunction of atoms is entailed by GF^2_{pres} formula. This is a fundamental object of study in the area of logic-based knowledge representation. All the proofs and appropriate definitions are moved to [7, Appendix D].

▶ **Theorem 11.** *Finite conjunctive query entailment for* GF^2_{pres} *is decidable.*

The rest of this section will be devoted to the proof of Theorem 10, which goes by reduction to the two-variable fragment of FO with counting quantifiers $\exists^{=k}, \exists^{\leq k}$ for $k \in \mathbb{N}$ with their obvious semantics. Since the finite satisfiability of C^2 is decidable [26], Theorem 10 follows.[4]

4.1 Transforming GF^2_{pres} formulae into C^2

It is convenient to work with formulae in the appropriate normal form. Following a routine renaming technique (see e.g. [19]) we can convert in linear time a GF^2_{pres} formula into the following equisatisfiable normal form (over an extended signature):

$$\Psi_0 \quad := \quad \forall x\ \gamma(x) \wedge \bigwedge_{i=1}^{n} \Big(\forall x \forall y\ e_i(x,y) \rightarrow \alpha_i(x,y)\Big) \wedge \bigwedge_{i=1}^{m} \forall x \Big(\sum_{j=1}^{n_i} \lambda_{i,j} \cdot \#_y^{r_{i,j}}[x \neq y] \circledast \delta_i\Big),$$

where $\gamma(x)$ and each $\alpha_i(x,y)$ are quantifier-free formulae, each $e_i(x,y)$ is atomic predicate and all $\lambda_{i,j}$'s and δ_i's are integers, and $\circledast$ is as in Section 2.2.

Then, for every non-null type η, we replace each of the expressions $\#_y^{r_{i,j}}[x \neq y]$ with the sum of all the degrees $\deg_\eta(x)$ with η containing $r_{i,j}(x,y)$, i.e. the sum $\sum_{r_{i,j}(x,y)\in\eta} \deg_\eta(x)$. Moreover, since $\bigwedge \forall$ commutes, we obtain the following formula:

$$\Psi' \ := \ \forall x\ \gamma(x) \wedge \bigwedge_{i=1}^{n} \Big(\forall x \forall y\ e_i(x,y) \rightarrow \alpha_i(x,y)\Big) \wedge \forall x \bigwedge_{i=1}^{m} \Big(\sum_{j=1}^{n_i} \lambda_{i,j} \cdot \sum_{r_{i,j}(x,y)\in\eta} \deg_\eta(x) \ \circledast \ \delta_i\Big)$$

Note that the conjunction $\bigwedge_{i=1}^{m} \Big(\sum_{j=1}^{n_i} \lambda_{i,j} \cdot \sum_{r_{i,j}(x,y)\in\eta} \deg_\eta(x) \ \circledast \ \delta_i\Big)$ is a Presburger formula with free variables $\deg_\eta(x)$'s, for every non-null type η.[5] Thus, by Theorem 1, we can compute a set of tuples of vectors $\{(\bar{v}_{1,0}, \bar{c}_{1,1}, \ldots, \bar{v}_{1,k_1}), \ldots, (\bar{v}_{p,0}, \bar{v}_{p,1}, \ldots, \bar{v}_{p,k_p})\}$ and further rewrite Ψ' into the following formula:

$$\Psi = \forall x\ \gamma(x) \ \wedge \ \textstyle\bigwedge_{i=1}^{n} \Big(\forall x \forall y\ e_i(x,y) \rightarrow \alpha_i(x,y)\Big) \ \wedge \ \forall x \ \ \deg(x) \in S$$

where $S = \bigcup_{i=1}^{p} \ L(\bar{c}_{i,0}; \bar{c}_{i,1}, \ldots, \bar{c}_{i,k_i})$. We stress that technically Ψ is no longer in GF^2_{pres}.

In the following we will show how to transform Ψ into a C^2 formula Ψ^* such that they are (finitely) equi-satisfiable. For every $i = 1, \ldots, p$, let $S_i = L(\bar{v}_{i,0}; \bar{v}_{i,1}, \ldots, \bar{v}_{i,k_i})$. Recall that $\mathrm{offset}(S_i)$ is the offset vector $\bar{v}_{i,0}$ and $\mathrm{prd}(S_i)$ is the set of periodic vectors of S_i, i.e. $\{\bar{v}_{i,1}, \ldots, \bar{v}_{i,k_i}\}$. Consider the following formulae ξ and ϕ.

$$\xi \ := \ \forall x \bigvee_{i=1}^{p} \underline{\deg(x){=}\mathrm{offset}(S_i)} \ \vee \ \underline{\deg(x){\in}\mathrm{prd}(S_i)}, \quad \phi \ := \ \forall x \bigwedge_{i=1}^{p} \underline{\deg(x){\neq}\mathrm{offset}(S_i)} \rightarrow \exists y\ \varphi(x,y)$$

where $\varphi(x,y)$ is the conjunction expressing the following properties:

- The 1-types of x and y equal. It can be expressed with the formula $\bigwedge_U U(x) \leftrightarrow U(y)$, where U ranges over unary predicates appearing in Ψ.
- $\deg(x) \in \mathrm{prd}(S_j)$ and $\deg(y) = \mathrm{offset}(S_j)$ for some $1 \leq j \leq p$.

[4] Note that we propose a reduction into C^2, not into the *guarded* C^2, which might seem to be more appropriate. As we will see soon, a bit of non-guarded quantification is required in our proof.

[5] Technically speaking, in the standard definition of Presburger formula, the equality $f \equiv_d g$ is not allowed. However, it can be rewritten as $\exists x_1 \exists x_2 (f + x_1 d = g + x_2 d)$.

Note that $\underline{\deg(x) = \mathrm{offset}(S_i)}$ can be written as a C^2 formula. For example, if $\bar{v}_{i,0} = (d_1, \ldots, d_\ell)$, it is written as $\bigwedge_{j=1}^{\ell} \exists^{=d_j} y\ \eta_j(x,y)$. We can proceed with $\underline{\deg(x) \in \mathrm{prd}(S_i)}$ similarly, since $\mathrm{prd}(S_i)$ contains only finitely many vectors. Finally, we put Ψ^* to be

$$\Psi^* \quad := \quad \forall x\ \gamma(x) \ \wedge \ \bigwedge_{i=1}^{n} \forall x \forall y\ e_i(x,y) \rightarrow \alpha_i(x,y) \quad \wedge \quad \xi \quad \wedge \quad \phi.$$

We will show that Ψ and Ψ^* are finitely equi-satisfiable, as stated formally below.

▶ **Lemma 12.** *Ψ is finitely satisfiable if and only if Ψ^* is.*

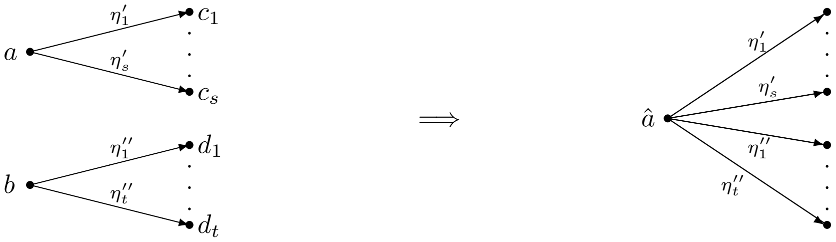
We delegate the proof of Lemma 12 to the next section. We conclude by stating that the complexity of our decision procedure is 3NExpTime. For more details of our analysis, see Section 4.3. Note that if we follow the decision procedure described in [13] for converting a system of linear equations to its semilinear set representation we will obtain a non-elementary complexity. This is because we need to perform $k-1$ intersections, where k is the number of linear constraints in the formula Ψ', and the procedure in [13] for handling each intersection yields an exponential blow-up. Instead, we use the results in [12, 25, 10] and obtain the complexity 3NExpTime, which though still high, falls within the elementary class.

4.2 Correctness of the translation

Before we proceed with the proof, we need to define some terminology. Let $\mathcal{M}$ be a finite model. Let $a, b \in A$ be such that the 2-type of (a,b) is η_{null}, i.e. the null-type and that a and b have the same 1-type. Suppose $c_1, \ldots, c_s$ are all elements such that the 2-type of each (a, c_j), denoted by η'_j, is non-null. Likewise, $d_1, \ldots, d_t$ are all the elements such that the 2-type of (b, d_j), denoted by η''_j, is non-null. Moreover, $c_1, \ldots, c_s, d_1, \ldots, d_t$ are pair-wise different.

"Merging" a and b into one new element $\hat{a}$ is defined similarly to the one in the graph-theoretic sense where a and b are merged into $\hat{a}$ such that the following holds.

- The 2-types of each $(\hat{a}, c_j)$ are equal to the original 2-types of (a, c_j), for all $j = 1, \ldots, s$.
- The 2-types of each $(\hat{a}, d_j)$ are equal to the original 2-types of (a, d_j), for all $j = 1, \ldots, t$.
- The 2-types of $(\hat{a}, a')$ is the null type, for every $a' \notin \{c_1, \ldots, c_2, d_1, \ldots, d_t\}$.
- The 1-type of $\hat{a}$ is the original 1-type of a (which is the same as the 1-type of b).



Note that we require that the original 2-type of (a,b) is the null type. Thus, after the merging, the degree of $\hat{a}$ is the sum of the original degrees of a and b. Moreover, the 1-type of $\hat{a}$ is the same as the original 1-type of a and b. Thus, if $\forall x \forall y\ e_i(x,y) \rightarrow \alpha_i(x,y)$ holds in $\mathcal{M}$, after the merging, it will still hold. Likewise, if $\forall x\ \gamma(x)$ holds in $\mathcal{M}$, it will still hold after the merging.

For the inverse, we define the "splitting" of an element $\hat{a}$ into two elements a and b as illustrated above, where the 1-type of a and b is the same as the 1-type of $\hat{a}$ and the 2-type of (a,b) is set to be η_{null}. After the splitting, the sum of the degrees of a and b is the same as the original degree of $\hat{a}$. Moreover, since the 2-type of (a,b) is η_{null}, $\mathcal{M}, x/a, y/b \not\models e_i(x,y)$. Thus, if $\forall x \forall y\ e_i(x,y) \rightarrow \alpha_i(x,y)$ holds in the original $\mathcal{M}$, it will still hold after the splitting.

▶ **Lemma 13.** *If Ψ is finitely satisfiable then Ψ^* is.*

Proof. Let $\mathcal{M}$ be a finite model of Ψ. We will construct a finite model $\mathcal{M}^* \models \Psi^*$ by splitting every element in $\mathcal{M}$ into several elements so that their degrees are either one of the offset vectors of S or one of the period vectors.

Let $a \in A$ and $\deg_{\mathcal{M}}(a) \in S_i$, for some $1 \leq i \leq p$. Suppose $\deg_{\mathcal{M}}(a) = \bar{v}_{i,0} + \sum_{j=1}^{k_i} n_j \bar{v}_{i,j}$, for some $n_1, \ldots, n_{k_i} \geq 0$. Let $N = 1 + \sum_{j=1}^{k_i} n_j$. We split a into N elements $b_1, \ldots, b_N$. Let $\mathcal{M}^*$ denote the resulting model after such splitting. Note that it should be finite since the degree of a is finite. It is straightforward to show that $\mathcal{M}^* \models \Psi^*$. ◀

▶ **Lemma 14.** *If Ψ^* is finitely satisfiable then Ψ is.*

Proof. Let $\mathcal{M}^*$ be a finite model of Ψ^*. Note that the degree of every element in $\mathcal{M}^*$ is either the offset vector or one of the period vectors of S_i, for some $1 \leq i \leq p$. To construct a finite model $\mathcal{M} \models \Psi$, we can appropriately "merge" elements so that the degree of every element is a vector in S_i, for some $1 \leq i \leq p$.

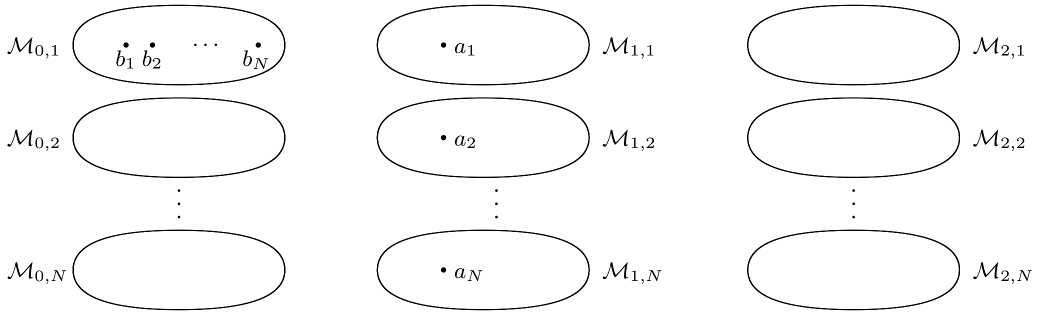
To this end, we call an element a in $\mathcal{M}^*$ a *periodic* element, if its degree is not an offset vector of some S_i. Let N be the number of periodic elements in $\mathcal{M}^*$. We make $3N$ copies of $\mathcal{M}^*$, which we denote by $\mathcal{M}_{i,j}$, where $0 \leq i \leq 2$ and $1 \leq j \leq N$. Let $\mathcal{M}$ be a model obtained by the disjoint union of all of $\mathcal{M}_{i,j}$'s, where for every b, b' that do not come from the same $\mathcal{M}_{i,j}$, the 2-type of (b, b') is the null-type.

We will show how to eliminate periodic elements in $\mathcal{M}$ by appropriately "merging" its elements. We need the following terminology. Recall that $S = S_1 \cup \cdots \cup S_p$, where each S_i is a linear set. For two vectors $\bar{u}$ and $\bar{v}$, we say that *$\bar{u}$ and $\bar{v}$ are compatible* (w.r.t. the semilinear set S), if there is S_i such that $\bar{u}$ is the offset vector of S_i and $\bar{v}$ is one of the period vectors of S_i. We say that two elements a and b in $\mathcal{M}$ are *merge-able*, if their 1-types are the same and their degrees are compatible.

We show how to merge periodic elements in $\mathcal{M}_{0,j}$, for every $j = 1, \ldots, N$.

- Let $b_1, \ldots, b_N$ be the periodic elements in $\mathcal{M}_{0,j}$.
- For each $l = 1, \ldots, N$, let a_l be an offset element in $\mathcal{M}_{1,l}$ such that every b_l and a_l are merge-able. (Such b_l exists, since $\mathcal{M}^*$ satisfies Ψ^* and each $\mathcal{M}_{i,j}$ is isomorphic to $\mathcal{M}^*$.)
- Then, merge a_l and b_l into one element, for every $l = 1, \ldots, k$.

See below, for an illustration for the case when $j = 1$.



Obviously, after this merging, there is no more periodic element in $\mathcal{M}_{0,j}$, for every $j = 1, \ldots, N$. We can perform similar merging between the periodic elements in $\mathcal{M}_{1,1} \cup \cdots \cup \mathcal{M}_{1,N}$ and the offset elements in $\mathcal{M}_{2,1} \cup \cdots \cup \mathcal{M}_{2,N}$, and between the periodic elements in $\mathcal{M}_{2,1} \cup \cdots \cup \mathcal{M}_{2,N}$ and the offset elements in $\mathcal{M}_{0,1} \cup \cdots \cup \mathcal{M}_{0,N}$.

After such merging, there is no more periodic element in $\mathcal{M}$ and the degree of every element is now a vector in S_i, for some $1 \leq i \leq p$. Moreover, since the merging preserves the satisfiability of $\forall x\ \gamma(x)$ and each $\forall x \forall y\ e_i(x, y) \to \alpha_i(x, y)$, the formula Ψ holds in $\mathcal{M}$. That is, Ψ is finitely satisfiable. ◀

4.3 Complexity analysis of the decision procedure

We need to introduce more terminology. For a vector/matrix X, we write $\|X\|$ to denote its L_∞-norm, i.e. the maximal absolute value of its entries. For a set of vector/matrices B, we write $\|B\|$ to denote $\max_{X \in B} \|X\|$.

Let $P = \{\bar{v}_1, \ldots, \bar{v}_k\} \subseteq \mathbb{N}^\ell$ be a finite set of (row) vectors of natural number components. To avoid clutter, we write $L(\bar{u}; P)$ to denote the linear set $L(\bar{u}; \bar{v}_1, \ldots, \bar{v}_k)$. For a finite set $B \subseteq \mathbb{N}^\ell$, we write $L(B; P)$ to denote the set $\bigcup_{\bar{u} \in B} L(\bar{u}; P)$.

We will use the following fact from [12, 25]. See also Proposition 2 in [10].

▶ **Proposition 15.** *Let $A \in \mathbb{Z}^{\ell \times m}$ and $\bar{c} \in \mathbb{Z}^m$. Let Γ be the space of the solutions of the system $\bar{x}A = \bar{c}$ (over the set of natural numbers $\mathbb{N}$).*[6] *Then, there are finite sets $B, P \subseteq \mathbb{N}^\ell$ such that the following holds.*

- $L(B; P) = \Gamma$.
- $\|B\| \leq ((m + 1)\|A\| + \|\bar{c}\| + 1)^\ell$.
- $\|P\| \leq (m\|A\| + 1)^\ell$.
- $|B| \leq (m + 1)^\ell$.
- $|P| \leq m^\ell$.

By repeating some of the vectors, if necessary, we can assume that Proposition 15 states that $|B| = |P| = (m + 1)^\ell$.

Proposition 15 immediately implies the following naïve construction of the sets B and P in deterministic double-exponential time (in the size of input A and $\bar{c}$).

- Enumerate all possible sets $B, P \subseteq \mathbb{N}^\ell$ of cardinality $(m + 1)^\ell$ whose entries are all bounded above by $((m + 1)\|A\| + \|\bar{c}\| + 1)^\ell$.
- For each pair B, P, where $P = \{\bar{v}_1, \ldots, \bar{v}_k\}$, check whether for every $i_1, \ldots, i_k \in \mathbb{N}$ and every $\bar{u} \in B$, the following equation holds.

$$(\bar{u} + \sum_{j=1}^{k} i_j \bar{v}_j)A \quad = \quad \bar{c}. \tag{1}$$

The number of bits needed to represent the sets B and P is $O(\ell^2 (m + 1)^\ell \log K)$, where $K = (m + 1)\|A\| + \|\bar{c}\| + 1$. Since Eq. 1 can be checked in deterministic exponential time (more precisely, it takes non-deterministic polynomial time to check if there is $i_1, \ldots, i_k$ such that Eq. 1 does not hold) in the length of the bit representation of the vectors in B, P, A and the vector $\bar{c}$, see, e.g. [24], constructing the sets B and P takes double-exponential time.

For completeness, we repeat the complexity analysis in Section 4. First, the formula Ψ_0 takes linear time in the size of the input formula. Constructing the formula Ψ' requires exponential time (in the number of binary predicates), i.e. $\ell = 2^k - 1$, where k is the number of binary predicates. Thus, constructing the sets B and P takes deterministic triple exponential time in the size of Ψ_0. However, the size of B and P is $O(2^{2k}(m + 1)^{2^k} \log K)$, i.e. double exponential in the size of Ψ_0. The C^2 formulas ξ and ϕ are constructed in polynomial time in the size of B and P. Since both the satisfiability and finite satisfiability of C^2 formulas is decidable in nondeterministic exponential time, we have another exponential blow-up. Altogether, our decision procedure runs in 3NExpTime.

[6] Recall that vectors in this paper are row vectors. So, $\bar{x}$ and $\bar{c}$ are row vectors of ℓ variables and m constants, respectively.

5 Concluding remarks

In the paper we studied the finite satisfiability problem for classical decidable fragments of FO extended with percentage quantifiers (as well as arithmetics in the full generality), namely the two-variable fragment FO^2 and the guarded fragment GF. We have shown that even in the presence of percentage quantifiers they quickly become undecidable.

The notable exception is the intersection of GF and FO^2, i.e. the two-variable guarded fragment, for which we have shown that it is decidable with elementary complexity, even when extended with local Presburger arithmetics. The proof is quite simple and goes via an encoding into the two-variable logic with counting (C^2). One of the bottlenecks in our decision procedure is the conversion of systems of linear equations into the semilinear set representations, which incurs a double-exponential blow-up. We leave it for future work whether a decision procedure with lower complexity is possible and/or whether the conversion to semilinear sets is necessary.

We stress that our results are also applicable to the unrestricted satisfiability problem (whenever the semantics of percentage quantifiers make sense), see [7, Appendix C].

References

1 Hajnal Andréka, István Németi, and Johan van Benthem. Modal Languages and Bounded Fragments of Predicate Logic. *J. Philosophical Logic*, 1998.

2 Franz Baader. A new description logic with set constraints and cardinality constraints on role successors. In Clare Dixon and Marcelo Finger, editors, *FroCoS*, 2017.

3 Franz Baader, Bartosz Bednarczyk, and Sebastian Rudolph. Satisfiability checking and conjunctive query answering in description logics with global and local cardinality constraints. In *DL*, 2019.

4 Franz Baader, Bartosz Bednarczyk, and Sebastian Rudolph. Satisfiability and query answering in description logics with global and local cardinality constraints. In *ECAI*, 2020.

5 Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic.* Cambridge University Press, 2017.

6 Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. *J. ACM*, 2015.

7 Bartosz Bednarczyk, Maja Orłowska, Anna Pacanowska, and Tony Tan. On Classical Decidable Logics extended with Percentage Quantifiers and Arithmetics. *CoRR*, abs/2106.15250, 2021. arXiv:2106.15250.

8 Michael Benedikt, Egor V. Kostylev, and Tony Tan. Two variable logic with ultimately periodic counting. In *ICALP 2020*, 2020.

9 Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer, 1997.

10 Dmitry Chistikov and Christoph Haase. The taming of the semi-linear set. In *ICALP*, 2016.

11 Stéphane Demri and Denis Lugiez. Complexity of modal logics with presburger constraints. *J. Appl. Log.*, 2010.

12 Eric Domenjoud. Solving systems of linear diophantine equations: An algebraic approach. In *MFCS*, 1991.

13 Seymour Ginsburg and Edwin Henry Spanier. Semigroups, presburger formulas, and languages. *Pac. J. of Math.*, 16:285–296, 1966.

14 Erich Grädel. Description logics and guarded fragments of first order logic. In *DL*, 1998.

15 Erich Grädel. On the restraining power of guards. *J. Symb. Log.*, 1999.

16 Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 1997.

17 Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *LICS*, 1997.

18 Erich Grädel, Martin Otto, and Eric Rosen. Undecidability results on two-variable logics. *Arch. Math. Log.*, 1999.

19 Yevgeny Kazakov. A polynomial translation from the two-variable guarded fragment with number restrictions to the guarded fragment. In *JELIA*, volume 3229 of *LNCS*, 2004.

20 Clemens Kupke and Dirk Pattinson. On modal logics of linear inequalities. In Lev D. Beklemishev, Valentin Goranko, and Valentin B. Shehtman, editors, *AIML*, 2010.

21 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

22 Yuri V. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.

23 Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity results for first-order two-variable logic with counting. *SIAM J. Comput.*, 29(4):1083–1117, 2000.

24 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.

25 Loic Pottier. Minimal solutions of linear diophantine systems: Bounds and algorithms. In *RTA*, 1991.

26 Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *J. Log. Lang. Inf.*, 14(3):369–395, 2005.

27 Ian Pratt-Hartmann. Complexity of the guarded two-variable fragment with counting quantifiers. *J. Log. Comput.*, 17(1):133–155, 2007.

28 Dana Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 1962.

29 B. Trakhtenbrot. The impossibility of an algorithm for the decidability problem on finite classes. In *Proc. USSR Acad. Sci.*, volume 70(4), pages 569–572, 1950.

30 Willard van Orman Quine. *The Ways of Paradox and Other Essays, Revised Edition*. Harvard University Press, 1976.

Branching Automata and Pomset Automata

Nicolas Bedon ✉
LITIS (EA 4108), University of Rouen, France

Abstract

We compare, in terms of expressive power, two notions of automata recognizing finite N-free pomsets: *branching automata* by Lodaya and Weil [7, 8, 9, 10] and *pomset automata* by Kappé, Brunet, Luttik, Silva and Zanasi [5]. In the general case, they are equivalent. We also consider sub-classes of both kind of automata that we prove equivalent.

2012 ACM Subject Classification Theory of computation → Regular languages

Keywords and phrases Finite N-free Pomsets, Finite Series-Parallel Pomsets, Branching Automata, Pomset Automata, Series-Parallel Rational Languages

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.37

Acknowledgements The author would like to thank the anonymous referees of this paper, whose comments helped in improving its quality.

1 Introduction

Automata are among the main tools in theoretical computer science. They are at the center of a large number of theoretical results and practical applications. Among them, let us cite as examples pattern matching, lexical analysis in compilers, and model-checking. In the latter, automata are used both for modeling sequential processes and to represent logical specifications. A state of an automaton represents a state of the system that is modeled, and transitions are used to change from states to states when an event occurs or an instruction is executed.

Inputs of automata as they were originally defined by Kleene [6] are finite words, that naturally model finite totally ordered sequences of events. Mainly motivated by the use of automata as a key argument in decidability algorithms in formal logic and circuits modeling, automata have quickly been extended to more complex inputs, such as for example infinite (ω) and transfinite words, terms, finite and infinite trees.

In this paper we focus on automata for languages of finite series-parallel pomsets. Informally speaking, a pomset is a word in which the total ordering of elements is not required. When A is an alphabet, finite words over A are the elements freely generated by A in the variety of monoids, and finite series-parallel pomsets over A are the elements freely generated by A in the variety of algebras $(X, \cdot, \|)$, with $(X, \cdot)$ a monoid and $(X, \|)$ a commutative monoid. Series-parallel pomsets have natural applications in computer science: when words are though of as traces of sequential executions of programs, series-parallel pomsets are traces of concurrent programs in which concurrency relies of the fork/join principle: a process forks into several concurrent parallel processes, waits for all of them to end their executions, and then continues its run. The class of series-parallel pomsets have an interesting characterisation in terms of sub-ordering: it coincides with that of N-free pomsets [11, 12].

In [7, 8, 9, 10], Lodaya and Weil introduced a class of automata on finite N-free pomsets, named *branching automata*, that extends Kleene automata with two kinds of unlabeled transitions: the *fork* and *join* transitions. A fork transition splits a path into several paths that run in parallel. When they are all finished, those parallel paths are grouped together with a join transition that goes into a single state. This join transition can be any of the join transitions: it does not depend on the definition of the branching automata, but must

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 37; pp. 37:1–37:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

be chosen consistently with the definition of a path; in particular it may not be unique, and may not exist. Lodaya and Weil defined rational expressions for this class, and studied the algebraic counterpart of branching automata.

Kappé, Brunet, Luttik, Silva and Zanasi [5] introduced another class of automata on finite N-free pomsets, named *pomset automata*. Their approach is an extension of Kleene automata by an additional kind of transitions that split a path into several paths that run in parallel, and define also the destination state to reach when all those parallel paths terminate. Whereas in the definition of branching automata the fork transition that starts parallel paths and the join transition that ends are not linked, both the start and the end are defined by the same transition in pomset automata. In the general case, languages of pomset automata are those of context-free grammars. Assuming a restriction on the definition of pomset automata, their languages are precisely the series-parallel rational languages, which are defined similarly to the usual rational languages of finite words with additional parallel product and a parallel iteration.

In this paper we compare branching and pomset automata. We first slightly generalize the original definition of branching automata by allowing the empty pomset in parallel parts of paths, and show that this corresponds to remove a condition on the rational expressions of the Kleene-like theorem of Lodaya and Weil. Under this generalisation, languages of branching automata are exactly the languages of context-free grammars. As a consequence, they are also exactly the languages of pomset automata. We finally characterize the sub-class of branching automata corresponding to series-parallel rational languages. All results are effective.

2 Notation and basic definitions

Let E be a set. We denote by $\mathcal{P}(E)$, $\mathcal{P}^+(E)$ and $\mathcal{M}^{>1}(E)$ respectively the set of subsets of E, the set of non-empty subsets of E and the set of multi-subsets of E with at least two elements. For any integer n, the set $\{1, \ldots, n\}$ is denoted $[n]$ and the group of permutations of $[n]$ by S_n. The cardinality of E is denoted by $|E|$. When $c = (x, y)$ is a pair, we denote by $\pi_1(c) = x$ and $\pi_2(c) = y$.

An alphabet is a set A whose elements are named *letters*. Since in this paper all alphabets are finite and non-empty we will omit to mention it. *Pomsets* (*partially ordered multi-sets*) are a generalization of words [2, 3, 13]. A *labeled poset* $(P, <_P, \rho_P)$ over an alphabet A consists of a set P, a partial ordering $<_P$ of the elements of P and a labeling map $\rho_P \colon P \to A$. For simplicity we often denote $(P, <_P, \rho_P)$ by P. Two labeled posets $(P, <_P, \rho_P)$ and $(Q, <_Q, \rho_Q)$ are *isomorphic* if there is a bijection from P to Q that preserves and reflects both labeling and ordering. A *pomset* P over A is (a representative of) an isomorphism class of labeled posets over A. The *width* of P is the maximal size of an antichain of P. Observe that the finite pomsets of width 1 correspond precisely to the usual finite words: finite totally ordered sequences of letters. The unique empty pomset is denoted by ϵ, and the unique pomset consisting of only one element labeled by $a \in A$ is simply denoted a. Since in this paper all labeled posets and pomsets are finite we omit to say it by now.

Let $(P, <_P, \rho_P)$ and $(Q, <_Q, \rho_Q)$ be two disjoint pomsets over respectively A and A'. The *parallel product* of P and Q, denoted $P \parallel Q$, is the pomset $(P \cup Q, <_P \cup <_Q, \rho_P \cup \rho_Q)$ over $A \cup A'$. The *sequential product* of P and Q, denoted by $P \cdot Q$ or PQ for simplicity, is the pomset $(P \cup Q, <_P \cup <_Q \cup P \times Q, \rho_P \cup \rho_Q)$ over $A \cup A'$. Observe that the parallel product is an associative and commutative operation on pomsets, whereas the sequential product does not commute (but is associative). The parallel and sequential products can be generalized to finite sequences of pomsets. Let $(P_i)_{i\leq n}$ be a finite sequence of pomsets. We denote by $\prod_{i\leq n} P_i = P_0 \cdot \dots \cdot P_n$ and $\|_{i\leq n} P_i = P_0 \parallel \dots \parallel P_n$.

The class of *series-parallel* pomsets over A, denoted $SP(A)$, is defined as the smallest class containing ϵ and a for all $a \in A$, and closed under finite parallel and finite sequential product. It is well known that this class corresponds precisely to the class of N-free pomsets [11, 12] over A, in which the exact ordering relation between any four elements x_1, x_2, x_3, x_4 cannot be $x_1 < x_2$, $x_3 < x_2$ and $x_3 < x_4$. We write $SP^+(A)$ for $SP(A) - \{\epsilon\}$. Note that for every pomset P of $SP(A)$ exactly one of the following is true: (i) $P = \epsilon$, (ii) $P = a \in A$, (iii) $P = RS$ or (iv) $P = R \parallel S$ for some non-empty pomsets R, S.

A *language* of $SP(A)$ is a sub-class of $SP(A)$. Sequential and parallel products are extended from pomsets to languages of pomsets in the usual way: when L and L' are languages of pomsets and *op* is either the sequential or the parallel product, then $L\ op\ L' = \{P\ op\ P' : P \in L, P' \in L'\}$.

Let A and B be two alphabets, $P \in SP(A)$, $L \subseteq SP(B)$ and $\xi \in A$. The language of $SP(A \setminus \{\xi\} \cup B)$ consisting of the pomset P in which each element labeled by the letter ξ is non-uniformly replaced by a pomset of L is denoted by $L \circ_\xi P$. By *non-uniformly* we mean that the elements labeled by ξ may be replaced by different elements of L. This substitution $L\circ_\xi$ is the homomorphism from $(SP(A), \|, \prod)$ into the power-set algebra $(\mathcal{P}(SP(A \cup B)), \|, \prod)$ with $\xi \mapsto L$ and $a \mapsto a$ for all $a \in A \setminus \{\xi\}$. Formally:

$$\begin{aligned} L \circ_\xi \epsilon &= \{\epsilon\} \\ L \circ_\xi a &= \begin{cases} \{a\} & \text{if } a \in A \setminus \{\xi\} \\ L & \text{if } a = \xi \end{cases} \\ L \circ_\xi (P_1 \cdot P_2) &= (L \circ_\xi P_1) \cdot (L \circ_\xi P_2) \\ L \circ_\xi (P_1 \parallel P_2) &= (L \circ_\xi P_1) \parallel (L \circ_\xi P_2) \end{aligned}$$

This operation can again be extended from pomsets to languages of pomsets by $L' \circ_\xi L = \cup_{P\in L} L' \circ_\xi P$.

▶ **Example 1.** Let $B = \{a, b\}$, $A = B \cup \{\xi\}$, $P = b \parallel (\xi \cdot \xi) \in SP(A)$ and $L = \{a \parallel b, b \cdot a\} \subseteq SP(B)$. Then $L \circ_\xi P = \{b \parallel ((a \parallel b) \cdot (a \parallel b)), b \parallel ((b \cdot a) \cdot (b \cdot a)), b \parallel ((a \parallel b) \cdot (b \cdot a)), b \parallel ((b \cdot a) \cdot (a \parallel b))\}$.

We also set

$$L^{*\xi} = \bigcup_{i\in\mathbb{N}} L^{i\xi} \text{ with } L^{0\xi} = \{\xi\} \text{ and } L^{(i+1)\xi} = (\bigcup_{j\leq i} L^{j\xi}) \circ_\xi L$$

$$L^* = \{\prod_{i<n} P_i : n \in \mathbb{N}, P_i \in L \text{ for each } i < n\} \qquad L^+ = L^* \setminus \{\epsilon\}$$

Assuming ξ is not used in L, we use the following abbreviation:

$$L^{\circledast} = \{\epsilon\} \circ_\xi (L \parallel \xi)^{*\xi} = \{\|_{i<n} P_i : n \in \mathbb{N}, P_i \in L \text{ for each } i < n\} \tag{1}$$

and $L^{\oplus} = L^{\circledast} \setminus \{\epsilon\}$. L^* and L^+ are the sequential iterations of L whereas $L^{\circledast}$ and $L^{\oplus}$ are its parallel iterations.

3 Branching Automata

Branching automata are a generalization of usual Kleene automata. They were introduced by Lodaya and Weil [7, 8, 9, 10]. A *branching automaton* is a tuple $\mathcal{A} = (Q, A, E, I, F)$ where Q is a finite set of states, A is an alphabet, $I \subseteq Q$ is the set of *initial states*, $F \subseteq Q$ the set of *final states*, and E is the finite set of *transitions* of $\mathcal{A}$. The set of transitions of E is partitioned into $E = (E_{\text{seq}}, E_{\text{fork}}, E_{\text{join}})$:

- $E_{\text{seq}} \subseteq Q \times A \times Q$ contains the *sequential* transitions, which are usual transitions of Kleene automata;
- $E_{\text{fork}} \subseteq Q \times \mathcal{M}^{>1}(Q)$ and $E_{\text{join}} \subseteq \mathcal{M}^{>1}(Q) \times Q$ are respectively the sets of *fork* and *join* transitions.

Sequential transitions $(p, a, q) \in Q \times A \times Q$ are sometimes denoted by $p \xrightarrow{a} q$. The *arity* of a fork (resp. join) transition $(p, R) \in Q \times \mathcal{M}^{>1}(Q)$ (resp. $(R, q) \in \mathcal{M}^{>1}(Q) \times Q$) is $|R|$. Here the *source* of a sequential or fork transition is p and the *destination* of a sequential or join transition is q.

We now turn to the definition of paths in $\mathcal{A}$. We give two definitions, namely *b-paths* and *b*-paths*, which are not equivalent: paths labeled by ϵ are allowed in the latter but not in the former. As we will see, considering ϵ as a possible label for paths changes the expressive power of branching automata.

3.1 b-regular and b-rational languages

We recall in this section the original definitions and basic results from Lodaya and Weil.

We define the relation $\underset{\mathcal{A}}{\rightarrow} \subseteq Q \times SP^{+}(A) \times Q$ as the smallest relation satisfying:

1. $p \underset{\mathcal{A}}{\xrightarrow{a}} q$ if and only if $(p, a, q) \in E$;
2. if $p \underset{\mathcal{A}}{\xrightarrow{P}} q$ and $q \underset{\mathcal{A}}{\xrightarrow{Q}} r$ then $p \underset{\mathcal{A}}{\xrightarrow{PQ}} r$;
3. for all integer $n > 1$, if $p_i \underset{\mathcal{A}}{\xrightarrow{P_i}} q_i$ for all $i \in [n]$, $(p, \{p_1, \dots, p_n\}) \in E_{\text{fork}}$, $(\{q_1, \dots, q_n\}, q) \in E_{\text{join}}$ then $p \underset{\mathcal{A}}{\xrightarrow{\|_{i \in [n]} P_i}} q$.

If $p \underset{\mathcal{A}}{\xrightarrow{P}} q$ we say that there is a *b-path* from p to q labeled by P in $\mathcal{A}$.

A *b-path* is an equivalence class of (finite) terms over $X = \{p \underset{\mathcal{A}}{\xrightarrow{a}} q : (p, a, q) \in E\}$ using (2) and (3) in the definition above as composition rules, in which terms are equivalent up to the associativity of (2) and to the ordering of the multi-sets $\{p_1, \dots, p_n\}$ and $\{q_1, \dots, q_n\}$ in the fork and join transitions of (3). Thus, the signature of terms is $X \cup \{\cdot\} \cup_{n>1} E_{\text{fork,n}} \times E_{\text{join,n}}$, where elements of X are symbols of arity 0, $\cdot$ has arity 2, and the elements of $E_{\text{fork,n}} \times E_{\text{join,n}}$, pairs of a fork and a join transition of same arity n, have arity n. When Rule (3) is used to form a term t from n terms using fork and join transitions f and j we say that t is a *parallel term rooted* by (f, j). When Rule 2 is used to form t from two terms then t is *sequential*. Each term or b-path t naturally evaluates into a unique $p \underset{\mathcal{A}}{\xrightarrow{P}} q$ (in this case t is from p to q labeled by P in $\mathcal{A}$). Reciprocally, each element of $\underset{\mathcal{A}}{\rightarrow}$ is the evaluation of at least one term (or b-path). We denote by $t' \preceq t$ (resp. $t' \prec t$) when t' is a (resp. strict) sub-term of the term t. A term t *uses* a transition u if u is a sequential transition used to form t or u is a fork or a join transition and there is some $t' \preceq t$ rooted by some (f, j) with either $f = u$ or $j = u$. It uses a state q if q appear in a transition used in t. Let f and j be respectively a fork and a join transition. A term t uses (f, j) *at the upper level* if there is some $t' \preceq t$ rooted by (f, j) and if $t' \preceq t'' \preceq t$ for some t'' rooted by some (f', j') then t'' is t'. It uses (f, j) *at a sequential level* if there are some $t'' \preceq t' \preceq t$ with t'' rooted by (f, j) and t' a sequential term. Observe that two terms of the same b-path p use exactly the same transitions and states, which allows us to say that p *uses* a transition u (or a state r) if there is some term (or equivalently, if all terms) of p that uses u (or r). The same remark applies for pairs of fork and join transitions used at the upper level or at a sequential level, and can be used to naturally qualify a b-path to be *parallel* or *sequential*.

Set $L_{p,q} = \{P \in SP^+(A) : p \xrightarrow[\mathcal{A}]{P} q\}$. The *language* of $\mathcal{A}$ is $L(\mathcal{A}) = \cup_{(i,f)\in I\times F} L_{i,f}$. We call *b-automaton* a branching automaton equipped with the notion of b-path above. A language $L \subseteq SP^+(A)$ is *b-regular* if L is the language of some b-automaton.

The class of *b-rational* languages of $SP^+(A)$ is the smallest containing $\emptyset$, $\{a\}$ for all $a \in A$, and closed under the operations of $S_b = \{\cup, \cdot, ^+, \|, \circ_\xi, ^{*\xi}\}$, provided that

▶ Condition 1. In $L^{*\xi}$ any element labeled by ξ in some $P \in L$ is incomparable with another element of P.

In particular, Condition 1 excludes from the b-rational languages those of the form $(a\xi b)^{*\xi} = \{a^n \xi b^n : n \in \mathbb{N}\}$, for example.

Let S be a set of functions of arity > 0 on languages. A *S-rational expression* e is a well-formed term of signature $\{\emptyset\} \cup A \cup S$ denoting a language $L(e)$. The *b-rational expressions* are the S_b-rational expressions (verifying Condition 1).

▶ **Theorem 2** ([7]). *A language of $SP^+(A)$ is b-regular if and only if it is b-rational.*

Condition 1 is mandatory in the proof of Theorem 2. That ϵ is forbidden in labels for the parallel composition of b-paths (in Item 3 of the definition of $\xrightarrow[\mathcal{A}]{}$ each P_i is different from ϵ) is also mandatory.

3.2 b*-regular and b*-rational languages

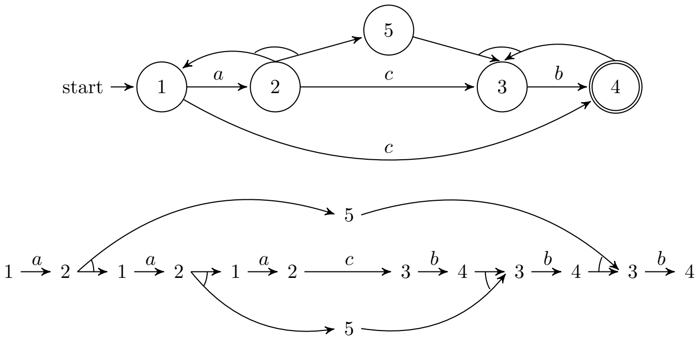
In this section we slightly modify the definition of a b-path by allowing ϵ as a label, in particular in parallel parts.

We define the relation $\xrightarrow[\mathcal{A}]{} \subseteq Q \times SP(A) \times Q$ as the smallest relation satisfying:

1. $p \xrightarrow[\mathcal{A}]{\epsilon} p$ for all $p \in Q$;
2. $p \xrightarrow[\mathcal{A}]{a} q$ if and only if $(p, a, q) \in E$;
3. if $p \xrightarrow[\mathcal{A}]{P} q$ and $q \xrightarrow[\mathcal{A}]{Q} r$ then $p \xrightarrow[\mathcal{A}]{PQ} r$;
4. for all integer $n > 1$, if $p_i \xrightarrow[\mathcal{A}]{P_i} q_i$ for all $i \in [n]$, $(p, \{p_1, \dots, p_n\}) \in E_{\text{fork}}$, $(\{q_1, \dots, q_n\}, q) \in E_{\text{join}}$ then $p \xrightarrow[\mathcal{A}]{\|_{i\in[n]} P_i} q$.

The notions of b*-path, b*-automaton, b*-regularity, etc. are defined similarly as in Section 3.1 by a replacement of the relation $\xrightarrow[\mathcal{A}]{}$ with the definition above. As in b-automata, there is no ϵ-transition in a b*-automaton. However, $p \xrightarrow[\mathcal{A}]{\epsilon} q$ with $p \neq q$ is possible using Item 4 with $p_i \xrightarrow[\mathcal{A}]{\epsilon} q_i$ and $p_i = q_i$ for all $i \in [n]$. In a b*-automaton, we have $L_{p,q} = \{P \in SP(A) : p \xrightarrow[\mathcal{A}]{P} q\}$ and $L(\mathcal{A}) = \cup_{(i,f)\in I\times F} L_{i,f}$. Note that because of Item 1 in the definition of the relation $\xrightarrow[\mathcal{A}]{}$ above, there are b*-paths of the form $p \xrightarrow[\mathcal{A}]{\epsilon} p$ that do not use any transition. Such b*-paths are named *trivial*. A b*-path t uses a pair (f, j) of a fork and a join transition at a sequential level if there are some $t'' \preceq t' \preceq t$ with t'' rooted by (f, j) and t' of the form $t' = t'_1 \cdot t'_2$ with t'_1, t'_2 *both non-trivial*.

▶ **Example 3.** Let $A = \{a, b, c\}$, $L = \{a^n cb^n : n \geq 0\}$, and $\mathcal{A}$ be the b*-automaton pictured in Figure 1. Then $L(\mathcal{A}) = L$. Note that L is not b-regular, thus the class of b-regular languages is strictly included into the class of b*-regular ones.



■ **Figure 1** On the top, a b*-automaton $\mathcal{A}$ with $L(\mathcal{A}) = \{a^n cb^n : n \geq 0\}$. The only fork transition is $(2, \{1, 5\})$, the only join transition $(\{4, 5\}, 3)$, the only initial state is 1 and the only final state 4. At the bottom, a representation of a b*-path labeled by *aaacbbb*.

A *(pomset) context-free grammar*, or CFG for short, $G = (T, N, S, R)$ is given by finite sets T of *terminals*, N of *non-terminals*, R of *rules* (or *productions*) and an *axiom* $S \in N$. Rules are of the form $X \to u$ with $X \in N$ and u a finite term built from $N \cup T \cup \{\epsilon\}$ with the sequential and parallel products as operations. The *language* $L(G)$ of G is defined with the axiom S as a start symbol as usual.

▶ **Theorem 4.** *A language of $SP(A)$ is context-free if and only if it is b*-regular.*

Proof. First consider a language $L \subseteq SP(A)$ with $L = L(G)$ for some context-free grammar $G = (T, N, S, R)$. Up to usual transformations, we may assume that there there is at most one production whose right member is ϵ, if there is such a production it is $S \to \epsilon$, and that the axiom S does not appear in any of the right member of the productions in R. For each rule $X \to u \in R$, $X \in N$, build a b*-automaton $\mathcal{A}_{X \to u}$ on the alphabet $T \cup N$ such that $L(\mathcal{A}_{X \to u}) = \{u\}$. For each $X \in N$, build a b*-automaton $\mathcal{A}_X$ from the disjoint union of all $\mathcal{A}_{X \to u}$, $X \to u \in R$. Now build a b*-automaton $\mathcal{A}_G$ such that $L = L(\mathcal{A}_G)$ as follows. For all $X \in N \setminus \{S\}$, take 2 copies $\mathcal{A}_{X,1}$ and $\mathcal{A}_{X,2}$ of $\mathcal{A}_X$. Consider the disjoint union $\mathcal{A}_G$ of $\mathcal{A}_S$ and of all b*-automata $\mathcal{A}_{X,i}$, $x \in N$, $i \in [2]$. For each transition $t = (p, X, q)$, $X \in N$, in $\mathcal{A}_S$ or $\mathcal{A}_{Y,i}$, $Y \neq X$, $i \in [2]$, add a new state t, a fork transition $(p, \{s, t\})$ for each initial state s of $\mathcal{A}_{X,1}$, and a join transition $(\{s', t\}, q)$ for each final state s' of $\mathcal{A}_{X,1}$. Remove the transition t. For each transition $t = (p, X, q)$, $X \in N$, in $\mathcal{A}_{X,i}$, $i \in [2]$, add a new state t, a fork transition $(p, \{s, t\})$ for each initial state s of $\mathcal{A}_{X,j}$, $j \neq i$, and a join transition $(\{s', t\}, q)$ for each final state s' of $\mathcal{A}_{X,j}$. Remove the transition t. The initial and final states of $\mathcal{A}_G$ are taken from $\mathcal{A}_S$. The accepting b*-paths of $\mathcal{A}_G$ are precisely those of $\mathcal{A}_S$ is which each use of a transition (p, X, q), $X \in N$, is replaced by an accepting path of $\mathcal{A}_X$. Immediately, we get $L(\mathcal{A}_G) = L(G)$.

Let us turn to the other direction. Consider a b*-automaton $\mathcal{A}$, and for each pair (p, q) of its states consider the language $L_{p,q}$ of the labels of b*-paths from p to q. Following a McNaughton-Yamada like construction, we build a finite system S of equalities where each $L_{p,q}$ is expressed as a term depending of the $L_{r,s}$, the letters of the alphabet, union, parallel and sequential composition. We refer to [7, Proof of Theorem 6] for the construction of such S. The system S can be easily transformed into a CFG G with $L(G) = L(\mathcal{A})$. ◀

As a consequence, the class of b*-regular languages of $SP(A)$ is not closed under boolean operations, whereas the class of b-regular languages of $SP^+(A)$ is [1].

The class of *b*-rational* languages of $SP(A)$ is the smallest containing $\emptyset$, $\{a\}$ for all $a \in A$, and closed under $S_{b*} = \{\cup, \cdot, {}^*, \|, \circ_\xi, {}^{*\xi}\}$. This definition is the same as b-rational languages, except that sequential iteration $^+$ has been replaced by * and thus ϵ is taken into consideration, and that the restriction expressed by Condition 1 has been removed.

▶ **Example 5.** The language L of Example 3 is given by the b*-rational expression $L = c \circ_\xi (a\xi b)^{*\xi}$.

Observe that the usual Kleene rational languages of A^* are a particular case of the b*-rational languages of $SP(A)$, in which the operators $\|$, $\circ_\xi$ and $^{*\xi}$ are not allowed. The class of *commutative rational* languages of $A^{\circledast}$ (or *over* A), which is the smallest containing $\emptyset$, $\{a\}$ for all $a \in A$, and closed under $\cup$, $\|$ and $^{\circledast}$, is also a particular case of the b*-rational languages of $SP(A)$ (recall Equalities (1)).

▶ **Theorem 6.** *A language of $SP(A)$ is b*-regular if and only if it is b*-rational.*

Proof. First we build a b*-automaton $\mathcal{A}_e$ from a b*-rational expression e such that $L(\mathcal{A}_e) = L(e)$. Using Theorem 4 it suffices to build a CFG G from e, such that $L(G) = L(e)$. This is done by induction over e. For the cases where e has one of the form $e = \emptyset$, $e = \{\epsilon\}$, $e = \{a\}$ with $a \in A$, $e = e_1 \cup e_2$, $e = e_1 \cdot e_2$, $e = e_1 \| e_2$, $e = f^*$, the CFG is directly obtained using the induction hypothesis and usual techniques, so we focus on $e = e_1 \circ_\xi e_2$ and $e = f^{*\xi}$. First assume $e = e_1 \circ_\xi e_2$ and that by induction hypothesis we have two CFG $G_i = (A, N_i, S_i, R_i)$ with $L(G_i) = L(e_i)$, $i \in [2]$. Build $G = (A, N_1 \cup N_2 \cup \{X_\xi\}, S_2, R_1 \cup R \cup \{X_\xi \to \xi, X_\xi \to S_1\})$ in which $X_\xi \notin N_1 \cup N_2$ is a new non-terminal and R is R_2 is which every occurrence of the terminal ξ has been replaced by X_ξ. Then $L(G) = L(e_1 \circ_\xi e_2)$. Assume now $e = f^{*\xi}$ for some b*-rational expression f and let $G_f = (A, N_f, S_f, R_f)$ be the CFG build from f by induction hypothesis. Let $G = (A, N_f, S_f, R \cup \{S_f \to \xi\})$ where R is R_f is which every occurrence of the terminal ξ has been replaced by S_f. Then $L(G) = L(f^{*\xi})$.

Now let $\mathcal{A}$ be a b*-automaton. The proof that $L(\mathcal{A})$ is b*-rational uses exactly the same arguments as those of the direction from left to right of Theorem 2. ◀

We will need later the following particular form of branching automata, adapted from [7] to our case. A b*-automaton $\mathcal{A}$ is *misbehaved* if it has a fork transition $(p, \{p_1, \ldots, p_n\})$ such that $p_j \xrightarrow[\mathcal{A}]{P} f$ for some $j \in [n]$, P and final state f, or if it has a join transition $(\{p_1, \ldots, p_n\}, p)$ such that $i \xrightarrow[\mathcal{A}]{P} p_j$ for some $j \in [n]$, P and initial state i. If $\mathcal{A}$ is not misbehaved then it is *behaved.*

▶ **Proposition 7.** *For every b*-automaton $\mathcal{A}$ there is a behaved b*-automaton $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$.*

Proof. For each fork transition $f = (p, \{p_1, \ldots, p_n\})$ we take n copies $(\mathcal{A}_{f,i})_{i \in [n]}$ of $\mathcal{A}$. The b*-automaton $\mathcal{B}$ is the disjoint union of these copies with another copy $\mathcal{A}_0$. Delete all the fork and join transitions from $\mathcal{A}_0$. For each fork transition $f = (p, \{p_1, \ldots, p_n\})$ of $\mathcal{A}$, we add to $\mathcal{B}$ a fork transition $(p, \{p_1, \ldots, p_n\})$ where p is taken in $\mathcal{A}_0$ and for all $i \in [n]$, p_i is taken in $\mathcal{A}_{f,i}$. For each join transition $j = (\{q_1, \ldots, q_n\}, q)$, we add all the possible join transitions simulating j where q is taken in $\mathcal{A}_0$ and all the q_i are taken in the different copies $(\mathcal{A}_{j,i})_{i \in [n]}$. It can be verified that if the initial and final states of $\mathcal{B}$ are those of $\mathcal{A}$ taken in $\mathcal{A}_0$, then $\mathcal{B}$ is behaved and that $L(\mathcal{B}) = L(\mathcal{A})$. ◀

The definitions and results about behaveness also trivially apply to b-automata.

4 Pomset Automata

Pomset automata are also a generalization of usual Kleene automata, introduced by Kappé, Brunet, Luttik, Silva and Zanasi [5]. A *pomset automaton* is a tuple $\mathcal{A} = (Q, A, E, \{i\}, F)$ where Q is a finite set of states, A is an alphabet, i is the *initial state*, $F \subseteq Q$ the set of *final states*, and E forms the *transitions* of $\mathcal{A}$. The transitions E consists in two functions:

- $E_{\text{seq}} \colon Q \times A \to Q$ is the *sequential* transition function, as for usual transitions in complete deterministic Kleene automata;
- $E_{\text{par}} \colon Q \times Q \times Q \to Q$ is the *parallel transition function.*

We define the relation $\xrightarrow[\mathcal{A}]{} \subseteq Q \times SP(A) \times Q$ as the smallest relation satisfying:

1. $p \xrightarrow[\mathcal{A}]{\epsilon} p$;
2. $p \xrightarrow[\mathcal{A}]{a} E_{\text{seq}}(p, a)$;
3. if $p \xrightarrow[\mathcal{A}]{P} q$ and $q \xrightarrow[\mathcal{A}]{Q} r$ then $p \xrightarrow[\mathcal{A}]{PQ} r$;
4. if $p \xrightarrow[\mathcal{A}]{P} q \in F$ and $r \xrightarrow[\mathcal{A}]{Q} s \in F$ then $t \xrightarrow[\mathcal{A}]{P\|Q} E_{\text{par}}(t, p, r)$.

When presenting a pomset automaton $\mathcal{A}$, we may define the transition function only partially and implicitely assume the existence of an additional sink state $\bot$ (if $\bot \xrightarrow[\mathcal{A}]{P} q$ for some P then $q = \bot$) and a final state $\top$ such that all transitions from $\top$ go to $\bot$.

If $p \xrightarrow[\mathcal{A}]{P} q$ we say that there is a p-path from p to q labeled by P in $\mathcal{A}$. We call *p-automaton* a pomset automaton equipped with the notion of p-path defined as in Section 3.1 but with the relation $\xrightarrow[\mathcal{A}]{}$ above. The *language* of $\mathcal{A}$ is $L(\mathcal{A}) = \{P \in SP(A) : i \xrightarrow[\mathcal{A}]{P} q \text{ for some } q \in F\}$. A language $L \subseteq SP(A)$ is *p-regular* if L is the language of some p-automaton.

▶ **Example 8.** A p-automaton with same language as the b*-automaton of Example 3 is pictured in Figure 2. For simplicity we do not consider the states $\bot$ and $\top$.

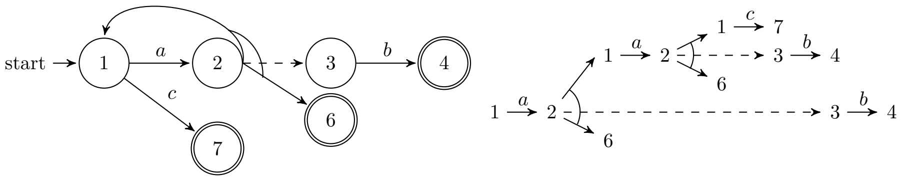


Figure 2 On the left, a p-automaton $\mathcal{A}$ with $L(\mathcal{A}) = \{a^n cb^n : n \geq 0\}$. The parallel transition function is $E_{\text{par}} \colon (2, 1, 6) \to 3$, the only initial state is 1 and the final states are 4,6,7. On the right, a representation of a p-path labeled by *aacbb*.

It is to notice that the transitions in branching automata are in the definition given by relations, whereas the transitions in pomset automata are functions. However, this does not mean that $p \xrightarrow[\mathcal{A}]{P} r$ and $p \xrightarrow[\mathcal{A}]{P} s$ implies $r = s$ in a p-automaton $\mathcal{A}$: consider for example that $\mathcal{A}$ may have different p-paths starting from a state q and labeled with $P = a \parallel b \parallel c$: one composing p-paths $p_1 \xrightarrow[\mathcal{A}]{a} p_2$ and $p_3 \xrightarrow[\mathcal{A}]{b\|c} p_4$ using a transition $(p, p_1, p_3) \to r$, and another composing some $q_1 \xrightarrow[\mathcal{A}]{a\|b} q_2$ and $q_3 \xrightarrow[\mathcal{A}]{c} q_4$ using a transition $(p, q_1, q_3) \to s$.

▶ **Theorem 9** ([5]). *A language of $SP(A)$ is context-free if and only if it is p-regular.*

As an immediate corollary of Theorems 4 and 9, b*-automata and p-automata have the same expressive power:

▶ **Corollary 10.** *A language of $SP(A)$ is b*-regular if and only if it is p-regular.*

5 Series-parallel rational languages

The class of *series-parallel rational* languages of $SP(A)$ is the smallest containing $\emptyset$, $\{a\}$ for all $a \in A$, and closed under $S_{sp} = \{\cup, \cdot, ^*, \|, ^{\circledast}\}$. As a consequence of Equalities (1), any series-parallel rational language of $SP(A)$ is also b*-rational, and any series-parallel rational language of $SP^+(A)$ (ϵ not considered) is also b-rational. As noticed in the conclusion of [8], the inclusion is strict, since for example $a \circ_\xi (a \parallel (a\xi))^{*\xi}$ is b-rational but not series-parallel rational.

In the conclusion of [9] the authors left open the question of a necessary and sufficient condition on a b-automaton $\mathcal{A}$ for $L(\mathcal{A})$ to be series-parallel rational. We answer this question in this section with b*-automata. The result also applies to b-automata, provided that ϵ is not taken into consideration on both automata and rational expressions sides (for example $^{\circledast}$ and * have to be replaced by respectively $^{\oplus}$ and $^+$ in the definition of series-parallel rationality above).

A language is *series-parallel regular* if it is the language of some b*-automaton $\mathcal{A}$ verifying Condition 2 below.

▶ Condition 2. There is no b*-path p rooted by some pair (f, j) of a fork and a join transition, such that p uses (f, j) at a sequential level.

Whether a b*-automaton verifies Condition 2 or not is decidable using methods similar to those developed in [10].

▶ **Theorem 11.** *A language L of $SP(A)$ is series-parallel regular if and only if it is series-parallel rational.*

Proof. From right to left we proceed by induction over a series-parallel rational expression e with $L = L(e)$. Since the construction given in the proof of Proposition 7 preserves Condition 2 we may assume that b*-automata constructed at induction steps are behaved. The cases where e has an elementary form, or $e = e_1 \cup e_2$ for some e_1, e_2 are as usual in automata theory. Assume $e = e_1 \cdot e_2$; by induction hypothesis we have behaved b*-automata $\mathcal{A}_1$ and $\mathcal{A}_2$ for respectively e_1 and e_2. Consider the disjoint union $\mathcal{A}$ of $\mathcal{A}_1$ and $\mathcal{A}_2$. For each final state f of $\mathcal{A}_1$, initial state i of $\mathcal{A}_2$ and sequential or fork transition t of source i, duplicate t by replacing the source i with f. The initial states of the resulting b*-automaton are those of $\mathcal{A}_1$. The final states are those of $\mathcal{A}_2$ and in addition the final states of $\mathcal{A}_1$ when $\epsilon \in L(\mathcal{A}_2)$. Assume now $e = e'^*$ and let $\mathcal{A}'$ be a behaved b*-automaton for e'. For each final state f, initial state i, sequential and fork transition t of source i, duplicate t by replacing the source i with f. The initial states are those of $\mathcal{A}'$, the final states are those of $\mathcal{A}'$ plus the initial states. When $e = e_1 \parallel e_2$ the construction is the disjoint union of $\mathcal{A}_1$ and $\mathcal{A}_2$ with a unique initial new state i, a unique final new state f, for each initial states i_1 and i_2 of respectively $\mathcal{A}_1$ and $\mathcal{A}_2$ a new normal fork transition $(i, \{i_1, i_2\})$, for each final states f_1 and f_2 of respectively $\mathcal{A}_1$ and $\mathcal{A}_2$ a new join transition $(\{f_1, f_2\}, f)$. Assume finally $e = e'^{\circledast}$. We build from $\mathcal{A}'$ a b*-automaton $\mathcal{A}$ for e as follows. Let T be the set of all sequential or join transitions whose destination is a final state of $\mathcal{A}'$. Let $\mathcal{A}'_0$ and $\mathcal{A}'_t$, $t \in T$, be copies of $\mathcal{A}'$. We build $\mathcal{A}$ from the disjoint union of these copies. Add two new states i and f. For each initial state i' (resp. final state f') of $\mathcal{A}'_0$ add a fork transition $(i, \{i, i'\})$ (resp. a join transition $(\{f', f\}, f)$). For each $t \in T$ duplicate each sequential and fork transition of source i' taken in $\mathcal{A}'_t$ by replacing the source i' with i. Duplicate $t \in T$ in $\mathcal{A}_t$ by replacing the destination f' with f. For each sequential transition (i', a, f') add (i, a, f). The unique initial state of $\mathcal{A}$ is i, and its final states are i and f.

We now prove that the language of some b*-automaton $\mathcal{A} = (Q, A, E, I, F)$ verifying Condition 2 is series-parallel rational. We adapt the McNaughton-Yamada construction of a rational expression from an automaton (see [7, Section 4.2] for the case of b-automata). When $p, q \in Q$, $D, D' \subseteq E_{\text{fork}} \times E_{\text{join}}$, $(f, j) \in E_{\text{fork}} \times E_{\text{join}}$, denote by:

- $L_{p,q} = \{P \in SP(A) : p \xrightarrow[\mathcal{A}]{P} q\}$;
- $L^{D,D'}_{p,q}$ is the set of labels of b*-paths from p to q that can use only pairs of fork and join transitions from D at the upper level and from D' at a sequential level;
- $L(D', f, j)$ is the set of label of b*-paths rooted by (f, j) that can use only pairs of fork and join transitions from D' at a sequential level.

A rational expression for $L^{\emptyset,D'}_{p,q}$ is found as for automata on words since no fork and join transitions are allowed. Otherwise, $D \neq \emptyset$ and since $\mathcal{A}$ verifies Condition 2:

$$\begin{aligned}
L_{p,q} =& L^{E_{\text{fork}} \times E_{\text{join}}, E_{\text{fork}} \times E_{\text{join}}}_{p,q} \\
L^{D,D'}_{p,q} =& \bigcup_{(f,j) \in D \cap D'} \left(L^{D \setminus \{(f,j)\}, D'}_{p,q} \cup L^{D \setminus \{(f,j)\}, D'}_{p, \pi_1(f)} (L(D', f, j) L^{D \setminus \{(f,j)\}, D'}_{\pi_2(j), \pi_1(f)})^* L^{D \setminus \{(f,j)\}, D'}_{\pi_1(f), q}\right) \\
& \bigcup_{\substack{(f,j) \in D \\ \pi_1(f) = p \\ \pi_2(j) = q}} \left(L^{D \setminus \{(f,j)\}, D'}_{p,q} \cup L(D', f, j)\right) \\
L(D', f, j) =& \bigcup_{\sigma \in S_k} \mathop{\|}_{i \in [k]} L^{E_{\text{fork}} \times E_{\text{join}}, D' \setminus \{(f,j)\}}_{r_i, s_{\sigma_i}}
\end{aligned}$$

where f and j have the form $f = (r, \{r_1, \ldots, r_k\})$ and $j = (\{s_1, \ldots, s_k\}, s)$ in the last equality. The above equalities form a system of equations where the unknowns are the $L^{D,D'}_{p,q}$, $D \neq \emptyset$, and the $L^{\emptyset,D'}_{p,q}$ are the constants. First observe that when $D \neq \emptyset$, $L^{D,\emptyset}_{p,q}$ depends only of the $L^{D',\emptyset}_{p',q'}$, $D' \subset D$ or $D' = E_{\text{fork}} \times E_{\text{join}}$. The only operations involved in the equalities for the $L^{D,\emptyset}_{p,q}$ are $\cup$ and $\|$. The system of equations is solved as usual using substitutions and $^{\circledast}$ is used to resolve circular substitutions. ◀

A similar result holds for p-automata [5] $\mathcal{A}$. In $\mathcal{A}$, define $\preceq$ as the smallest preorder on states such that $E_{\text{seq}}(q, a) \preceq q$, $E_{\text{par}}(q, r, s) \preceq q$, and if $E_{\text{par}}(q, r, s) \neq \bot$ then $r, s \preceq q$. Set also $p \prec q$ if and only if $p \preceq q$ and $q \npreceq p$. Say that $\mathcal{A}$ is *well-nested* if each state q verifies *exactly* one of the following properties:

1. $r, s \prec q$ for all states r, s with $E_{\text{par}}(q, r, s) \neq \bot$;
2. $q \in F$, $E_{\text{seq}}(q, a) = \bot$ for all a, and if $E_{\text{par}}(q, r, s) \neq \bot$ then $E_{\text{par}}(q, r, s) = \top$, $s = q$ and $r \prec q$.

▶ **Theorem 12** ([5]). *A language L of $SP(A)$ is series-parallel rational if and only if there is a well-nested p-automaton $\mathcal{A}$ with $L = L(\mathcal{A})$.*

We now show that a very similar characterisation also exists for b*-automata to have them correspond to series-parallel rational languages. For every b*-automaton there is a b*-automaton with the same language and with all its fork and join transitions of arity 2. We assume here that all fork and join transitions of b*-automata are of arity 2. In a b*-automaton $\mathcal{A}$, define $\preceq_f$ as the smallest preorder on states such that

1. if $p \xrightarrow[\mathcal{A}]{} q$ then $q \preceq_f p$;
2. if $(p, \{p_1, p_2\}) \in E_{\text{fork}}$ then $p_1, p_2 \preceq_f p$;
3. if $(\{q_1, q_2\}, q) \in E_{\text{join}}$ then $q_1, q_2 \preceq_f q$.

Set $p \prec_f q$ if and only if $p \preceq_f q$ and $q \not\preceq_f p$. Define $\preceq_j$ and $\prec_j$ similarly, by replacing Item 1 above by: if $p \underset{\mathcal{A}}{\rightarrow} q$ then $p \preceq_f q$. Observe that if there is a b*-path $p \underset{\mathcal{A}}{\rightarrow} q$ then $r \preceq_f p$ and $r \preceq_j q$ for all states r used in the b*-path.

A fork transition $(p, \{p_1, p_2\})$ (resp. join transition $(\{p_1, p_2\}, p)$) is *normal* if $p_1, p_2 \neq p$. It is *recursive* otherwise. A state p is *normal* when all the fork transitions $(p, \{p_1, p_2\})$ and join transitions $(\{p_1, p_2\}, p)$ verify $p_1, p_2 \prec_f p$ and $p_1, p_2 \prec_j p$. It is *fork recursive* when all the conditions below are verified:

- there is some recursive fork transition $(p, \{p, p'\})$, and in this case $p' \prec_f p$ and $p' \prec_j p$ for all such transitions;
- for all normal fork transition $(p, \{p_1, p_2\})$ then $p_1, p_2 \prec_f p$ and $p_1, p_2 \prec_j p$;
- for all normal fork transition $(p, \{p_1, p_2\})$ and join transition $(\{q_1, q_2\}, q)$, when $p_1 \underset{\mathcal{A}}{\rightarrow} q_1$ and $p_2 \underset{\mathcal{A}}{\rightarrow} q_2$ then $q \prec_f p$;
- for all sequential transition (p, a, q) then $q \prec_f p$;
- there is no transition of the form (q, a, p) or $(\{p_1, p_2\}, p)$.

It is *join recursive* when all the conditions below are verified:

- there is some recursive join transition $(\{p, p'\}, p)$, and in this case $p' \prec_f p$ and $p' \prec_j p$ for all such transitions;
- for all normal join transition $(\{p_1, p_2\}, p)$ then $p_1, p_2 \prec_f p$ and $p_1, p_2 \prec_j p$;
- for all normal fork transition $(q, \{q_1, q_2\})$ and join transition $(\{p_1, p_2\}, p)$, when $q_1 \underset{\mathcal{A}}{\rightarrow} p_1$ and $q_2 \underset{\mathcal{A}}{\rightarrow} p_2$ then $q \prec_j p$;
- for all sequential transition (q, a, p) then $q \prec_j p$;
- there is no transition of the form (p, a, q) or $(p, \{p_1, p_2\})$.

Then $\mathcal{A}$ is *well-nested* if each state as a unique classification as normal, fork recursive or join recursive, and, when $x = (p, \{p_1, p_2\})$ and $y = (\{q_1, q_2\}, q)$ are a fork and a join transition and $p_i \underset{\mathcal{A}}{\rightarrow} q_i$, $i \in [2]$, then x, y are both recursive, or both normal.

▶ **Proposition 13.** *For every well-nested b*-automaton $\mathcal{A}$ there is a behaved and well-nested b*-automaton $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$, and the initial and final states of $\mathcal{B}$ are normal.*

Proof. It suffices to check that the construction of the proof of Proposition 7 preserves well-nestedness, and that in the copy $\mathcal{A}_0$ all states are normal. ◀

▶ **Theorem 14.** *A language L of $SP(A)$ is series-parallel rational if and only if there is a well-nested b*-automaton $\mathcal{A}$ with $L = L(\mathcal{A})$.*

Proof. The implication from left to right is by induction over a series-parallel rational expression e for L. The steps are the same as in the proof of Theorem 11 (it suffices to check that the constructions preserve well-nestedness).

For the implication from right to left we show that a well-nested b*-automaton $\mathcal{A}$ verifies Condition 2, and the conclusion follows by Theorem 11. Assume by contradiction that $\mathcal{A}$ does not verify Condition 2, ie. it has a b*-path p rooted by some pair (f, j) of a pair and a join transition such that p uses (f, j) at a sequential level. Let $f = (r, \{r_1, r_2\})$ and $j = (\{s_1, s_2\}, s)$. Take a term t which is a representative of p: there are some $t'' \preceq t' \preceq t$ with t'' rooted by (f, j) and t' of the form $t' = t'_1 \cdot t'_2$ with t'_1, t'_2 both non-trivial. Consider t as a tree. In this tree, consider the path α from the root node n of t to the root node n' of the sub-tree t''. This path goes through the root node n'' of the sub-tree t', that we may consider the first one along α such that $t' = t'_1 \cdot t'_2$ for some non-trivial t'_1, t'_2. The term t has the form $(f, j)(t_1, t_2)$, with t_i a b*-path $r_i \underset{\mathcal{A}}{\rightarrow} s_i$, $i \in [2]$. Either the root node of t_1 or of t_2 belongs to

α, say wlog. t_1. If $r_1 \prec_f r$ or $s_1 \prec_j s$ we get a contradiction since $x \preceq_f r_1$ and $x \preceq_j s_1$ for all states x used in t_1, and thus for $x = r$. Thus $r_1 = r$ and $s_1 = s$. This reasoning is true for all nodes of α between n and n'. Thus t'_1 and t'_2 are respectively non-trivial b*-paths $r \underset{\mathcal{A}}{\to} x$ and $x \underset{\mathcal{A}}{\to} s$ for some x, that can not use recursive fork or join transitions at the upper level, since there is no fork recursive state y and state y' such that $y' \underset{\mathcal{A}}{\to} y$, and there is no join recursive state y and state y' such that $y \underset{\mathcal{A}}{\to} y'$. Thus $x \prec_f r$ and $x \prec_j s$. For all states y used in t'_1 it holds $y \preceq_f r$, and for all states y used in t'_2 we have $y \preceq_j s$. Assume first t'' is a sub-term of t'_2. Then we have $z \preceq_f x$ for all states z appearing in t'_2, in particular $r \preceq_f x$, which is a contradiction. Thus t'' is a sub-term of t'_1. We have $z \preceq_j x$ for all states z appearing in t'_1, in particular $s \preceq_j x$, which is again a contradiction. ◀

▶ **Example 15.** Figure 3 represents a well-nested b*-automaton $\mathcal{A}$ obtained by induction on the series-parallel rational expression $e = ((a \parallel b)^*)^{\circledast}$ following the steps of the proof of Theorem 14. Note that it is misbehaved since for example 1 is initial, $1 \xrightarrow[\mathcal{A}]{a\|b} 8$ and because of the join transition $(\{2', 8\}, 8)$. The only fork recursive state is 1, the only join recursive state is 8, and all other states are normal.

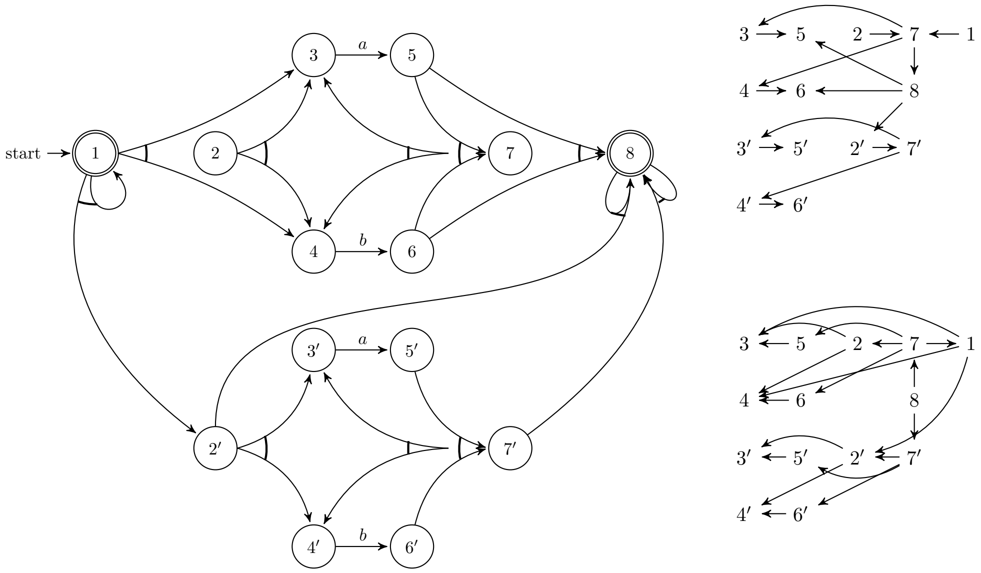


Figure 3 On the left, a well-nested b*-automaton $\mathcal{A}$ with $L(\mathcal{A}) = ((a \parallel b)^*)^{\circledast}$. It has 6 fork transitions $f_1 = (1, \{3, 4\})$, $f_2 = (1, \{1, 2'\})$, $f_3 = (2, \{3, 4\})$, $f_4 = (7, \{3, 4\})$, $f_5 = (2', \{3', 4'\})$, $f_6 = (7', \{3', 4'\})$, 4 join transitions $j_1 = (\{5, 6\}, 8)$, $j_2 = (\{7', 8\}, 8)$, $j_3 = (\{5, 6\}, 7)$, $j_4 = (\{2', 8\}, 8)$, $j_5 = (\{5', 6'\}, 7')$. On the right top, a diagram of the preorder $\preceq_f$ over the states of $\mathcal{A}$. On the right bottom, a diagram of $\preceq_j$.

6 Conclusion

We have compared branching versus p-automata. When they are defined in the most general manner, they have the same expressive power which corresponds to that of context-free grammars, or equivalently, to series-parallel rational expressions with additional $L \circ_\xi L'$ and $L^{*\xi}$ operations that enable respectively substitutions and iterated substitutions. As consequences of the equivalence between b*-automata and context-free grammars questions such as "Is a b*-regular language b-regular, or series-parallel rational?" are undecidable.

We also gave characterizations on branching automata to have them exactly as expressive as series-parallel rational expressions, answering of question of [9]; a similar restriction (*well-nestedness*) was already known for p-automata [5]. All the results are effective. *Series-rational* languages are defined similarly to series-parallel languages without the ability to iterate parallelism (ie. series-rational expressions are series-parallel expressions without $L^{\circledast}$). They have been investigated in [9] for branching automata and in [4] for p-automata. Corresponding automata have the fork-acyclicity property, ie. they can not use a transition that splits an execution flow into several parallel flows $f_1, \ldots, f_n$ into the f_i's again. The following diagram sums-up those results:

b*-rational b*-regular p-regular context-free	$\supsetneq$	b-rational b-regular	$\supsetneq$	series-parallel rational b-regular ∩ Condition 2 b*-regular ∩ well-nested p-regular ∩ well-nested	$\supsetneq$	series-rational b-regular ∩ fork-acyclic p-regular ∩ fork-acyclic

References

1 Nicolas Bedon. Logic and branching automata. *Logical Methods in Computer Sciences*, 11(4:2):1–38, October 2015.

2 Jay L. Gischer. The equational theory of pomsets. *Theoret. Comput. Sci.*, 61(2):199–224, 1988. doi:10.1016/0304-3975(88)90124-7.

3 Jan Grabowski. On partial languages. *Fundam. Inform.*, 4(1):427–498, 1981.

4 Tobias Kappé, Paul Brunet, Bas Luttik, Alexandra Silva, and Fabio Zanasi. Brzozowski goes concurrent - A Kleene theorem for pomset languages. In *Proc. CONCUR 2017: 28th International Conference on Concurrency Theory*, volume 28, pages 21:1–21:15, January 2017.

5 Tobias Kappé, Paul Brunet, Bas Luttik, Alexandra Silva, and Fabio Zanasi. On series-parallel pomset languages: Rationality, context-freeness and automata. *Journal of Logical and Algebraic Methods in Programming*, 103, December 2018. doi:10.1016/j.jlamp.2018.12.001.

6 Stephen C. Kleene. Representation of events in nerve nets and finite automata. In Shannon and McCarthy, editors, *Automata studies*, pages 3–41, Princeton, New Jersey, 1956. Princeton University Press.

7 Kamal Lodaya and Pascal Weil. A Kleene iteration for parallelism. In V. Arvind and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1530 of *Lect. Notes in Comput. Sci.*, pages 355–367. Springer-Verlag, 1998.

8 Kamal Lodaya and Pascal Weil. Series-parallel posets: algebra, automata and languages. In M. Morvan, Ch. Meinel, and D. Krob, editors, *STACS'98*, volume 1373 of *Lect. Notes in Comput. Sci.*, pages 555–565. Springer-Verlag, 1998.

9 Kamal Lodaya and Pascal Weil. Series-parallel languages and the bounded-width property. *Theoret. Comput. Sci.*, 237(1–2):347–380, 2000.

10 Kamal Lodaya and Pascal Weil. Rationality in algebras with a series operation. *Inform. Comput.*, 171:269–293, 2001.

11 Jacobo Valdes. Parsing flowcharts and series-parallel graphs. Technical Report STAN-CS-78-682, Computer science departement of the Stanford University, Standford, Ca., 1978.

12 Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. *SIAM J. Comput.*, 11:298–313, 1982. doi:10.1137/0211023.

13 Józef Winkowski. An algebraic approach to concurrence. In *MFCS'79*, volume 74 of *Lect. Notes in Comput. Sci.*, pages 523–532. Springer Verlag, 1979.

History Determinism vs. Good for Gameness in Quantitative Automata

Udi Boker
Reichman University, Herzliya, Israel

Karoliina Lehtinen
CNRS, Marseille-Aix Université, Université de Toulon, LIS, Marseille, France

Abstract

Automata models between determinism and nondeterminism/alternations can retain some of the algorithmic properties of deterministic automata while enjoying some of the expressiveness and succinctness of nondeterminism. We study three closely related such models – history determinism, good for gameness and determinisability by pruning – on quantitative automata.

While in the Boolean setting, history determinism and good for gameness coincide, we show that this is no longer the case in the quantitative setting: good for gameness is broader than history determinism, and coincides with a relaxed version of it, defined with respect to thresholds. We further identify criteria in which history determinism, which is generally broader than determinisability by pruning, coincides with it, which we then apply to typical quantitative automata types.

As a key application of good for games and history deterministic automata is synthesis, we clarify the relationship between the two notions and various quantitative synthesis problems. We show that good-for-games automata are central for "global" (classical) synthesis, while "local" (good-enough) synthesis reduces to deciding whether a nondeterministic automaton is history deterministic.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Good for games, history determinism, alternation, quantitative automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.38

Acknowledgements We thank Jan Otop for discussing Borel definability in quantitative automata.

1 Introduction

Boolean automata recognise languages of finite or infinite words, often used in verification to describe system behaviours. In contrast, quantitative automata define functions from words to values, and can describe system properties such as energy usage, battery-life or costs. Like Boolean automata, quantitative automata can have nondeterministic choices (disjunctions) and universal choices (conjunctions), which make them more powerful than deterministic models. Alternating automata combine both nondeterministic and universal choices.

However, not all nondeterminism is born equal. Generally, nondeterminism increases the expressiveness and succinctness of an automata model, but at the cost of also increasing the complexity of algorithmic problems on it, sometimes even rendering them undecidable. However, restricted forms of nondeterministic and even alternating automata can enjoy some of the good algorithmic properties of deterministic automata while also gaining in expressiveness and succinctness.

We focus on three closely related restrictions on nondeterminism and alternations, relevant to the synthesis problem. *History determinism* [11] postulates that the choices in the automaton – whether they be nondeterministic or universal – should not depend on the future of the input word. That is, one should be able to construct runs letter by letter while reading the input word, so that the resulting run is as good as one constructed with the knowledge of the full word. The notion of *good for games* automata comes from solving two-player games without determinisation [14]. It postulates that the composition of such an

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 38; pp. 38:1–38:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

automaton $\mathcal{A}$ with games whose payoff function is described by $\mathcal{A}$ should be an equivalent game – that is, one with the same winner in the Boolean setting, or the same value in the quantitative setting. Finally, an automaton is *determinisable by pruning* if it embeds an equivalent deterministic automaton and, at least in the nondeterministic case, this notion can be seen as a (stronger) "semi-syntactic" version of history determinism.

The three notions are well studied in the Boolean setting. There, history determinism and good for gameness coincide, and are broader than determinisability by pruning in general, but coincide with it for some automata types [8].

We generalize these notions to the quantitative setting and study the relations between them. Some versions of these notions already appear in the literature with respect to quantitative automata, as we elaborate on in the related-work paragraph, however not in a systematic and consistent way, and without analysis of the relations between them.

We start with general results concerning arbitrary quantitative automata and then provide a more specific analysis of the following most common types of quantitative automata: $\mathsf{Sum}, \mathsf{Avg}, \mathsf{Inf}, \mathsf{Sup}$, discounted sum ($\mathsf{DSum}$), $\mathsf{LimInf}, \mathsf{LimSup}, \mathsf{LimInfAvg}$ and $\mathsf{LimSupAvg}$.

Surprisingly, it turns out that good for gameness and history determinism no longer coincide in the quantitative setting. The surprise comes from the fact that the two names are used interchangeably in the Boolean setting and are already starting to mix in the quantitative setting. (In the Boolean setting, even the seminal paper of Henzinger and Piterman [14], which named the "good for games" notion, defined history deterministic automata and showed that they are indeed good for games, while the other direction was only shown later [8]. In the quantitative setting, [16, 17, 18] speak of good for games quantitative automata, although their definition is closer to history determinism.)

We first observe that in the quantitative setting, the three notions need sub-notions, relating to whether one considers automata/games equivalence with respect to values or thresholds. (See Section 3 for the exact definitions.)

We then show that while good for gameness coincides with threshold good for gameness, history determinism is stricter than threshold history determinism, and only the latter, under some assumptions, is equivalent to good for gameness. (See Figure 2 for a detailed scheme of the relations.) The assumption for the equivalence of threshold history determinism and good for gameness is that the "letter game" played on the quantitative automaton (which defines whether or not it is history deterministic) is determined. We show that this is guaranteed for quantitative automata whose threshold versions define Borel sets.

Determinizability by pruning, which has an appealing structural definition, is generally stricter than history determinism for nondeterministic automata, already in the Boolean setting, while equivalent to it for some automata types. We observe that the two notions are incomparable for alternating automata, already in the Boolean setting (see Figure 5). We then analyse general properties of value functions that guarantee the equivalence of determinizability by pruning and history determinism for all nondeterministic quantitative automata whose value function has these properties. We apply these results to specific automata types. Specifically, we show the equivalence for $\mathsf{Sum}, \mathsf{Avg}, \mathsf{Inf}$ and Sup automata on finite words and DSum automata on finite and infinite words.

Finally, we discuss how the different notions are relevant for different quantitative synthesis problems. In quantitative synthesis [9, 2], the specification is a function f that maps sequences of input-output pairs onto values. The goal of the system is to respond to input letters by producing output letters while maximising the value of the resulting input-output sequence. Given a function f, one can ask several questions: (i) what is the best value a system can guarantee over all inputs [4]? (ii) can it guarantee at least a threshold value? (iii) can it guarantee for each input sequence I the best value that an input-output sequence including I has [12]? (iv) can it achieve a threshold value t for all inputs that appear in an input-output sequence with value at least t? In a nutshell, we show that on one hand, (threshold) good for games alternating quantitative automata can be used to solve (i) and (ii) via a product construction similar to the one used for deterministic automata [4]; and on the other hand, (iii) and (iv) for (threshold) history deterministic nondeterministic automata are linearly inter-reducible with deciding the (threshold) history-determinism of an automaton.

Related work. Thomas Colcombet's original definition of history determinism [11] also considered non-Boolean automata, namely cost automata. While the restriction of his definition to ω-regular automata coincides with the original definition in [14] of good for games automata [8], in the quantitative setting his definition is different from what we provide here. His notion can be viewed as 'approximated history-deterministic with respect to a threshold' as it asks for an approximation ratio that describes the difference between the value achieved by a strategy without the knowledge of the full input word and the actual value of the word. Another notion of approximative history determinism appears in [16, 17, 18] under the name of r-GFGness, where r is a bound on the difference of the two values. Zero-regret determinizability [3, 17] on the other hand lies somewhere between approximative determinizability by pruning and approximative history determinism. It requires an automaton to be approximatively equivalent to a deterministic automaton obtained by taking the product of the input automaton with a finite memory, with both the size of the memory and permitted regret as parameters. When both are set to zero, we have determinizability by pruning.

Observe that we use the term "quantitative automata" rather than "weighted automata". The latter usually relates to the algebraic definition, whereby the value of a nondeterministic automaton on a word is the semiring sum (or valuation-monoid sum) of its accepting runs' values. It is generally not defined for alternating automata. The former defines the value of a nondeterministic or alternating automaton on a word to be the supremum/infimum of its runs' values, having the "choice" and "obligation" interpretation of nondeterminism and universality, respectively. (See [5] for a discussion on the differences between the two.) Since history determinism naturally relates to "choice" and "obligation" in nondeterministic and alternating automata, quantitative automata better fit the present work.

Due to space constraints, some of the proofs appear in the appendix.

2 Preliminaries

Words. An *alphabet* Σ is a finite nonempty set of letters. A finite (resp. infinite) *word* $u = u_0 \dots u_k \in \Sigma^*$ (resp. $w = w_0 w_1 \dots \in \Sigma^\omega$) is a finite (resp. infinite) sequence of letters from Σ. We write Σ^∞ for $\Sigma^* \cup \Sigma^\omega$. We use $[i..j]$ to denote a set $\{i, \dots, j\}$ of integers, $[i]$ for $[i..i]$, $[..j]$ for $[0..j]$, and $[i..]$ for integers equal to or larger than i. We write $w[i..j], w[..j]$, and $w[i..]$ for the infix $w_i \dots w_j$, prefix $w_0 \dots w_j$, and suffix $w_i \dots$ of w. A *language* is a set of words, and the empty word is written ε.

Games. We consider turn-based zero-sum games between Adam and Eve, with Σ-labelled transitions. A play generates a word, and each word has a value, given by the game's payoff function. Eve tries to maximise the value of the play, while Adam tries to minimise it. Formally, for a payoff function f, an f *game* is defined on an *arena* $(V, E, V_E, V_A, L : E \to \Sigma \cup \{\varepsilon\})$, which consists of a (potentially infinite) set of positions V, partitioned into Eve's positions V_E and Adam's positions V_A, and a set of edges $E \subseteq V \times V$, labelled by L with letters from $\Sigma \cup \{\varepsilon\}$. In infinite-duration games every position has at least one outgoing edge. A play is a maximal path over V; its non-ε labels induce a word $w \in \Sigma^*$ or Σ^ω. The payoff of a play is the value of this word, given by the payoff function f.

Strategies for Adam and Eve map partial plays ending in a position v in V_A and V_E respectively to outgoing edges from v. A play or partial play π agrees with a strategy s_P, written $\pi \in s_P$, for a player $P \in \{A, E\}$, if whenever its prefix p ends in a position in V_P, the next edge is $s_P(p)$. The value $f(s_E)$ of a strategy s_E for Eve is $\inf_{\pi \in s_E} f(\pi)$ and the value $f(s_A)$ of a strategy s_A for Adam is $\sup_{\pi \in s_A} f(\pi)$. Let S_E and S_A be the sets of strategies for Eve and Adam respectively. If $\sup_{s \in S_E} f(s)$ (the best Eve can do) coincides with $\inf_{s \in S_A} f(s)$ (the best Adam can do), we say that G is determined and $\sup_{s \in S_E} f(s) = \inf_{s \in S_A} f(s)$ is called the value of G. Eve wins the t-threshold game on G, for some $t \in \mathbb{R}$, if the value of G is at least t; else Adam wins. Eve wins the strict t-threshold game on G if the value of G is greater than t. Two games are equivalent in this context if they have the same value. We restrict the scope of this article to determined games.

Quantitative Automata. An *alternating quantitative automaton* on words is a tuple $\mathcal{A} = (\Sigma, Q, \iota, \delta)$, where: Σ is an alphabet; Q is a finite nonempty set of states; $\iota \in Q$ is an initial state; and $\delta \colon Q \times \Sigma \to \mathsf{B}^+(\mathbb{Q} \times Q)$ is a transition function, where $\mathsf{B}^+(\mathbb{Q} \times Q)$ is the set of positive Boolean formulas (*transition conditions*) over weight-state pairs.

A *transition* is a tuple $(q, a, x, q') \in Q \times \Sigma \times \mathbb{Q} \times Q$, sometimes also written $q \xrightarrow{a:x} q'$. (Note that there might be several transitions with different weights over the same letter between the same pair of states[1].) We write $\gamma(t) = x$ for the weight of a transition $t = (q, a, x, q')$.

An automaton $\mathcal{A}$ is nondeterministic (resp. universal) if all its transition conditions are disjunctions (resp. conjunctions), and it is deterministic if all its transition conditions are just weight-state pairs. We represent the transition function of nondeterministic and universal automata as $\delta \colon Q \times \Sigma \to 2^{(\mathbb{Q} \times Q)}$, and of a deterministic automaton as $\delta \colon Q \times \Sigma \to \mathbb{Q} \times Q$.

We require that the automaton $\mathcal{A}$ is *total*, namely that for every state $q \in Q$ and letter $a \in \Sigma$, there is at least one state q' and a transition $q \xrightarrow{a:x} q'$. For a state $q \in Q$, we denote by $\mathcal{A}^q$ the automaton that is derived from $\mathcal{A}$ by setting its initial state ι to q.

A run of the automaton on a word w is intuitively a play between Adam and Eve. It starts in the initial state ι, and in each round, when the automaton is in state q and the next letter of w is a, Eve resolves the nondeterminism (disjunctions) of the transition condition $\delta(q, a)$ and Adam resolves its universality (conjunctions), yielding a transition $q \xrightarrow{a:x} q'$. The output of a play is thus a sequence $\pi = t_0 t_1 t_2 \ldots$ of transitions. As each transition t_i carries a weight $\gamma(t_i) \in \mathbb{Q}$, the sequence π provides a weight sequence $\gamma(\pi) = \gamma(t_0)\gamma(t_1)\gamma(t_2)\ldots$. More formally, given the automaton $\mathcal{A} = (\Sigma, Q, \iota, \delta)$ and a word $w \in \Sigma^*$ (resp. $w \in \Sigma^\omega$), we define the arena $G(\mathcal{A}, w)$ with positions $Q \times \Sigma^* \times \mathsf{B}^+(\mathbb{Q} \times Q)$ (resp. $Q \times \Sigma^\omega \times \mathsf{B}^+(\mathbb{Q} \times Q)$), the initial position $(\iota, w, \delta(\iota, w[0]))$, ε-labelled edges from (q, u, b) to (q, u, b') when b' is an immediate subformula of b, and x-labelled edges from $(q, u, (x, q'))$ to $(q', u[1..], \delta(q', u[1]))$. Conjunctive positions belong to Adam while disjunctive ones belong to Eve.

A Val automaton (for example a Sum automaton) is one equipped with a *value function* $\mathsf{Val} : \mathbb{Q}^* \to \mathbb{R}$ or $\mathsf{Val} : \mathbb{Q}^\omega \to \mathbb{R}$. The corresponding game is the Val game on the arena $G(\mathcal{A}, w)$: each run π (play in $G(\mathcal{A}, w)$) has a real value $\mathsf{Val}(\gamma(\pi))$, which we abbreviate by

[1] This extra flexibility of allowing for "parallel" transitions with different weights is often omitted (e.g., in [10]) since it is redundant for some value functions while important for others.

$\mathsf{Val}(\pi)$. When this game is determined, we say that the value of $\mathcal{A}(w)$ is the value of $G(\mathcal{A}, w)$, and if $G(\mathcal{A}, w)$ is determined for all $w \in \Sigma^\omega$, we say that $\mathcal{A}$ realizes a function from words to real numbers. We restrict the scope of this article to automata realizing functions.

Two automata $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent*, denoted by $\mathcal{A} \equiv \mathcal{A}'$, if they realize the same function. For a threshold $t \in \mathbb{R}$ and a Val automaton $\mathcal{A}$, we also speak of a corresponding Boolean t-threshold Val automaton $\mathcal{A}'$ that accepts the words w such that $\mathcal{A}(w) \geq t$.

Observe that when $\mathcal{A}$ is nondeterministic, a run of $\mathcal{A}$ on a word w is a sequence π of transitions, and the value of $\mathcal{A}$ on w is the supremum of $\mathsf{Val}(\pi)$ over all these runs π.

Value functions. We list here the most common value functions for quantitative automata on finite/infinite words, defined over sequences of rational weights[2]:

- For finite sequences $v = v_0 v_1 \dots v_{n-1}$:
  - $\mathsf{Sum}(v) = \sum_{i=0}^{n-1} v_i$
  - $\mathsf{Avg}(v) = \frac{1}{n} \sum_{i=0}^{n-1} v_i$
- For finite and infinite sequences $v = v_0 v_1 \dots$:
  - $\mathsf{Inf}(v) = \inf\{v_n \mid n \geq 0\}$
  - $\mathsf{Sup}(v) = \sup\{v_n \mid n \geq 0\}$
  - For a discount factor $\lambda \in \mathbb{Q} \cap (0,1)$, $\mathsf{DSum}(v) = \sum_{i \geq 0} \lambda^i v_i$
- For infinite sequences $v = v_0 v_1 \dots$:
  - $\mathsf{LimInf}(v) = \lim_{n \to \infty} \inf\{v_i \mid i \geq n\}$
  - $\mathsf{LimSup}(v) = \lim_{n \to \infty} \sup\{v_i \mid i \geq n\}$
  - $\mathsf{LimInfAvg}(v) \ = \mathsf{LimInf}(\mathsf{Avg}(v_0), \mathsf{Avg}(v_0, v_1), \mathsf{Avg}(v_0, v_1, v_2), \dots)$
  - $\mathsf{LimSupAvg}(v) = \mathsf{LimSup}(\mathsf{Avg}(v_0), \mathsf{Avg}(v_0, v_1), \mathsf{Avg}(v_0, v_1, v_2), \dots)$

($\mathsf{LimInfAvg}$ and $\mathsf{LimSupAvg}$ are also called $\underline{\mathsf{MeanPayoff}}$ and $\overline{\mathsf{MeanPayoff}}$.)

Products. The synchronized product of a Σ-labelled game G and an automaton $\mathcal{A}$ over alphabet Σ is (like in the Boolean setting, see e.g., [8, Definition 1]) a game $G \times \mathcal{A}$ obtained by taking the product of the positions of G and the states and transition conditions of $\mathcal{A}$, and their corresponding transitions. Positions with nondeterminism are of Eve and positions with universality are of Adam. Transitions carry their weight from the corresponding transition in $\mathcal{A}$. The payoff function of the game is the value function of $\mathcal{A}$.

3 Good For Gameness, History Determinism, and Determinizability By Pruning

In the Boolean setting, "good for gameness" and "history determinism", stemming from different concepts, coincide both for nondeterministic and alternating automata [8].

We generalize these definitions to quantitative automata, observing that under this setting they need some sub-variants, relating to whether one considers automata/games equivalence with respect to all values or some threshold[3]. As shown in Section 4, the two main notions, as well as some of their variants, are generally not equivalent in the quantitative setting.

[2] There are also value functions that are more naturally defined over sequences of tuples of rational numbers, for example discounted-summation with multiple discount factors [6].

[3] For a threshold $t \in \mathbb{R}$, we provide the definitions with respect to a non-strict inequality $\geq t$. Using strict inequality $> t$ instead, yields the same relations between the notions, as stated in Theorem 4.

The notion of determinizability by pruning, which has an appealing structural definition, is generally stricter than good for gameness and history determinism in the setting of nondeterministic automata, already in the Boolean setting, yet we show that for some value functions it is equivalent to history determinism. For alternating automata, we show that it is incomparable with history determinism and good for gameness.

▶ **Definition 1** (Good for gameness). *An automaton $\mathcal{A}$ realizing a function $f : \Sigma^* \to \mathbb{R}$ or $f : \Sigma^\omega \to \mathbb{R}$ is*

- good for games *if for every determined*[4] *game G with a Σ-labelled arena and payoff function f, we have that G and $G \times \mathcal{A}$ have the same value;*
- good for t-threshold games*, for some $t \in \mathbb{R}$, if for every determined game G with a Σ-labelled arena and payoff function f, Eve wins the t-threshold game on G if and only if she wins the t-threshold game on $G \times \mathcal{A}$;*
- good for threshold games *if it is good for t-threshold games for all $t \in \mathbb{R}$.*

An automaton is history deterministic if there are strategies to resolve its nondeterminism and universality, such that for every word, the (threshold) value remains the same.

▶ **Definition 2** (History-determinism). *Consider an alternating* Val *automaton $\mathcal{A} = (\Sigma, Q, \iota, \delta)$ realizing a function $f : \Sigma^* \to \mathbb{R}$ or $f : \Sigma^\omega \to \mathbb{R}$. Formally,* history determinism *is defined via* letter games*, detailed below.*

- *$\mathcal{A}$ is* history deterministic *if Eve and Adam win their letter games.*
- *$\mathcal{A}$ is* t-threshold history deterministic*, for some $t \in \mathbb{R}$, if Eve and Adam win their t-threshold letter games.*
- *$\mathcal{A}$ is* threshold history deterministic *if it is t-threshold history deterministic for all $t \in \mathbb{R}$.*

Eve's (Adam's) letter games are the following win-lose games, in which Adam (Eve) chooses the next letter and Eve and Adam resolve the nondeterminism and universality, aiming to construct a run whose value is (threshold) equivalent to the generated word's value.

Eve's letter game: *A configuration is a pair (σ, b) where $b \in \mathsf{B}^+(Q)$ is a transition condition and $\sigma \in \Sigma \cup \{\varepsilon\}$ is a letter. (We abuse ε to also be an empty letter.) A play begins in $(\sigma_0, b_0) = (\varepsilon, \iota)$ and consists of an infinite sequence of configurations $(\sigma_0, b_0)(\sigma_1, b_1)\ldots$. In a configuration (σ_i, b_i), the play proceeds to the next configuration (σ_{i+1}, b_{i+1}) as follows.*

- *If b_i is a state of Q, Adam picks a letter a from Σ, and $(\sigma_{i+1}, b_{i+1}) = (a, \delta(b_i, a))$.*
- *If b_i is a conjunction $b_i = b' \wedge b''$, Adam chooses between (ε, b') and (ε, b'').*
- *If b_i is a disjunction $b_i = b' \vee b''$, Eve chooses between (ε, b') and (ε, b'').*

In the limit, a play consists of an infinite word w that is derived from the concatenation of $\sigma_0, \sigma_1, \ldots$, as well as an infinite sequence $b_0, b_1, \ldots$ of transition conditions, which yields an infinite sequence $\pi = t_0, t_1, \ldots$ of transitions.

If $\mathcal{A}$ is over infinite words, Eve wins a play in the letter-game if $\mathsf{Val}(\pi) \geq \mathcal{A}(w)$. In the t-threshold letter game, Eve wins if $\mathcal{A}(w) \geq t \implies \mathsf{Val}(\pi) \geq t$. For $\mathcal{A}$ over finite words, Eve wins if $\mathsf{Val}(\pi[0..i]) \geq \mathcal{A}(w[0..i])$ or $\mathcal{A}(w[0..i]) \geq t \implies \mathsf{Val}(\pi[0..i]) \geq t$ for all i.

Adam's letter game *is similar to Eve's game, except that Eve chooses the letters instead of Adam, and Adam wins a play in his letter game if $\mathsf{Val}(\pi) \leq \mathcal{A}(w)$ and in his t-threshold letter game if $\mathcal{A}(w) < t \implies \mathsf{Val}(\pi) < \mathcal{A}(w)$. (The asymmetry of $<$ and $\leq$ is intended).*

Intuitively, an automaton is determinizable by pruning if it can be determinized to an equivalent (w.r.t. a threshold) deterministic automaton by removing some of its states and transitions. (In an alternating automaton, "removing transitions" means removing some disjunctive and conjunctive choices.)

[4] We discuss in the conclusion questions that arise if this restriction is lifted

▶ **Definition 3** (Determinizability by Pruning). *A* Val *automaton $\mathcal{A}$ is*

- determinizable by pruning *if there exists a deterministic* Val *automaton $\mathcal{A}'$ that is derived from $\mathcal{A}$ by pruning, such that $\mathcal{A}' \equiv \mathcal{A}$;*
- t-threshold determinizable by pruning *if there is a deterministic* Val *automaton $\mathcal{A}'$ that is derived from $\mathcal{A}$ by pruning, such that for every word w, we have $\mathcal{A}'(w) \geq t$ iff $\mathcal{A}(w) \geq t$;*
- threshold determinizable by pruning *if it is t-threshold determinizable by pruning $\forall t \in \mathbb{R}$.*

Observe that a Val-automaton can be good for games, history deterministic, or determinizable by pruning when interpreted on infinite words, but not when interpreted on finite words, as demonstrated in Figure 1 .

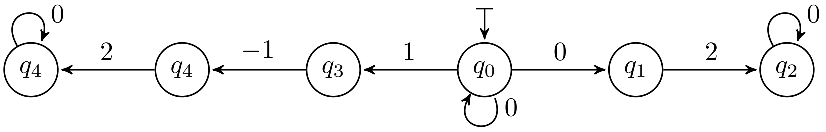


Figure 1 A nondeterministic DSum-automaton with discount factor $\frac{1}{2}$ over a unary alphabet that is determinizable by pruning, good for games, and history deterministic with respect to infinite words, but none of them with respect to finite words: For the single infinite word, the initial choice of going from q_0 to q_1 provides the optimal value of 1, making it all of the above. On finite words, on the other hand, it is not even threshold history deterministic (and by Theorem 4 neither of the rest), since in order to guarantee a value of at least 1, the first transition should be different for the word of length 1 and the word of length 2, going to q_3 for the former and to q_1 for the latter.

4 The Relations Between Notions

Having defined these notions, we now establish which inclusions hold in general, and which are conditional on characteristics of the value function, as summarised in Figure 2.

▶ **Theorem 4.** *(Threshold) good for gameness, (threshold) history determinism, and (threshold) determinizability by pruning of quantitative automata are related as described in Figure 2.*

Considering good for gameness, if an automaton $\mathcal{A}$ is good for all games then it is obviously good for all threshold games. The implication for the other direction stems from the fact that every concrete game G has a single value v. Then for G, it is enough to be good for v-threshold games, and for all automata, it is enough to be good for all threshold games.

▶ **Lemma 5.** *Good for Gameness $\iff$ Threshold Good for Gameness.*

For a t-threshold history deterministic automaton $\mathcal{A}$, Eve and Adam have strategies to win their t-letter games on $\mathcal{A}$. Thus, whenever Eve or Adam win some t-threshold game G, they can combine their two winning strategies to win $G \times \mathcal{A}$.

▶ **Lemma 6.** *Threshold History Determinism $\implies$ Threshold Good for Gameness*

For the other direction, we generalize proofs from [7, 8]: assuming that the automaton $\mathcal{A}$ is not threshold history deterministic we construct a threshold game G with respect to which $\mathcal{A}$ is not good for composition (namely, the product of G with $\mathcal{A}$ does not have the same winner as G). However, to build this game, we assume that either Adam wins Eve's letter game on $\mathcal{A}$ or Eve wins Adam's letter game on $\mathcal{A}$, that is, we assume that the letter games on $\mathcal{A}$ are determined. We later show that this determinacy requirement holds for all the specific value functions that we consider in the paper.

$$\begin{tikzcd}[column sep=small]
\text{Good For Gameness} \arrow[r, phantom, "=^1"] & \text{Threshold Good For Gameness} \arrow[r, phantom, "\cong^2"] \arrow[d, phantom, "\text{(for nondet.) } \rotatebox{90}{$\subsetneq$}^4"] & \text{Threshold History Determinism} \arrow[d, phantom, "\rotatebox{90}{$\subsetneq$}^3"] \\
& \text{Threshold Determinizability by Pruning} \arrow[r, phantom, "\neq^5"] \arrow[d, phantom, "\rotatebox{90}{$\subsetneq$}^3"] & \text{History Determinism} \arrow[d, phantom, "\rotatebox{90}{$\subsetneq$}^4 \text{ (for nondet.)}"] \\
& \text{Determinizability by Pruning} &
\end{tikzcd}$$

1. Always holds (Lemma 5).

2. The $\Longleftarrow$ implication always holds (Lemma 6); The $\Longrightarrow$ implication holds at least for all Val automata whose threshold letter games are determined (Lemma 7), e.g., for $\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}, \mathsf{DSum}$ and all functions on finite words (Theorem 9).

3. Strict containment for all non-trivial value functions with at least three values (Lemma 10); Equal (the same notion) for value functions with two values.

4. Strict containment, in general, for nondeterministic automata (Propositions 11 and 12); Equivalent notions for some nondeterministic Val automata (Section 4.1); Incomparable for alteranting automata (Proposition 13).

5. Incomparable, in general, for value functions with at least three values (Lemma 10 and Propositions 12 and 13); For value functions with two values, as relation 4 above.

Figure 2 The relations between the different notions.

▶ **Lemma 7.** *For* Val *automata whose threshold letter games are determined, Threshold Good for Gameness* $\Longrightarrow$ *Threshold History Determinism.*

Proof. Consider a Val automaton $\mathcal{A}$ whose threshold letter games are determined. Then, if $\mathcal{A}$ is not threshold history deterministic, it follows that Adam wins Eve's t-letter game on $\mathcal{A}$ for some threshold t, or Eve wins Adam's t-letter game on $\mathcal{A}$ for some t. We show below that in both cases $\mathcal{A}$ is not good for threshold games, proving the contra-positive of the claim.

Assume that Adam wins Eve's t-letter game $G_{\mathcal{A},t}$ on $\mathcal{A}$ for some threshold t with a strategy s. We can build a one-player Σ-labelled (infinite) game G_s in which the positions, which all belong to Adam, are the finite words that can be constructed along plays of $G_{\mathcal{A},t}$ that agree with s, and where for every positions u and $u \cdot a$, there is an a-labelled edge from the position u to the position $u \cdot a$. The empty word ε is the initial position. In other words, this is the one-player arena in which plays correspond to (infinite) words that occur in the letter game if Adam uses the strategy s. Notice that since s is a winning strategy in the t-letter game, all words w that are plays of G_s have $\mathcal{A}(w) \geq t$. The t-threshold game on G_s is therefore winning for Eve.

We now argue that Adam wins the product game $G_s \times \mathcal{A}$. Indeed, Adam can now use the strategy s to choose directions in G_s according to the run constructed so far in $\mathcal{A}$, and resolve conjunctions in $\mathcal{A}$ according to the history of the word and run so far. Since s is a winning strategy for Adam in the letter game, this guarantees that the resulting run ρ is such that $\mathsf{Val}(\rho) < t$. Then $\mathcal{A}$ is not threshold-good-for-games, as witnessed by G_s.

By a similar argument, if Eve wins Adam's t-letter game for some t with a strategy s, then we can construct a one-player game G_s in which all positions belong to Eve such that G_s is winning for Adam (i.e., all words have value strictly smaller than t), but in the product

$G \times \mathcal{A}$, Eve wins, i.e., can force value at least t.

Hence if either player has a winning strategy in the other player's threshold letter game for some threshold, then the automaton is not good for threshold games. ◀

We now show that letter games on Val automata whose threshold variants define Borel sets are determined. This stems from the fact that their winning condition is a union between two conditions that can be defined by threshold Val automata or their complement.

▶ **Proposition 8.** *If for some value function* Val*, all threshold* Val *automata define Borel sets, then threshold letter games on* Val *automata are determined.*

Proof. Consider Eve's t-letter game on a Val automaton $\mathcal{A}$, for some threshold $t \in \mathbb{R}$. A play of the game generates a sequence $\rho \in (\Sigma \times V)^\omega$, where Σ is $\mathcal{A}$'s alphabet and V is the finite set of its weights. We may view ρ as a pair of sequences (ρ_Σ, ρ_V), where $\rho_\Sigma \in \Sigma^\omega$ and $\rho_V \in V^\omega$. Then the winning set of Eve is $\{\rho \mid \mathsf{Val}(\rho_V) \geq t \text{ or } \mathcal{A}(\rho_\Sigma) < t\}$.

Observe that the set $S_V = \{\rho \mid \mathsf{Val}(\rho_V) \geq t\}$ can be defined by a t-threshold deterministic Val automaton $\mathcal{B}$, in which the weight of a transition over the input letter (σ, v) is v. Let $\mathcal{A}'$ be a t-threshold Val automaton that is identical to $\mathcal{A}$, except that its alphabet is $\Sigma \times V$, while the transitions are sensitive, as in $\mathcal{A}$, only to the Σ component of the input. Then the set $S_\Sigma = \{\rho \mid \mathcal{A}(\rho_\Sigma) \geq t\}$ is defined by $\mathcal{A}'$.

As the winning condition of Eve's letter game is the union of S_V and the complement of S_Σ, and as both are Borel sets, so is the winning condition. Hence, by [20] the game is determined.

The argument regarding Adam's letter game is analogous. ◀

A direct corollary of Proposition 8 is that for most of the common quantitative automata, we have that good for gameness is equivalent to threshold history determinism. In particular, this is the case for all the concrete value functions that are considered in this paper.

▶ **Theorem 9.** *Good For Gameness* $\iff$ *Threshold History Determinism for all* Val *automata on finite words, and* $\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}, \mathsf{LimInfAvg}, \mathsf{LimSupAvg}$ *and* DSum *automata on infinite words.*

Proof. It is enough to show that threshold automata of these types define Borel sets, and then the claim directly follows from Lemmas 5 and 6 and Proposition 8.

Automata on finite words. Every threshold Val automaton on finite words defines a set of finite words, which is a countable union of singletons and thus a Borel set.

Inf and Sup automata. Observe that $\mathsf{Inf}, \mathsf{Sup}$ automata on infinite words are "almost" like automata on finite words, in the sense that the value of the automaton on a word is equal to its value on some prefix of the word. Formally, for a Sup automaton $\mathcal{A}$, we have that the set of infinite words $\{w \in \Sigma^\omega \mid \mathcal{A}(w) \geq t\}$ is equal to the set of infinite words $\{w \in \Sigma^\omega \mid \text{ exists } p \in \mathbb{N} \text{ such that } \mathcal{A}(w[..p]) \geq t\}$ (when considering $\mathcal{A}$ to operate on finite words). Observe that it is indeed a Borel set, since it is a countable union of open sets. The argument for Inf-automata is analogous, having a countable intersection of closed sets.

LimInf and LimSup automata. Observe that threshold LimInf and LimSup automata are equivalent to coBüchi and Büchi automata, respectively, thus defining ω-regular languages, which are known to be Borel sets [22].

LimInfAvg and LimSupAvg automata. Directly follows from [15, Corollaries 6 and 10].

DSum automata. For DSum automata the argument stems from the continuity with respect to the Cantor topology of functions defined by DSum automata.

Consider a DSum automaton $\mathcal{A}$ and a threshold $t \in \mathbb{R}$. Define the following set of infinite words $B_t = \{w \in \Sigma^\omega \mid$ for every $n \in \mathbb{N}$ and $p_0 \in \mathbb{N}$, there exists $p > p_0$, such that $\mathcal{A}(w[..p]) \geq t - \frac{1}{n}\}$ (when considering $\mathcal{A}$ to operate on finite words).

Observe that B_t is a Borel set, since $\{w \mid \mathcal{A}(w[..p]) \geq t - \frac{1}{n}\}$ is an open set, and the existential and universal quantifiers can be defined by countable unions and intersections.

We claim that B_t is equivalent to the set $A_t = \{w \mid \mathcal{A}(w) \geq t\}$, which will prove the required statement. One direction is immediate – if a word w is in A_t then by the definition of $\mathcal{A}(w)$, there are runs of $\mathcal{A}$ on w whose supremum is at least t, admitting the membership of w in B_t.

As for the other direction, we show that for every $n \in \mathbb{N}$, there is a run r of $\mathcal{A}$ on w, such that $\mathsf{Val}(r) \geq t - \frac{1}{n}$, proving that w is in A_t. (One can then even combine these runs to create a single run that attains a value at least t.) Consider some $n \in \mathbb{N}$, and let R be the infinite set of finite runs $r_1, r_2, \ldots$ that witness the membership of w in B_t with respect to $2n$. That is, r_i is a run on a prefix of w of length at least i, whose value is at least $t - \frac{1}{2n}$. We create a single run r from R in a "Konig's lemma" approach (for simplicity, we detail the construction for a nondeterministic automaton, and later explain how to extend it to an alternating automaton):

We choose the first transition t_1 in r to be a transition that appears as the first transition in infinitely many r_i's. We then choose the next transition t_2 to be a transition that appears as the second transition, where t_1 is the first transition, in infinitely many r_i's, and so on. Notice that r is indeed a run of $\mathcal{A}$ and its value is at least $t - \frac{1}{n}$: By the discounted-sum value function, if the value of a long enough prefix is at least $t - \frac{1}{2n}$, the value of the entire run cannot be smaller than $t - \frac{1}{n}$.

Now, for an alternating automaton, rather than "choosing transitions" we need to "resolve the nondeterminism", while ensuring that the choice we make appears in infinitely many runs after the previous nondeterministic and *universal* choices that were already made. ◀

History determinism and determinizability by pruning obviously imply their threshold versions; Figure 3 demonstrates that the converse does not hold.

▶ **Lemma 10.**

- *History Determinism* $\underset{\not\Longleftarrow}{\Longrightarrow}$ *Threshold History Determinism;*
- *Determinizability by Pruning* $\underset{\not\Longleftarrow}{\Longrightarrow}$ *Threshold Determinizability by Pruning;*
- *History Determinism* $\not\Longleftarrow$ *Threshold Determinizability by Pruning*

Proof. The implications are straightforward: a winning strategy for each player in their letter game is also a winning strategy in their t-threshold letter game, for every threshold $t \in \mathbb{R}$; further, if an automaton $\mathcal{A}'$ that results from pruning $\mathcal{A}$ is equivalent to $\mathcal{A}$, then for every threshold t and word w, if $\mathcal{A}(w) \geq t$ then $\mathcal{A}'(w) \geq t$.

As for the non-implications, Figure 3 provides such counter examples, which hold, with some variations, with respect to every non-trivial value function with at least 3 values, and in particular with respect to all value functions discussed in the paper.

Consider, for example, the automaton $\mathcal{A}$ of Figure 3 with respect to the Sup value function. It is not history deterministic, since if the nondeterminism in q_0 is resolved by going to q_1, the resulting automaton is not equivalent to $\mathcal{A}$ with respect to the finite word aa and infinite word a^ω, and if it is resolved by going to q_2, the resulting automaton fails on ab and ab^ω.

On the other hand, $\mathcal{A}$ is threshold determinizable by pruning and threshold history deterministic: For a threshold up to 1, the nondeterminism is resolved by going to q_1 and for the threshold 2 by going to q_2. ◀

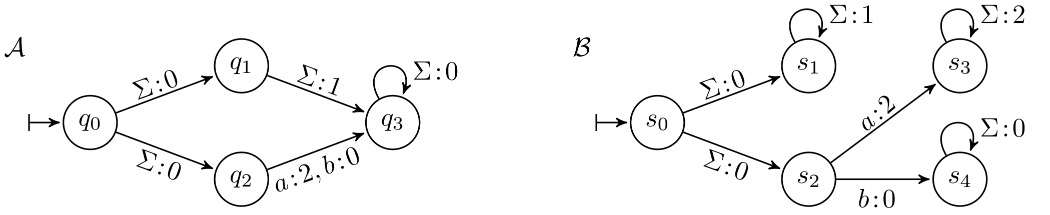


Figure 3 Nondeterministic automata that are threshold history deterministic and threshold determinizable by pruning, but not history deterministic and not determinizable by pruning. The automaton $\mathcal{A}$ has this property with respect, for example, to the Sum/DSum/Sup value functions, and $\mathcal{B}$ with respect, for example, to Avg/LimSup/LimInf/LimSupAvg/LimInfAvg.

History Determinism $\neq$ Determinizability by Pruning

For nondeterministic automata, it is clear that determinizability by pruning implies history determinism: the pruning provides a strategy for Eve in her letter game.

▶ **Proposition 11.** *For nondeterministic automata, determinizability by pruning $\Longrightarrow$ history determinism.*

The converse was shown to be false for Büchi and coBüchi automata [7], directly implying the same for LimSup and LimInf automata. Considering LimInfAvg and LimSupAvg automata, the automaton depicted in Figure 4, which is similar to the coBüchi automaton in [7, Figure 3], is history deterministic but not determinizable by pruning.

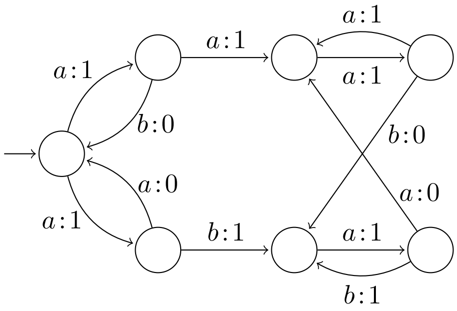


Figure 4 (Similar to [7, Figure 3].) A history deterministic LimInfAvg or LimSupAvg automaton that is not determinizable by pruning. (Missing transitions lead to a sink with a 0-weighted self loop on both a and b.) It is history deterministic by a strategy that chooses in the initial state to go up if and only if it went down the previous time. Following this strategy in the letter game, Eve returns infinitely often to the initial state only on $(aaab)^\omega$, getting a value $\frac{1}{2}$, which is also the automaton's value on it. For every other word, the run of Eve moves to the right part of the automaton, which is deterministic, guaranteeing Eve the optimal value on the word. On the other hand, every pruning of it yields an automaton whose value on either a^ω or $(ab)^\omega$ is $\frac{1}{2}$ instead of 1.

▶ **Proposition 12.** *For nondeterministic* LimInf*,* LimSup*,* LimInfAvg*, and* LimSupAvg *automata, history determinism $\not\Longrightarrow$ determinizability by pruning.*

For alternating automata, it turns out that (threshold) history determinism and determinizability by pruning are incomparable, as demonstrated in Figure 5.

▶ **Proposition 13.** *For (Boolean and quantitative) alternating automata, determinizability by pruning $\not\Rightarrow$ history determinism, good for gameness.*

Proof. The claim holds for Boolean automata as well as quantitative automata with every non-trivial value function. Consider the alternating finite automaton on finite words (which can also be viewed, for example as a Sup automaton) in Figure 5. It does not accept any word, and can be determinized by pruning the right nondeterministic transition. However, it is not history deterministic: Eve wins Adam's letter game, by choosing the right nondeterministic transition. ◀

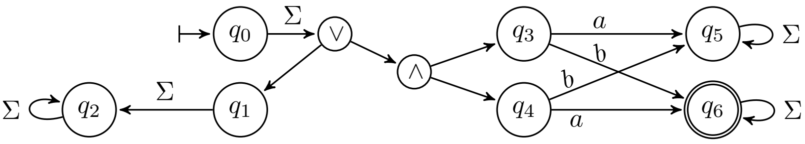


Figure 5 An alternating finite automaton on finite words that is determinizable by pruning, but not history deterministic nor good for games.

4.1 When (History Determinism = Determinizability by Pruning)

In general for nondeterministic automata, determinizability by pruning is strictly contained in history determinism; here we study when the two notions coincide. In the Boolean setting, they are equivalent for nondeterministic finite automata on finite words (NFAs) [19] as well as for nondeterministic weak automata on infinite words [21]. Here we analyse general properties of value functions that guarantee this equivalence, and then consider specific value functions on finite and infinite words. The general properties that we analyze relate to how "sensitive" the value function is to the prefix, current position, and suffix of the weight sequence.

We begin by defining *cautious*[5] *strategies* for Eve in the letter game, that we then use to define value functions that are "present focused". Intuitively, a strategy is cautious if it avoids mistakes, that is, it only builds run prefixes that can still achieve the maximal value of any continuation of the word so far.

▶ **Definition 14** (Cautious strategies). *Consider Eve's letter game on a* Val *automaton $\mathcal{A}$. A move (transition) $t = q \xrightarrow{\sigma:x} q'$ of Eve, played after some run ρ ending in a state q, is* non-cautious *if for some word w, there is a run π' from q over σw such that* $\mathsf{Val}(\rho\pi')$ *is strictly greater than the value of* $\mathsf{Val}(\rho\pi)$ *for any π starting with t.*

A strategy is cautious *if it makes no non-cautious moves.*

We call a value function *present focused* if, morally, it depends on the prefixes of the value sequence, formalized by winning the letter game via cautious strategies.

▶ **Definition 15** (Present-focused value functions). *A value function* Val*, on finite or infinite sequences, is* present focused *if for all automata $\mathcal{A}$ with value function* Val*, every cautious strategy in the letter game on $\mathcal{A}$ is also a winning strategy in that game.*

Value functions on finite sequences are present focused, as they can only depend on prefixes.

[5] Similar transitions are sometimes called "residual" in the literature.

▶ **Lemma 16.** *Every value function* Val *on finite sequences is present focused.*

Proof. Assume Eve plays a cautious strategy s in some letter game on an automaton $\mathcal{A}$ on finite words. Towards a contradiction, assume that there is a finite play π, in which Adam plays some word w and Eve plays a run ρ over w such that $\mathsf{Val}(\rho) < \mathcal{A}(w)$. Then, let ρ' be the longest prefix of ρ such that the highest value of a run over w starting with ρ' is $\mathcal{A}(w)$. Since ρ is not a run with value $\mathcal{A}(w)$, ρ' is a strict prefix of ρ. However, since ρ' is the longest prefix that could be continued into a run with value $\mathcal{A}(w)$, Eve's next move after ρ' must be non-cautious, contradicting that s never plays non-cautious moves. ◀

▶ Remark 17. Value functions on infinite sequences are not necessarily present focused. For example, consider the automaton depicted in Figure 1, but viewed as a Sup automaton on infinite words rather than a DSum automaton. Observe that Eve can forever stay in q_0, always having the potential to continue to an optimal run with value 2, but never fulfilling this potential.

We now define "suffix monotonicity" of value functions, which, with present-focus, will guarantee the equivalence of history determinism and determinizability by pruning.

▶ **Definition 18.** *A value function* Val *is* suffix monotonic *if for every finite set* $S \subset \mathbb{Q}$*, sequence* $\alpha \in S^*$ *and sequences* $\beta, \beta' \in S^\infty$*, we have* $\mathsf{Val}(\beta) \geq \mathsf{Val}(\beta')$ *iff* $\mathsf{Val}(\alpha\beta) \geq \mathsf{Val}(\alpha\beta')$*.*

Observe that the above definition does not consider arbitrary sequences of rational numbers, but rather sequences of finitely many different rational numbers, which is the case in sequences of weights that are generated by runs of quantitative automata.

Value functions that are *suffix dependent* (namely Val functions such that for every finite set $S \subset \mathbb{Q}$, sequences $\alpha, \alpha' \in S^*$ and sequence $\beta \in S^\infty \setminus \{\varepsilon\}$, we have $\mathsf{Val}(\alpha\beta) = \mathsf{Val}(\alpha'\beta)$) are obviously suffix monotonic. Examples for such value functions are the acceptance condition of NFAs (i.e, a "last" value function, that depends only on the last weight of 0 for rejection and 1 for acceptance), all ω-regular conditions (which depend on the states/transitions that are visited infinitely often), LimInf, LimSup, $\mathsf{LimInfAvg}$, and $\mathsf{LimSupAvg}$. Examples for value functions that are suffix monotonic but not suffix dependent are $\mathsf{Sum}, \mathsf{Avg}$ and DSum, and examples for value functions that are not suffix monotonic are Inf and Sup.

We next show that suffix monotonicity together with present-focus guarantee the equivalence of history determinism and determinizability by pruning. The idea is that under these conditions, every cautious strategy in the letter game can be arbitrarily pruned into a positional strategy (with respect to the automaton states).

▶ **Theorem 19.** *For nondeterministic* Val *automata, where* Val *is a present-focused and suffix-monotonic value function, we have that history determinism* $\iff$ *determinizability by pruning.*

Proof. We show that Eve wins her letter game on $\mathcal{A}$ with a positional strategy, which implies that $\mathcal{A}$ is determinizable by pruning.

Let s be a cautious strategy for Eve in the letter game on $\mathcal{A}$. Let $\hat{s}$ be an arbitrary positional strategy that only uses transitions also used by s. We argue that $\hat{s}$ is also cautious. Indeed, if $\hat{s}$ chooses $\tau = q \xrightarrow{\sigma:x} q'$ after a play $(\hat{w}, \hat{\rho})$ of the letter game, there is some play (w, ρ) from which s plays τ. Since s is cautious, for every word v and every run π' from q over σv, there is a run π from q starting with τ such that $\mathsf{Val}(\rho\pi) \geq \mathsf{Val}(\rho\pi')$. Thus, by suffix monotonicity, we have $\mathsf{Val}(\pi) \geq \mathsf{Val}(\pi')$, and then again by the other direction of suffix monotonicity, we get that $\mathsf{Val}(\hat{\rho}\pi) \geq \mathsf{Val}(\hat{\rho}\pi')$, implying that $\hat{s}$ choosing τ is a cautious move.

Then $\mathcal{A}$ is determinisable by pruning: the subautomaton $\mathcal{A}_{\hat{s}}$ that only has transitions used by $\hat{s}$ is equivalent to $\mathcal{A}$. Indeed, for every word w, $\mathcal{A}_{\hat{s}}(w)$ is $\mathsf{Val}(\rho_w)$, where ρ_w is the unique run of $\mathcal{A}_{\hat{s}}$ over w. The run ρ_w is also the run built by $\hat{s}$ in the letter game over w. Since $\hat{s}$ is cautious and Val is present focused, we have that $\hat{s}$ is a history-deterministic strategy, which guarantees that $\mathsf{Val}(\rho_w) = \mathcal{A}(w)$, giving us the equivalence of $\mathcal{A}$ and $\mathcal{A}_{\hat{s}}$. ◀

▶ Remark 20. Both present-focus and suffix-monotonicity are necessary in Theorem 19. For example LimInf is suffix monotonic, but LimInf automata are not determinizable by pruning. On the other hand, Figure 6 demonstrates a present-focused value function whose history deterministic automata on finite words are not determinizable by pruning.

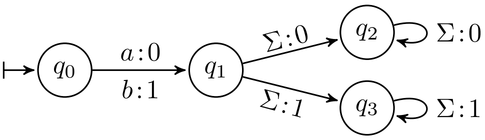


Figure 6 A nondeterministic Val automaton $\mathcal{A}$ on finite words with the value function $\mathsf{Val}(\rho) = 1$ if ρ has both even and odd values, and 0 otherwise. Notice that Val is present focused and $\mathcal{A}$ is history deterministic but not determinizable by pruning.

We now apply these results to specific value functions.

▶ **Theorem 21.** [6] *For nondeterministic* Sum *and* Avg *automata (on finite words), history determinism* $\iff$ *determinizability by pruning.*

Proof. From Lemma 16 and Theorem 19 and the suffix monotonicity of these value functions. ◀

We continue with showing that DSum is present focused due to the function's continuity.

▶ **Lemma 22.** DSum *on infinite sequences is a present-focused value function.*

▶ **Theorem 23** ([17, Section 5])**.** *For nondeterministic* DSum *automata on finite and infinite words, history determinism* $\iff$ *determinizability by pruning.*

Proof. The claim, which was also proved in [17, Section 5], is a direct consequence of Lemmas 16 and 22 and Theorem 19 and the suffix monotonicity of the DSum value functions. ◀

The Inf and Sup value function are not suffix monotonic, and indeed the proof of Theorem 19 does not hold for them – not every cautious transition of a history deterministic Sup automaton on finite words can be used for pruning it into a deterministic automaton. Yet, also for Inf and Sup automata on finite words we have that history determinism is equivalent to determinizability by pruning, using other characteristics of these value functions – we can prune the automaton, by choosing the transitions that are used by the strategy of the letter game after reading words with minimal values for Sup and maximal value for Inf.

▶ **Theorem 24.** *For nondeterministic* Inf *and* Sup *automata on finite words, history determinism* $\iff$ *determinizability by pruning.*

[6] A slightly weaker result is given in [3, Theorem 5.1]: a Sum automaton is history deterministic *with a finite-memory strategy for resolving the nondeterminism* if and only if it is determinizable by pruning.

5 Applications to Quantitative Synthesis

Establishing the non-equivalence of history determinism, good for gameness and their threshold versions leaves us with the question of which definitions, if any, are the most useful or interesting ones. We explore this question from the perspective of quantitative synthesis.

In the Boolean setting, Church's classical synthesis problem asks for a transducer $\mathcal{T}$ that produces, letter by letter, for every input sequence $I \in \Sigma_I^\omega$ an output sequence $\mathcal{T}(I) \in \Sigma_O^\omega$ such that $I \otimes \mathcal{T}(I) \in L$ for some specification language $L \in (\Sigma_I \otimes \Sigma_O)^\omega$. This synthesis requirement is *global*, in the sense that the output of all input sequences should satisfy the same constraint. A *local* variant of the problem, termed "good enough synthesis" in [1], considers each input sequence I separately, requiring that the output $\mathcal{T}(I)$ of the transducer on the input I satisfies $I \otimes \mathcal{T}(I) \in L$ only if $I \otimes O \in L$ for some sequence $O \in \Sigma_O^\omega$.

In quantitative synthesis, the specification is a function $f : (\Sigma_I \times \Sigma_O)^\omega \to \mathbb{R}$ (generalizing languages $L : (\Sigma_I \times \Sigma_O)^\omega \to \{\texttt{true}, \texttt{false}\}$), and the two synthesis problems above naturally generalize into two quantitative variants each – requiring either the best possible value or a value matching a given threshold. We thus have four variants of quantitative synthesis: Global/Local Threshold/Best-value synthesis. It turns out that **good for gameness is closely related to global synthesis, while history determinism is closely related to local synthesis**, both for the threshold and best-value settings.

Global Threshold and Best-value Synthesis. The global threshold variant is the closest to Church synthesis: given a function f and a threshold $t \in \mathbb{R}$, it requires that $f(I \otimes \mathcal{T}(I)) \geq t$ for all input sequences I. In the best-value version, t is not given and we are interested in what is the highest threshold that the system can guarantee.

Analogously to the Boolean setting, a t-threshold good for games Val automaton $\mathcal{A}$ realizing f can be used instead of a deterministic automaton to solve the global threshold synthesis problem: $\mathcal{A}$ is turned into a t-threshold Val game $G_\mathcal{A}$, in which Adam controls the input letters and Eve controls the output letters. Then, the synthesis problem is realizable if and only if Eve has a winning strategy in $G_\mathcal{A}$. If $\mathcal{A}$ is nondeterministic, Eve's winning strategy in $G_\mathcal{A}$ induces a transducer for the synthesis problem. In the best-value case, the same is true, but $\mathcal{A}$ must be good for games, rather than just for t-threshold games, and it is Eve's optimal strategy, if it exists, that induces the solution transducer.

Local Best-value and Threshold Synthesis. We define $\texttt{Best}_f(I) = \sup_{O \in \Sigma_O^\omega} f(I \otimes O)$ for $I \in \Sigma_I^\omega$, i.e., the best value that the input I can get, or converge to, according to f. The local best value synthesis problem requires that for every $I \in \Sigma_I^\omega$, we have $f(I \otimes \mathcal{T}(I)) = \texttt{Best}_f(I)$. Since $\texttt{Best}_f(I)$ is a supremum, it need not be attained by any word; then the synthesis problem is unrealisable, even if the system could force a value arbitrarily close to $\texttt{Best}_f(I)$. The threshold variant requires that for every $I \in \Sigma_I^\omega$, such that $\texttt{Best}_f(I) \geq t$, we have $f(I \otimes \mathcal{T}(I)) \geq t$, for a given threshold $t \in \mathbb{R}$.

The local best value (or t-threshold) synthesis problem of a function given by deterministic (or even history-deterministic nondeterministic) automata and the problem of whether a nondeterministic automaton is (t-threshold) history deterministic reduce to each other. The relationship between good-enough synthesis [1] and history determinism was noted for visibly pushdown automata in [13]; a similar reduction in [12] reduces the approximative local best-value synthesis of deterministic quantitative automata over finite words by finite transducers to the notion of r-regret determinisability, that is, whether a nondeterministic automaton is close enough to a deterministic automaton obtained by pruning its product with a finite memory. Our reductions are in the same spirit, but relate the synthesis problem to history determinism rather than determinisability, and obtain a two-way correspondence for all history-deterministic nondeterministic quantitative automata. In the alternating case, only one direction is preserved, and only for realisability, rather than synthesis.

▶ **Proposition 25.** *Deciding the local best value (resp. t-threshold) synthesis problem with respect to a function f given by a (t-threshold) history deterministic nondeterministic* Val*-automaton $\mathcal{A}$ and deciding whether a nondeterministic* Val*-automaton $\mathcal{A}'$ is (t-threshold) history deterministic are linearly inter-reducible. Furthermore, the witness of (t-threshold) history determinism of $\mathcal{A}'$ is implementable by the same computational models as a solution to the best-value (t-threshold) synthesis of $\mathcal{A}$.*

6 Conclusions

We have painted a picture of how definitions of good for gameness and history determinism behave in the quantitative setting, and how they relate to quantitative synthesis. Our work opens up many directions for further work, of which we name a few.

- The reductions between local synthesis and history determinism motivate expanding methods used to decide history determinism of ω-regular automata to quantitative ones.
- So far, we have restricted our attention to determined games, but one could also consider more general classes of games and study the effect of composition in that setting.
- One appeal of good for games and history deterministic automata is that they can be more expressive and more succinct than deterministic ones, while their synthesis problems retain the same complexity. The expressivity and succinctness of quantitative good for games and history deterministic automata is open for most value functions.
- It is natural to look at approximative versions of the discussed notions (as has been done, see the related work section); we expect our results to also generalise in that direction.

References

1 Shaull Almagor and Orna Kupferman. Good-enough synthesis. In *CAV*, volume 12225 of *Lecture Notes in Computer Science*, pages 541–563. Springer, 2020.

2 Shaull Almagor, Orna Kupferman, Jan Oliver Ringert, and Yaron Velner. Quantitative assume guarantee synthesis. In *International Conference on Computer Aided Verification*, pages 353–374. Springer, 2017.

3 Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Trans. Algorithms*, 6(2):28:1–28:36, 2010.

4 Roderick Bloem, Krishnendu Chatterjee, Thomas A Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *International Conference on Computer Aided Verification*, pages 140–156. Springer, 2009.

5 Udi Boker. Quantitative vs. weighted automata. In *Proc. of Reachbility Problems*, pages 1–16, 2021.

6 Udi Boker and Guy Hefetz. Discounted-sum automata with multiple discount factors. In *Proc. of CSL*, volume 183 of *LIPIcs*, pages 12:1–12:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

7 Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Proceedings of ICALP*, pages 89–100, 2013.

8 Udi Boker and Karoliina Lehtinen. Good for games automata: From nondeterminism to alternation. In *Proceedings of CONCUR*, volume 140 of *LIPIcs*, pages 19:1–19:16, 2019.

9 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 *Language and Automata Theory and Applications*, pages 3–23. Springer, 2016.

10 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating weighted automata. In *Proceedings of FCT*, pages 3–13, 2009.

11 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Proceedings of ICALP*, pages 139–150, 2009.

12 Emmanuel Filiot, Christof Löding, and Sarah Winter. Synthesis from weighted specifications with partial domains over finite words. In Nitin Saxena and Sunil Simon, editors, *FSTTCS*, volume 182 of *LIPIcs*, pages 46:1–46:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

13 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A bit of nondeterminism makes pushdown automata expressive and succinct. In *MFCS*, volume 202 of *LIPIcs*, pages 53:1–53:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.MFCS.2021.53`.

14 Thomas Henzinger and Nir Piterman. Solving games without determinization. In *Proceedings of CSL*, pages 395–410, 2006.

15 Paul Hunter, Arno Pauly, Guillermo A. Pérez, and Jean-François Raskin. Mean-payoff games with partial observation. *Theor. Comput. Sci.*, 735:82–110, 2018.

16 Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Reactive synthesis without regret. In Luca Aceto and David de Frutos-Escrig, editors, *CONCUR*, volume 42 of *LIPIcs*, pages 114–127, 2015.

17 Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Minimizing regret in discounted-sum games. In Jean-Marc Talbot and Laurent Regnier, editors, *CSL*, volume 62 of *LIPIcs*, pages 30:1–30:17, 2016.

18 Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Reactive synthesis without regret. *Acta Informatica*, 54(1):3–39, 2017.

19 Orna Kupferman, Shmuel Safra, and Moshe Y Vardi. Relating word and tree automata. *Ann. Pure Appl. Logic*, 138(1-3):126–146, 2006. Conference version in 1996.

20 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2), 1975.

21 Gila Morgenstern. Expressiveness results at the bottom of the ω-regular hierarchy. M.Sc. Thesis, The Hebrew University, 2003.

22 Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 389–455. Springer, 1997.

A Proofs of Section 4

Proof of Lemma 5. One direction is immediate: if an automaton $\mathcal{A}$ is good for all games then it is also good for all threshold games. Indeed, assuming that $\mathcal{A}$ is good for games, if the value of a game G is v, then the value of the product game $G \times \mathcal{A}$ is also v. Then, for all thresholds t, Eve wins the t-threshold game on both G and $G \times \mathcal{A}$ if and only if $v \geq t$.

As for the other direction, assume $\mathcal{A}$ is good for threshold games. Let G be a game with value v. Since $\mathcal{A}$ composes with threshold games, considering the v-threshold game on G, we know that Eve can achieve at least v in the product v-threshold game $G \times A$. Conversely, let $v' \geq v$ be the value of $G \times A$. Since Eve wins the v'-threshold game on $G \times A$, and $\mathcal{A}$ is good for threshold games, Eve can also achieve at least v' in G, i.e., $v' = v$, the value of G. ◀

Proof of Lemma 6. Consider a threshold history deterministic automaton $\mathcal{A}$ over an alphabet Σ, realizing a function f. Then for every threshold $t \in \mathbb{R}$, Eve has a winning strategy s' in the t-threshold letter game on $\mathcal{A}$.

Now, consider a Σ-labelled t-threshold game G with payoff function f, in which Eve has a winning strategy s. Then in the product game $G \times \mathcal{A}$, Eve can combine s and s' into a strategy $\hat{s}$, so that s guarantees that any play $\pi = (w, \rho)$ that agrees with $\hat{s}$ reads a word w such that $\mathcal{A}(w) \geq t$, and s' guarantees that $\mathsf{Val}(\rho) \geq t$ (since $\mathcal{A}(w) \geq t$).

By a similar argument, if Adam has a winning strategy in his threshold letter game, he can combine it with his winning strategy in a threshold game for getting a winning strategy in the product threshold game. ◀

A.1 Proofs of Section 4.1

Proof of Lemma 22. Consider a λ-DSum Val automaton $\mathcal{A}$ and let m be the maximal absolute transition weight in $\mathcal{A}$. Observe that for every word w and state q of $\mathcal{A}$, we have $|\mathcal{A}^q(w)| \leq \frac{m}{1-\lambda}$.

Let s be a cautious strategy of Eve in the letter game on $\mathcal{A}$. By the definition of a cautious strategy, for every finite word u, playing according to s on u generates a finite run ρ that ends in some state q, such that for every infinite word v, there is an infinite run π on v from q, such that $\mathsf{Val}(\rho\pi) = \mathcal{A}(uv)$.

Now, consider a word w, let r be the run of $\mathcal{A}$ on w that is generated by following s, and let r' be an optimal run of $\mathcal{A}$ on w. For every position i, let q_i be the state that $r[0..i]$ ends in and q'_i be the state that $r'[0..i]$ ends in. By the cautiousness of s, for every position i, there is a run π from q_i on $w[i+1..]$, such that for every run π' from q'_i on $w[i+1..]$, we have $\mathsf{Val}(r[0..i]) + \lambda^i \mathsf{Val}(\pi) \geq \mathsf{Val}(r'[0..i]) + \lambda^i \mathsf{Val}(\pi')$.

Since $\mathsf{Val}(\pi) \leq \frac{m}{1-\lambda}$ and $\mathsf{Val}(\pi') \geq -\frac{m}{1-\lambda}$, we get that $\mathsf{Val}(r'[0..i]) - \mathsf{Val}(r[0..i]) \leq \frac{2m\cdot\lambda^i}{1-\lambda}$. Since $\lim_{i\to\infty} \frac{2m\cdot\lambda^i}{1-\lambda} = 0$, we get that $\mathsf{Val}(r) = \mathsf{Val}(r')$, implying that Eve wins the letter game. ◀

Proof of Theorem 24. We provide the proof for Sup automata and then describe the required changes for adapting it to Inf automata.

Consider a history deterministic Sup automaton $\mathcal{A}$ on finite words in Σ^*, whose history determinism is witnessed by a strategy s. We derive from s a positional strategy s', by taking for every state q of $\mathcal{A}$ and letter $\sigma \in \Sigma$, the transition that s chooses over a minimal prefix, where minimality is with respect to the Sup function.

Formally, for every state q, let $m(q)$ be a Sup-minimal run that reaches q along s; namely $m(q) = \rho$, such that ρ is a run of $\mathcal{A}$ that agrees with s and ends in q, and such that for every run ρ' of $\mathcal{A}$ that agrees with s and ends in q, we have $\mathsf{Sup}(\rho) \leq \mathsf{Sup}(\rho')$. (Notice that since there are finitely many weights in $\mathcal{A}$, such a minimal run, which need not be unique, always exists.) For every state q of $\mathcal{A}$ and letter $\sigma \in \Sigma$, we define $s'(q, \sigma) = t$, such that s chooses t over the prefix run $m(q)$ and current letter σ.

We claim that s' is cautious. Indeed, for the correctness proof, we shall change s into s' iteratively, considering in each iteration a single state q and letter σ. Assume by way of contradiction that exists a word $u \in \Sigma^*$ on which s' generates a path τ that ends in a state q, such that $s'(u\sigma) = t$ for a non-cautious transition t. Without loss of generality, we may assume that this is not the case for any strict prefix of u, as otherwise we can consider that prefix instead of u.

By the definition of non-cautiousness, there exists a word w, such that the maximal value of $\mathsf{Sup}(\tau\pi)$ for a run π from q over σw starting with t is strictly smaller than the maximal value of $\mathsf{Sup}(\tau\pi')$ where π' is a run from q over σw that does not start with t.

It thus follows that $\mathsf{Sup}(\pi') > \mathsf{Sup}(\tau)$ and that for every run π from q over σw starting with t, we have $\mathsf{Sup}(\pi') > \mathsf{Sup}(\pi)$. Now, let ρ be a run that witnesses t's minimality in the definition of s', namely s chooses t when reading σ after reaching q over ρ, and for every run ρ' that ends in q, we have $\mathsf{Sup}(\rho) \leq \mathsf{Sup}(\rho')$.

Then, in particular, $\mathsf{Sup}(\rho) \leq \mathsf{Sup}(\tau)$. Hence, $\mathsf{Sup}(\pi') > \mathsf{Sup}(\rho)$. Therefore, for every run π from q over σw starting with t, we have $\mathsf{Sup}(\rho\pi') > \mathsf{Sup}(\rho\pi)$, contradicting the cautiousness of s.

Having that s' is cautious, we get from Lemma 16 that it is also winning in the letter game, implying that the deterministic automaton that results from pruning $\mathcal{A}$ along s' is indeed equivalent to $\mathcal{A}$.

Now, for Inf automata, the proof is analogous, choosing the Inf-maximal run rather than the Sup-minimal run, switching between some $\geq$ and $\leq$ and between some $<$ and $>$, and providing the following final argument: For every run π from q over σw starting with t, we have $\mathsf{Inf}(\pi) < \mathsf{Inf}(\tau)$ and $\mathsf{Inf}(\pi) < \mathsf{Inf}(\pi')$. Now, let ρ be a run that witnesses t's maximality in the definition of s', namely s chooses t when reading σ after reaching q over ρ, and for every run ρ' that ends in q, we have $\mathsf{Inf}(\rho) \geq \mathsf{Inf}(\rho')$.

Then, in particular, $\mathsf{Inf}(\rho) \geq \mathsf{Inf}(\tau)$. Hence, for every run π from q over σw starting with t, we have $\mathsf{Inf}(\pi) < \mathsf{Inf}(\rho)$ and $\mathsf{Inf}(\pi) < \mathsf{Inf}(\pi')$. Hence, for every run π from q over σw starting with t, we have $\mathsf{Inf}(\rho\pi) < \mathsf{Inf}(\rho\pi')$, contradicting the cautiousness of s. ◀

B Proofs of Section 5

Proof of Proposition 25.

$\Longrightarrow$: Reducing the synthesis problem to the history-determinism problem.

The idea of the reduction (both in the best-value and t-threshold case) is to turn output letter choices in $\mathcal{A}$ into nondeterministic choices in $\mathcal{A}'$. Then $\mathcal{A}'$ maps $I \in \Sigma_I^\omega$ onto $\mathtt{Best}_{\mathcal{A}}(I)$. A solution to the synthesis problem for $\mathcal{A}$ corresponds exactly to a function that resolves the nondeterminism of $\mathcal{A}'$ on the fly to build a run with value $\mathtt{Best}_{\mathcal{A}}(I)$, that is, a witnesses of the history determinism of $\mathcal{A}'$. If $\mathcal{A}$ is itself nondeterministic, then $\mathcal{A}'$ will have both the nondeterminism of $\mathcal{A}$ and the nondeterminism that stems from the choice of output letters. As long as the nondeterminism of $\mathcal{A}$ is history deterministic, the nondeterminism of $\mathcal{A}'$ is history deterministic if and only if $\mathcal{A}$ is local best value realisable.

More formally, first let us define formally the projection of $\mathcal{A}$ onto its first component: $\mathcal{A}' = (\Sigma_I, Q, \iota, \delta')$, where $\delta'(q,a) = \bigvee_{b\in\Sigma_O} \delta(q,(a,b))$. In other words, the automaton $\mathcal{A}'$ moves the Σ_O letters from the input word into a nondeterministic choice. It implements a mapping of inputs $I \in \Sigma_I^\omega$ onto $\mathtt{Best}_{\mathcal{A}}(I)$. We now argue that witnesses of history determinism for $\mathcal{A}'$ coincide with solutions to the best-value synthesis problem for $\mathcal{A}$. Let s be the witness of the history determinism of $\mathcal{A}$.

We first argue that a solution s' to the best-value synthesis problem for $\mathcal{A}$, combined with s is a witness that $\mathcal{A}'$ is history-deterministic. Indeed, in Eve's letter game on $\mathcal{A}'$, Eve has two types of choices: a choice $\bigvee_{b\in\Sigma_O} \delta(q,(a,b))$ of an Σ_O-letter, and the choice in $\delta(q,(a,b))$ that stems from $\mathcal{A}$. Let $\hat{s}$ be the strategy that after a run prefix ρ ending in a state q over a word $w \in \Sigma_I$ chooses the letter $s'(w)$, that is, the disjunct $\delta(q,(a,s'(w)))$ in the disjunction $\bigvee_{b\in\Sigma_O} \delta(q,(a,b))$. Then, from $\delta(q,(a,s'(w)))$, $\hat{s}$ behaves as s would after a run of $\mathcal{A}$ over $w \otimes \overline{s}(w)$.

First, observe that a run ρ of $\mathcal{A}'$ over $I \in \Sigma_I^\omega$, labelled with the choices of Σ_O-letters forming some $O \in \Sigma_O^\omega$, corresponds to a run of $\mathcal{A}$ over $I \otimes O$ with the same value.

Then, since s' is a solution to the best value synthesis problem, it guarantees that given an input word Σ_I, the sequence of Σ_O letters chosen by $\hat{s}$ is $\bar{s}(I)$, and $\mathcal{A}(I \otimes \bar{s}(I)) = \texttt{Best}_{\mathcal{A}}(I)$. Then, as s witnesses the history determinism of $\mathcal{A}$, $\hat{s}$ guarantees that ρ has value $\mathcal{A}(I \otimes \bar{s}(I)$, that is, $\hat{s}$ witnesses the history determinism of $\mathcal{A}'$.

For the converse direction, assume $\mathcal{A}'$ is history deterministic, as witnessed by some strategy s. We claim that s induces a solution s' to the synthesis problem for $\mathcal{A}$ as follows: after reading an finite sequence of inputs $Ia \in \Sigma_I^*$, s has built some run ρ over I that ends in a state q, after which s resolves a disjunction $\bigvee_{b \in \Sigma_O} \delta(q, \binom{a}{b})$ by choosing some $b \in \Sigma_O$. We then set $s'(Ia) = b$. Then, as s witnesses that $\mathcal{A}'$ is history-deterministic, the run chosen by s over an input $I \in \Sigma_I^\omega$ has the value $\texttt{Best}_{\mathcal{A}}(I)$. By construction of $\mathcal{A}'$ and s', this is the value $\mathcal{A}(I \otimes \bar{s'}(I)$, that is, s' is indeed a solution to the synthesis problem on $\mathcal{A}$. Furthermore, observe that an implementation of s also implements s' by ignoring the outputs of s that do not choose Σ_O letters, so the memory of the solution to the synthesis problem is bounded by the memory required by a witness of history determinism.

$\Longleftarrow$: Reducing the history-determinism problem to the synthesis problem.

Dually to the previous translation, we turn the nondeterminism in an automaton $\mathcal{A}$ into choices of output letters in the best-value synthesis problem. We build a deterministic automaton $\mathcal{A}'$ that is similar to $\mathcal{A}$ except that it reads both an input letter and a transition; then a transition can only be chosen if it is the second element of the input (that is, the output letter). Then $\mathcal{A}'$ maps valid runs of $\mathcal{A}$ to their value and a solution to the local best value synthesis problem of $\mathcal{A}'$ corresponds exactly to a witness of history-determinism for $\mathcal{A}$.

Formally, let $\mathcal{A}'$ be the Val automaton $(\Sigma \times \Delta, Q, \iota, \delta')$ where $\delta'(q, (a, q \xrightarrow{a:x} q')) = (x, q')$ if $(x, q') \in \delta(q, a)$. $\mathcal{A}'$ maps valid runs of $\mathcal{A}$ written as pairs (w, r) where r is a run of $\mathcal{A}$ over w, onto $\mathsf{Val}(r)$ and in particular $\texttt{Best}_{\mathcal{A}'}(I) = \mathcal{A}(I)$.

We claim that $\mathcal{A}'$ is best-value realisable if and only $\mathcal{A}$ is history-deterministic. Indeed, a solution s to the best value synthesis problem of $\mathcal{A}'$ corresponds to a function building a run of $\mathcal{A}$ over the input I transition by transition such that the value of the run is $\texttt{Best}_{\mathcal{A}'}(I)$. Since $\texttt{Best}_{\mathcal{A}'}(I)$ is $\mathcal{A}(I)$, s is precisely a witness of history-determinism in $\mathcal{A}$. Similarly, a witness of history-determinism in $\mathcal{A}$ induces a solution to the best value synthesis problem for $\mathcal{A}'$ since it builds a run of $\mathcal{A}$ over I with value at least $\mathcal{A}(I)$, exactly what is required from a solution to the best value synthesis. ◀

Local First-Order Logic with Two Data Values

Benedikt Bollig
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, France

Arnaud Sangnier
IRIF, Université de Paris, CNRS, France

Olivier Stietel
IRIF, Université de Paris, CNRS, France
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, France

Abstract

We study first-order logic over unordered structures whose elements carry two data values from an infinite domain. Data values can be compared wrt. equality so that the formalism is suitable to specify the input-output behavior of various distributed algorithms. As the logic is undecidable in general, we introduce a family of local fragments that restrict quantification to neighborhoods of a given reference point. Our main result establishes decidability of the satisfiability problem for one of these non-trivial local fragments. On the other hand, already slightly more general local logics turn out to be undecidable. Altogether, we draw a landscape of formalisms that are suitable for the specification of systems with data and open up new avenues for future research.

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

Keywords and phrases first-order logic, data values, specification of distributed algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.39

Related Version *Full Version*: https://hal.archives-ouvertes.fr/hal-03353214

Funding Partly supported by ANR FREDDA (ANR-17-CE40-0013).

Acknowledgements The authors would like to thank the reviewers for their careful reading of the paper and their helpful comments.

1 Introduction

Data logics have been introduced to reason about structures whose elements are labeled with a value from an infinite alphabet (e.g., XML documents) [26]. Expressive decidable fragments include notably two-variable logics over data words and data trees [5, 6]. The decidability frontier is fragile, though. Extensions to two data values, for example, quickly lead to an undecidable satisfiability problem. From a modeling point of view, those extensions still play an important role. When specifying the *input-output behavior* of distributed algorithms [13, 23], processes get an input value and produce an output value, which requires two data values per process. In leader election or renaming algorithms, for instance, a process gets its unique identifier as input, and it should eventually output the identifier of a common leader (leader election) or a unique identifier from a restricted name space (renaming).

In this paper, we consider a natural extension of first-order logic over unordered structures whose elements carry two data values from an infinite domain. There are two major differences between most existing formalisms and our language. While previous data logics are usually interpreted over words or trees, we consider unordered structures (or multisets). When each element of such a structure represents a process, we therefore do not assume a particular processes architecture, but rather consider clouds of computing units. Moreover, decidable data logics are usually limited to one value per element, which would not be sufficient to model an input-output relation. Hence, our models are algebraic structures consisting of a

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 39; pp. 39:1–39:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

universe and functions assigning to each element two integers. We remark that, for many distributed algorithms, the precise data values are not relevant, but whether or not they are the same is. Like [5, 6], we thus add binary relations that allow us to test if two data values are identical and, for example, whether all output values were already present in the collection of input values (as required for leader election).

The first fundamental question that arises is whether a given specification is consistent. This leads us to the satisfiability problem. While the general logic considered here turns out to be undecidable already in several restricted settings, our main result shows that an interesting fragment preserves decidability. The fragment is a *local* logic in the sense that data values can only be compared within the direct neighborhood of a (quantified) reference process. The first value at the reference point can be compared with any second value in the neighborhood in terms of what we call the *diagonal relation.* In this work, we do not allow the symmetrical relation, but we hope we could adapt our technique to this case as well.

However, we do not restrict comparisons of first values with each other in a neighborhood, nor do we restrict comparisons of second values with each other. Note that adding only one diagonal relation still constitutes an extension of the (decidable) two-variable first-order logic with two equivalence relations [18–20]: equivalence classes consist of those elements with the same first value, respectively, second value. In fact, our main technical contribution is a reduction to this two-variable logic. The reduction requires a careful relabelling of the underlying structures so as to be able to express the diagonal relation in terms of the two equivalence relations. In addition, the reduction takes care of the fact that our logic does not restrict the number of variables. We can actually count elements up to some threshold and express, for instance, that at most five processes crash (in the context of distributed algorithms). This is a priori not possible in two-variable logic.

More Related Work. Orthogonal extensions for multiple data values include shuffle expressions for nested data [3] and temporal logics [10, 17]. Other generalizations of data logics allow for an order on data values [24, 27]. The application of formal methods in the context of distributed algorithms is a rather recent but promising approach (cf. for a survey [21]). A particular branch is the area of parameterized systems, which, rather than on data, focuses on the (unbounded) number of processes as the parameter [4, 12]. Other related work includes [11], which considers temporal logics involving quantification over processes but without data, while [1] introduces an (undecidable) variant of propositional dynamic logic that allows one to reason about totally ordered process identifiers in ring architectures. First-order logics for *synthesizing* distributed algorithms were considered in [7, 14]. A counting extension of two-variable first-order logic over finite data words with one data value per position has been studied in [2].

Outline. Section 2 introduces basic notions such as structures and first-order logic, and our local first-order logic and the associated satisfiability problem(s). We identify and solve the decidable case in Section 3. In Section 4, we show that minor extensions of our logic result in undecidability. We conclude in Section 5. Some proof details can be found in the full version of this paper, which is available at: `https://hal.archives-ouvertes.fr/hal-03353214`.

2 Structures and First-Order Logic

2.1 Structures and First-Order Logic

Let Σ be a finite set of unary relation symbols, sometimes called unary predicates. A *data structure* over Σ is a tuple $\mathfrak{A} = (A, f_1, f_2, (P_\sigma)_{\sigma \in \Sigma})$ (in the following, we simply write $(A, f_1, f_2, (P_\sigma)))$ where A is a nonempty finite set, $P_\sigma \subseteq A$ for all $\sigma \in \Sigma$, and f_1 and f_2 are mappings $A \to \mathbb{N}$ assigning a *data value* to each element. We let $\mathit{Val}_{\mathfrak{A}} = \{f_1(a) \mid a \in A\} \cup \{f_2(a) \mid a \in A\}$. The set of all data structures over Σ is denoted by $\mathrm{Data}[\Sigma]$.

While this representation of data structures is often very convenient to refer to the first or second data value of an element, a more standard way of representing mathematical structures is in terms of binary relations. For every $(i, j) \in \{1, 2\} \times \{1, 2\}$, the mappings f_1 and f_2 determine a binary relation ${}_i{\sim}_j^{\mathfrak{A}} \subseteq A \times A$ as follows: $a \;{}_i{\sim}_j^{\mathfrak{A}}\; b$ if $f_i(a) = f_j(b)$. We may omit the superscript $\mathfrak{A}$ if it is clear from the context. This representation is particularly useful when we consider logics as specification languages.

Let $\Gamma \subseteq \{1, 2\} \times \{1, 2\}$ be a set of binary relation symbols, which determines the binary relation symbols ${}_i{\sim}_j$ at our disposal, and let $\mathcal{V} = \{x, y, \ldots\}$ be a countably infinite set of variables. The set $\mathrm{FO}[\Sigma; \Gamma]$ of first-order formulas interpreted over data structures over Σ is inductively given by the grammar $\varphi ::= \sigma(x) \mid x \;{}_i{\sim}_j\; y \mid x = y \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists x.\varphi$, where x and y range over $\mathcal{V}$, σ ranges over Σ, and $(i, j) \in \Gamma$. We use standard abbreviations such as $\wedge$ for conjunction and $\to$ for implication. We write $\varphi(x_1, \ldots, x_n)$ to indicate that the free variables of φ are among $x_1, \ldots, x_n$. A formula without free variables is called a *sentence*.

For $\mathfrak{A} = (A, f_1, f_2, (P_\sigma)) \in \mathrm{Data}[\Sigma]$ and a formula $\varphi \in \mathrm{FO}[\Sigma; \Gamma]$, the satisfaction relation $\mathfrak{A} \models_I \varphi$ is defined wrt. an interpretation function $I : \mathcal{V} \to A$. The purpose of I is to assign an interpretation to every (free) variable of φ so that φ can be given a truth value. For $x \in \mathcal{V}$ and $a \in A$, the interpretation function $I[x/a]$ maps x to a and coincides with I on all other variables. We then define:

$$\begin{array}{ll}
\mathfrak{A} \models_I \sigma(x) \text{ if } I(x) \in P_\sigma & \mathfrak{A} \models_I \varphi_1 \vee \varphi_2 \text{ if } \mathfrak{A} \models_I \varphi_1 \text{ or } \mathfrak{A} \models_I \varphi_2 \\
\mathfrak{A} \models_I x \;{}_i{\sim}_j\; y \text{ if } I(x) \;{}_i{\sim}_j^{\mathfrak{A}}\; I(y) & \mathfrak{A} \models_I \neg\varphi \text{ if } \mathfrak{A} \not\models_I \varphi \\
\mathfrak{A} \models_I x = y \text{ if } I(x) = I(y) & \mathfrak{A} \models_I \exists x.\varphi \text{ if there is } a \in A \text{ with } \mathfrak{A} \models_{I[x/a]} \varphi
\end{array}$$

Finally, for a sentence φ (without free variables), we write $\mathfrak{A} \models \varphi$ if there exists an interpretation function I such that $\mathfrak{A} \models_I \varphi$.

▶ **Example 1.** Assume a unary predicate leader $\in \Sigma$ and $(1, 2) \in \Gamma$. We use the first data value to denote the input of a distributed algorithm and the second data value to denote the output. The following formula from $\mathrm{FO}[\Sigma; \Gamma]$ expresses correctness of a leader-election algorithm: (i) there is a unique process that has been elected leader, and (ii) all processes agree, in terms of their output values, on the identity (the input value) of the leader: $\exists^{=1}x.\mathrm{leader}(x) \wedge \forall y.\exists x.(\mathrm{leader}(x) \wedge x \;{}_1{\sim}_2\; y)$. Here $\exists^{=1}x$ is a shortcut for "there exists exactly one x". Its definition is provided later on. ⌟

Note that every choice of Γ gives rise to a particular logic, whose formulas are interpreted over data structures over Σ. Instead of $\mathrm{FO}[\Sigma; \{(1, 1), (2, 2)\}]$, we may also simply write $\mathrm{FO}[\Sigma; (1, 1), (2, 2)]$ and so on. We will focus on the satisfiability problem for these logics. Let $\mathcal{F}$ denote a generic class of first-order formulas, parameterized by Σ and Γ. In particular, for $\mathcal{F} = \mathrm{FO}$, we have that $\mathcal{F}[\Sigma; \Gamma]$ is the class $\mathrm{FO}[\Sigma; \Gamma]$.

▶ **Definition 2.** *The problem* DataSat$(\mathcal{F}, \Gamma)$ *for* $\mathcal{F}$ *and* Γ *is defined as follows: Given a finite set* Σ *and a sentence* $\varphi \in \mathcal{F}[\Sigma; \Gamma]$*, is there* $\mathfrak{A} \in \mathrm{Data}[\Sigma]$ *such that* $\mathfrak{A} \models \varphi$ *?*

The following negative result, which was shown in [16, Theorem 1], calls for restrictions of the general logic:

▶ **Theorem 3** ([16]). $\textsc{DataSat}(\mathrm{FO}, \{(1,1),(2,2)\})$ *is undecidable, even when requiring that* $\Sigma = \emptyset$.

A Normal Form

When $\Gamma = \emptyset$, satisfiability of monadic first-order logic is decidable [9, Corollary 6.2.2] and the logic actually has a useful normal form. Let $\varphi(x_1, \ldots, x_n, y) \in \mathrm{FO}[\Sigma; \emptyset]$ and $k \geq 1$ be a natural number. We use $\exists^{\geq k} y.\varphi(x_1, \ldots, x_n, y)$ as an abbreviation for $\exists y_1 \ldots \exists y_k. \bigwedge_{1 \leq i < j \leq k} \neg(y_i = y_j) \wedge \bigwedge_{1 \leq i \leq k} \varphi(x_1, \ldots, x_n, y_i)$. Thus, $\exists^{\geq k} y.\varphi$ says that there are at least k distinct elements y that verify φ. We call a formula of the form $\exists^{\geq k} y.\varphi$ a *threshold formula*. We also use $\exists^{=k} y.\varphi$ as an abbreviation for $\exists^{\geq k} y.\varphi \wedge \neg\exists^{\geq k+1} y.\varphi$.

When $\Gamma = \emptyset$, the out-degree of every element is 0 so that, over this particular signature, we deal with structures of bounded degree. The following lemma will turn out to be useful. It is due to Hanf's locality theorem [15, 22] for structures of bounded degree (cf. [8, Theorem 2.4]).

▶ **Lemma 4.** *Every formula from* $\mathrm{FO}[\Sigma; \emptyset]$ *with one free variable* x *is effectively equivalent to a Boolean combination of formulas of the form* $\sigma(x)$ *with* $\sigma \in \Sigma$ *and threshold formulas of the form* $\exists^{\geq k} y.\varphi_U(y)$ *where* $U \subseteq \Sigma$ *and* $\varphi_U(y) = \bigwedge_{\sigma \in U} \sigma(y) \wedge \bigwedge_{\sigma \in \Sigma \setminus U} \neg\sigma(y)$.

Extended Two-Variable First-Order Logic

An orthogonal way to obtain decidability is to restrict to two variables and $\Gamma = \{(1,1),(2,2)\}$. The two-variable fragment $\mathrm{FO}^2[\Sigma; \Gamma]$ contains all $\mathrm{FO}[\Sigma; \Gamma]$ formulas that use only two variables (usually x and y). In a two-variable formula, however, each of the two variables can be used arbitrarily often. The satisfiability problem of two-variable logic over arbitrary finite structures with two equivalence relations is decidable [20, Theorem 15]. By a straightforward reduction to this problem, we obtain:

▶ **Theorem 5** ([20]). *The problem* $\textsc{DataSat}(\mathrm{FO}^2, \{(1,1),(2,2)\})$ *is decidable.*

Actually, this result can be generalized to *extended* two-variable first-order logic. A formula belongs to $\text{ext-FO}^2[\Sigma, \Gamma]$ if it is of the form $\varphi \wedge \psi$ where $\varphi \in \mathrm{FO}[\Sigma; \emptyset]$ and $\psi \in \mathrm{FO}^2[\Sigma, \Gamma]$. To obtain the next result, the idea consists in first translating the formula $\varphi \in \mathrm{FO}[\Sigma; \emptyset]$ to a two-variable formula thanks to new unary predicates.

▶ **Proposition 6.** *The problem* $\textsc{DataSat}(\text{ext-FO}^2, \{(1,1),(2,2)\})$ *is decidable.*

2.2 Local First-Order Logic

We are interested in logics that combine the advantages of the logics considered so far, while preserving decidability. With this in mind, we will study *local* logics, where the scope of quantification is restricted to the neighborhood of a given element.

The neighborhood of an element a includes all elements whose distance to a is bounded by a given radius. It is formalized using the notion of a Gaifman graph (for an introduction, see [22]). In fact, we use a variant that is suitable for our setting and that we call *data graph.* Fix sets Σ and Γ. Given a data structure $\mathfrak{A} = (A, f_1, f_2, (P_\sigma)) \in \mathrm{Data}[\Sigma]$, we define its *data graph* $\mathcal{G}(\mathfrak{A}) = (V_{\mathcal{G}(\mathfrak{A})}, E_{\mathcal{G}(\mathfrak{A})})$ with set of vertices $V_{\mathcal{G}(\mathfrak{A})} = A \times \{1,2\}$ and set of edges $E_{\mathcal{G}(\mathfrak{A})} = \{((a,i),(b,j)) \in V_{\mathcal{G}(\mathfrak{A})} \times V_{\mathcal{G}(\mathfrak{A})} \mid a = b \text{ and } i \neq j, \text{ or } (i,j) \in \Gamma \text{ and } a \;{}_i{\sim}_j\; b\}$. The graph $\mathcal{G}(\mathfrak{A})$ is illustrated in Figure 1.

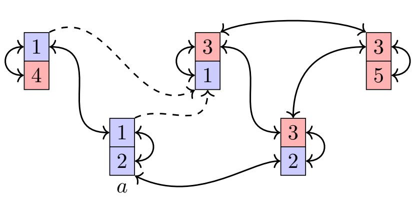


Figure 1 On the left: A data structure $\mathfrak{A}$ and its data graph $\mathcal{G}(\mathfrak{A})$ when $\Gamma = \{(1,1),(2,2),(1,2)\}$. Unidirectional edges are dashed. The blue nodes represent $B_1^{\mathfrak{A}}(a)$. On the right is $\mathfrak{A}|_a^1$.

We define the distance $d^{\mathfrak{A}}((a,i),(b,j)) \in \mathbb{N} \cup \{\infty\}$ between two elements (a,i) and (b,j) from $A \times \{1,2\}$ as the length of the shortest *directed* path from (a,i) to (b,j) in $\mathcal{G}(\mathfrak{A})$. In fact, as the graph is directed, the distance function might not be symmetric. For $a \in A$ and $r \in \mathbb{N}$, the *radius-r-ball around a* is the set $B_r^{\mathfrak{A}}(a) = \{(b,j) \in V_{\mathcal{G}(\mathfrak{A})} \mid d^{\mathfrak{A}}((a,i),(b,j)) \leq r$ for some $i \in \{1,2\}\}$. That is, it contains the elements of $V_{\mathcal{G}(\mathfrak{A})}$ that can be reached from $(a,1)$ or $(a,2)$ through a directed path of length at most r. In the left-hand side of Figure 1, $B_1^{\mathfrak{A}}(a)$ is given by the blue nodes.

Consider an injective mapping $\pi : A \times \{1,2\} \to \mathbb{N} \setminus \mathit{Val}_{\mathfrak{A}}$. We define the *$r$-neighborhood of a in $\mathfrak{A}$* as the structure $\mathfrak{A}|_a^r = (A', f_1', f_2', (P_\sigma')) \in \mathrm{Data}[\Sigma]$. Its universe is $A' = \{b \in A \mid (b,i) \in B_r^{\mathfrak{A}}(a)$ for some $i \in \{1,2\}\}$. Moreover, $f_i'(b) = f_i(b)$ if $(b,i) \in B_r^{\mathfrak{A}}(a)$, and $f_i'(b) = \pi((b,i))$ otherwise. Finally, P_σ' is the restriction of P_σ to A'. To illustrate this definition, we use again Figure 1. The structure $\mathfrak{A}|_a^1$ is given by the four elements that contain at least one blue node. However, the values of the red nodes have to be replaced by pairwise distinct fresh values not contained in $\{1,\ldots,5\}$. Note that the precise values do not matter.

We are now ready to present the logic r-Loc-FO$[\Sigma;\Gamma]$, where $r \in \mathbb{N}$, interpreted over structures from Data$[\Sigma]$. It is given by the grammar

$$\varphi \ ::= \ \langle\!\langle \psi \rangle\!\rangle_x^r \ \mid \ x = y \ \mid \ \exists x.\varphi \ \mid \ \varphi \vee \varphi \ \mid \ \neg\varphi$$

where ψ is a formula from FO$[\Sigma;\Gamma]$ with (at most) one free variable x. For $\mathfrak{A} \in \mathrm{Data}[\Sigma]$ and interpretation function I, we define $\mathfrak{A} \models_I \langle\!\langle \psi \rangle\!\rangle_x^r$ if $\mathfrak{A}|_{I(x)}^r \models_I \psi$.

▶ **Example 7.** We can rewrite the formula from Example 1 so that it falls into the fragment 1-Loc-FO$[\Sigma;(1,1),(2,2),(2,1)]$: $\exists^{=1}x.\langle\!\langle \mathrm{leader}(x) \rangle\!\rangle_x^1 \wedge \forall y.\langle\!\langle \exists x.\mathrm{leader}(x) \wedge y \ {}_2{\sim}_1 \ x \rangle\!\rangle_y^1$. The next formula specifies an algorithm in which all processes suggest a value and then choose a new value among those that have been suggested at least three times: $\forall x.\langle\!\langle \exists^{\geq 3} y.x \ {}_2{\sim}_1 \ y \rangle\!\rangle_x^1$. We can also specify partial renaming, i.e., two output values agree only if their input values are the same: $\forall x.\langle\!\langle \forall y.(x \ {}_2{\sim}_2 \ y \to x \ {}_1{\sim}_1 \ y) \rangle\!\rangle_x^1$. Conversely, $\forall x.\langle\!\langle \forall y.(x \ {}_1{\sim}_1 \ y \to x \ {}_2{\sim}_2 \ y) \rangle\!\rangle_x^1$ specifies partial fusion of equivalences classes. ⌟

3 Decidability With One Diagonal Relation

We will show in this section that DataSat(1-Loc-FO, $\{(1,1),(2,2),(1,2)\}$) (or, symmetrically, DataSat(1-Loc-FO, $\{(1,1),(2,2),(2,1)\}$)) is decidable. To this end, we will give a reduction to DataSat(ext-FO^2, $\{(1,1),(2,2)\}$). The rest of this section is devoted to this reduction.

Henceforth, we fix a finite set Σ as well as $\Gamma = \{(1,1),(2,2),(1,2)\}$ and the *diagonal-free set* $\Gamma_{df} = \{(1,1),(2,2)\}$. Moreover, we let Θ range over arbitrary finite sets such that $\Sigma \subseteq \Theta$ and $\Theta \cap \{\mathsf{eq},\mathsf{ed}\} = \emptyset$, where eq and ed are special unary symbols that are introduced below.

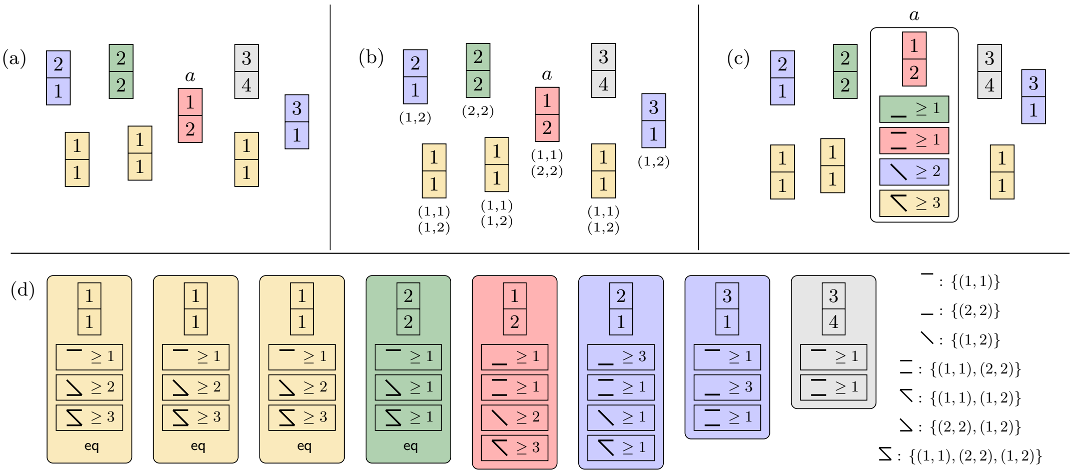


Figure 2 (a) A data structure over $\Sigma = \emptyset$. (b) Adding unary predicates for a given element a. (c) Adding counting constraints to a. (d) A well-typed data structure from $\mathrm{Data}[\{\mathtt{eq}\} \cup \mathtt{C}_3]$.

We start with some crucial notion. Suppose $\Gamma' \subseteq \Gamma$ (which will later be instantiated by either Γ_{df} or Γ). Consider a data structure $\mathfrak{A} = (A, f_1, f_2, (P_\sigma)) \in \mathrm{Data}[\Theta]$ with $\Sigma \subseteq \Theta$. Given $U \subseteq \Sigma$ and a nonempty set $R \subseteq \Gamma'$, the *environment* of $a \in A$ is defined as

$$\mathtt{Env}_{\mathfrak{A},\Sigma,\Gamma'}(a, U, R) = \left\{ b \in A \mid U = \{\sigma \in \Sigma \mid b \in P_\sigma\} \text{ and } R = \{(i,j) \in \Gamma' \mid a \;{}_i\!\!\sim_j^{\mathfrak{A}} b\} \right\}.$$

Thus, it contains the elements that carry exactly the labels from U (relative to Σ) and to which a is related precisely in terms of the relations in R (relative to Γ').

▶ **Example 8.** Consider $\mathfrak{A} \in \mathrm{Data}[\Sigma]$ from Figure 2(a) where $\Sigma = \emptyset$. Then, the set $\mathtt{Env}_{\mathfrak{A},\Sigma,\Gamma}(a, \emptyset, \{(1,1),(1,2)\}) = \mathtt{Env}_{\mathfrak{A},\Sigma,\Gamma_{df}}(a, \emptyset, \{(1,1)\})$ contains exactly the yellow elements (with data-value pairs $(1,1)$), and $\mathtt{Env}_{\mathfrak{A},\Sigma,\Gamma}(a, \emptyset, \{(1,2)\})$ contains the two blue elements (with data-value pairs $(2,1)$ and $(3,1)$). ⌟

Let us now go through the reduction step by step.

Step 1: Transform Binary into Unary Relations

In the first step, we get rid of the binary relations by representing them as unary ones. In fact, in a formula $\langle\!\langle \psi \rangle\!\rangle_x^1$ from 1-Loc-FO$[\Sigma; \Gamma]$, ψ only talks about elements that are directly related to $a = I(x)$ in terms of pairs from Γ. In fact, we can rewrite ψ into ψ' so that all comparisons are wrt. x, i.e., they are of the form $x \;{}_i\!\!\sim_j y$. Then, a pair $(i,j) \in \Gamma$ can be seen as a unary predicate that holds at b iff $a \;{}_i\!\!\sim_j b$. In this way, we eliminate the binary relations and replace ψ' with a first-order formula ψ'' over unary predicates.

▶ **Example 9.** Adding unary relations to a data structure for a given element a is illustrated in Figure 2(b) (recall that $\Sigma = \emptyset$). ⌟

Thanks to the unary predicates, we can now apply Lemma 4 (which was a consequence of locality of first-order logic over unary symbols only). That is, to know whether ψ'' holds when x is interpreted as a, it is enough to know how often every unary predicate is present in the environment of a, counted only up to some $M \geq 1$. However, we will then give up the information of whether the two data values at a coincide or not. Therefore, we introduce a unary predicate $\mathtt{eq}$, which shall label those events whose two data values coincide. Accordingly, we say that $\mathfrak{A} = (A, f_1, f_2, (P_\sigma)) \in \mathrm{Data}[\Theta \cup \{\mathtt{eq}\}]$ is *eq-respecting* if, for all $a \in A$, we have $a \in P_{\mathtt{eq}}$ iff $f_1(a) = f_2(a)$.

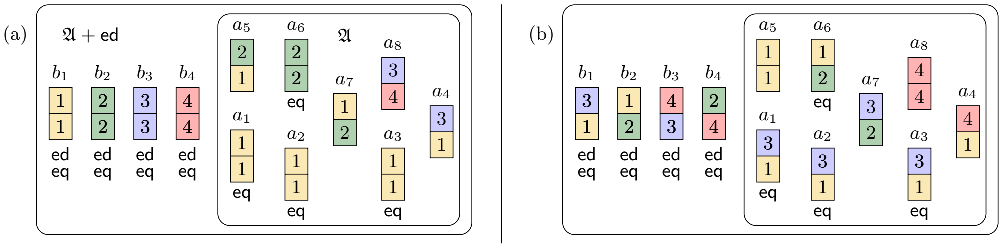


Figure 3 (a) Adding diagonal elements. (a)←(b) Making a data structure eq-respecting.

Once we add this information to a, it is enough to know the size of $\mathtt{Env}_{\mathfrak{A},\Sigma,\Gamma}(a, U, R)$ for every $U \subseteq \Sigma$ and nonempty $R \subseteq \Gamma$, measured up to M. To reason about these sizes, we introduce a unary predicate $\wr U, R, m \wr$ for all $U \subseteq \Sigma$, nonempty sets $R \subseteq \Gamma$, and $m \in \{1, \ldots, M\}$ (which is interpreted as "$\geq m$"). We also call such a predicate a *counting constraint* and denote the set of all counting constraints by $\mathtt{C}_M$ (recall that we fixed Σ and Γ). For a finite set Θ with $\Sigma \subseteq \Theta$, we call $\mathfrak{A} = (A, f_1, f_2, (P_\sigma)) \in \mathrm{Data}[\Theta \cup \mathtt{C}_M]$ *cc-respecting* if, for all $a \in A$, we have $a \in P_{\wr U,R,m \wr}$ iff $|\mathtt{Env}_{\mathfrak{A},\Sigma,\Gamma}(a, U, R)| \geq m$.

Finally, we call $\mathfrak{A} \in \mathrm{Data}[\Theta \cup \{\mathsf{eq}\} \cup \mathtt{C}_M]$ *well-typed* if it is eq-respecting and cc-respecting.

▶ **Example 10.** In Figure 2(c), where we suppose $M = 3$ and $\Sigma = \emptyset$, the element a satisfies the counting constraints $\wr\emptyset, \{(2, 2)\}, 1\wr$, $\wr\emptyset, \{(1, 1), (2, 2)\}, 1\wr$, $\wr\emptyset, \{(1, 2)\}, 2\wr$, and $\wr\emptyset, \{(1, 1), (1, 2)\}, 3\wr$, as well as all inherited constraints for smaller constants (which we omitted). We write $\wr\emptyset, R, m\wr$ as $R \geq m$. In fact, pairs from R are represented as black bars in the obvious way (cf. Figure 2(d)); moreover, for each constraint, the corresponding elements have the same color. Finally, the data structure from Figure 2(d) is well-typed, i.e., eq- and cc-respecting. Again, we omit inherited constraints. ⌟

To summarize, we have the following reduction:

▶ **Lemma 11.** *For each formula $\varphi \in$ 1-Loc-FO$[\Sigma; \Gamma]$, we can effectively compute $M \in \mathbb{N}$ and $\chi \in \mathrm{FO}[\Sigma \cup \{\mathsf{eq}\} \cup \mathtt{C}_M; \emptyset]$ such that φ is satisfiable iff χ has a well-typed model.*

Step 2: Well-Diagonalized Structures

In $\mathtt{C}_M$, we still have the diagonal relation $(1, 2) \in \Gamma$. Our goal is to get rid of it so that we only deal with the diagonal-free set $\Gamma_{\mathit{df}} = \{(1, 1), (2, 2)\}$. The idea is again to extend a given structure $\mathfrak{A}$, but now we add new elements, one for each value $n \in \mathit{Val}_{\mathfrak{A}}$, which we tag with a unary symbol ed and whose two data values are n. Diagonal equality will be ensured through making a detour via these "diagonal" elements (hence the name ed).

Formally, when we start from some $\mathfrak{A} = (A, f_1, f_2, (P_\sigma)) \in \mathrm{Data}[\Theta \cup \{\mathsf{eq}\}]$, the data structure $\mathfrak{A} + \mathsf{ed} \in \mathrm{Data}[\Theta \cup \{\mathsf{eq}, \mathsf{ed}\}]$ is defined as $(A', f'_1, f'_2, (P'_\sigma))$ where $A' = A \uplus \mathit{Val}_{\mathfrak{A}}$, $f'_i(a) = f_i(a)$ for all $a \in A$ and $i \in \{1, 2\}$, $f'_1(a) = f'_2(a) = a$ for all $a \in \mathit{Val}_{\mathfrak{A}}$, $P'_\sigma = P_\sigma$ for all $\sigma \in \Theta \setminus \{\mathsf{eq}\}$, $P'_{\mathsf{ed}} = \mathit{Val}_{\mathfrak{A}}$, and $P'_{\mathsf{eq}} = P_{\mathsf{eq}} \cup \mathit{Val}_{\mathfrak{A}}$.

▶ **Example 12.** The structure $\mathfrak{A} + \mathsf{ed}$ is illustrated in Figure 3(a), with $\Theta = \emptyset$. ⌟

With this, we say that $\mathfrak{B} \in \mathrm{Data}[\Theta \cup \{\mathsf{eq}, \mathsf{ed}\}]$ is *well-diagonalized* if it is of the form $\mathfrak{A} + \mathsf{ed}$ for some *eq-respecting* $\mathfrak{A} \in \mathrm{Data}[\Theta \cup \{\mathsf{eq}\}]$. Note that then $\mathfrak{B}$ is eq-respecting, too.

▶ **Example 13.** The data structure $\mathfrak{A} + \mathsf{ed}$ from Figure 3(a) is well-diagonalized. The one from Figure 3(b) is not well-diagonalized (in particular, it is not eq-respecting). ⌟

We will need a way to ensure that the considered data structures are well-diagonalized. To this end, we introduce the following sentence from $\mathrm{FO}^2[\Theta \cup \{\mathsf{eq}, \mathsf{ed}\}; \Gamma_{df}]$:

$$\begin{aligned}\xi^{\Theta}_{\mathsf{ed}} := \quad & \textstyle\bigwedge_{i\in\{1,2\}} \forall x.\exists y.(\mathsf{ed}(y) \wedge x \;{}_i{\sim}_i\; y) \wedge \big(\forall x.\forall y.(\mathsf{ed}(x) \wedge \mathsf{ed}(y) \wedge x \;{}_i{\sim}_i\; y) \rightarrow x = y\big)\\ & \wedge\ \forall x.\mathsf{eq}(x) \leftrightarrow \exists y.(\mathsf{ed}(y) \wedge x \;{}_1{\sim}_1\; y \wedge x \;{}_2{\sim}_2\; y)\\ & \wedge\ \forall x.\mathsf{ed}(x) \rightarrow \textstyle\bigwedge_{\sigma\in\Theta} \neg\sigma(x)\end{aligned}$$

Every structure that is well-diagonalized satisfies $\xi^{\Theta}_{\mathsf{ed}}$. The converse is not true in general. In particular, a model of $\xi^{\Theta}_{\mathsf{ed}}$ is not necessarily eq-respecting. However, if a structure satisfies a formula $\varphi \in \mathrm{FO}[\Theta \cup \{\mathsf{eq}, \mathsf{ed}\}; \Gamma_{df}]$, then it is possible to perform a permutation on the first (or the second) values of its elements while preserving φ. This allows us to get:

▶ **Lemma 14.** *Let* $\mathfrak{B} \in \mathrm{Data}[\Theta \cup \{\mathsf{eq}, \mathsf{ed}\}]$ *and* $\varphi \in \mathrm{FO}[\Theta \cup \{\mathsf{eq}, \mathsf{ed}\}; \Gamma_{df}]$*. If* $\mathfrak{B} \models \varphi \wedge \xi^{\Theta}_{\mathsf{ed}}$*, then there exists an eq-respecting* $\mathfrak{A} \in \mathrm{Data}[\Theta \cup \{\mathsf{eq}\}]$ *such that* $\mathfrak{A} + \mathsf{ed} \models \varphi$*.*

▶ **Example 15.** Consider Figure 3 and let $\Theta = \emptyset$. The data structure from Figure 3(b) satisfies $\xi^{\Theta}_{\mathsf{ed}}$, though it is not well-diagonalized. Suppose it also satisfies $\varphi \in \mathrm{FO}[\{\mathsf{eq}, \mathsf{ed}\}; \Gamma_{df}]$. By permutation of the first data values, we obtain the well-diagonalized data structure in Figure 3(a). As φ does not talk about the diagonal relation, satisfaction of φ is preserved. ⌟

Finally, we can inductively translate $\varphi \in \mathrm{FO}[\Theta \cup \{\mathsf{eq}\}; \emptyset]$ into a formula $[\![\varphi]\!]_{+\mathsf{ed}} \in \mathrm{FO}[\Theta \cup \{\mathsf{eq}, \mathsf{ed}\}; \emptyset]$ that avoids the extra "diagonal" elements: $[\![\sigma(x)]\!]_{+\mathsf{ed}} = \sigma(x)$, $[\![x = y]\!]_{+\mathsf{ed}} = (x = y)$, $[\![\exists x.\varphi]\!]_{+\mathsf{ed}} = \exists x.(\neg\mathsf{ed}(x) \wedge [\![\varphi]\!]_{+\mathsf{ed}})$, $[\![\varphi \vee \varphi']\!]_{+\mathsf{ed}} = [\![\varphi]\!]_{+\mathsf{ed}} \vee [\![\varphi']\!]_{+\mathsf{ed}}$, and $[\![\neg\varphi]\!]_{+\mathsf{ed}} = \neg[\![\varphi]\!]_{+\mathsf{ed}}$. We immediately obtain:

▶ **Lemma 16.** *Let* $\mathfrak{A} \in \mathrm{Data}[\Theta \cup \{\mathsf{eq}\}]$ *and* $\varphi \in \mathrm{FO}[\Theta \cup \{\mathsf{eq}\}; \emptyset]$ *be a sentence. We have* $\mathfrak{A} \models \varphi$ *iff* $\mathfrak{A} + \mathsf{ed} \models [\![\varphi]\!]_{+\mathsf{ed}}$*.*

Step 3: Getting Rid Of the Diagonal Relation

We will now exploit well-diagonalized data structures to reason about environments relative to Γ in terms of environments relative to Γ_{df}. Recall that Θ ranges over finite sets such that $\Sigma \subseteq \Theta$.

▶ **Lemma 17.** *Let* $\mathfrak{A} = (A, f_1, f_2, (P_\sigma)) \in \mathrm{Data}[\Theta \cup \{\mathsf{eq}\}]$ *be eq-respecting and* $\mathfrak{B} = \mathfrak{A} + \mathsf{ed}$*. Moreover, let* $a \in A$*,* $U \subseteq \Sigma$*, and* $R \subseteq \Gamma$ *be a nonempty set. We have* $\mathtt{Env}_{\mathfrak{A},\Sigma,\Gamma}(a, U, R) =$

$$\begin{cases} \mathtt{Env}_{\mathfrak{B},\Sigma,\Gamma_{df}}(a, U, \Gamma_{df}) \setminus P_{\mathsf{ed}} & \textit{if } a \in P_{\mathsf{eq}} \textit{ and } R = \Gamma & (1)\\ \mathtt{Env}_{\mathfrak{B},\Sigma,\Gamma_{df}}(a, U, \Gamma_{df}) & \textit{if } a \notin P_{\mathsf{eq}} \textit{ and } R = \Gamma_{df} & (2)\\ \mathtt{Env}_{\mathfrak{B},\Sigma,\Gamma_{df}}(a, U, \{(1,1)\}) \cap (P_{\mathsf{eq}} \setminus P_{\mathsf{ed}}) & \textit{if } a \notin P_{\mathsf{eq}} \textit{ and } R = \{(1,1),(1,2)\} & (3)\\ \mathtt{Env}_{\mathfrak{B},\Sigma,\Gamma_{df}}(a, U, \{(2,2)\}) & \textit{if } a \in P_{\mathsf{eq}} \textit{ and } R = \{(2,2),(1,2)\} & (4)\\ \mathtt{Env}_{\mathfrak{B},\Sigma,\Gamma_{df}}(a, U, \{(2,2)\}) \setminus P_{\mathsf{ed}} & \textit{if } a \notin P_{\mathsf{eq}} \textit{ and } R = \{(2,2)\} & (5)\\ \mathtt{Env}_{\mathfrak{B},\Sigma,\Gamma_{df}}(a, U, \{(1,1)\}) \setminus P_{\mathsf{eq}} & \textit{if } \qquad\qquad R = \{(1,1)\} & (6)\\ \mathtt{Env}_{\mathfrak{B},\Sigma,\Gamma_{df}}(d, U, \{(2,2)\}) & \textit{if } a \notin P_{\mathsf{eq}} \textit{ and } R = \{(1,2)\} & (7)\\ \quad \textit{for the unique } d \in P_{\mathsf{ed}} \textit{ such that } d \;{}_1{\sim}_1^{\mathfrak{B}}\; a & & \\ \emptyset & \textit{otherwise} & (8) \end{cases}$$

▶ **Example 18.** Let us go through some cases of Lemma 17 using Figure 3(a), and letting $\Sigma = \Theta = \emptyset$.

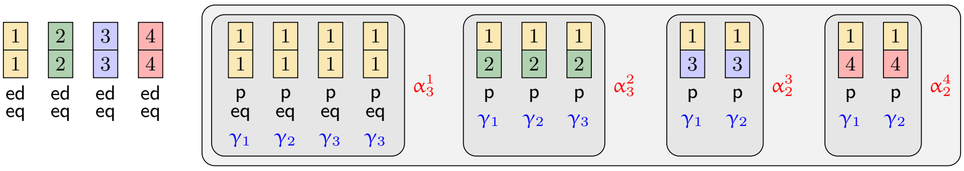


Figure 4 Counting intersections for $M = 3$ and elements with label p.

(1) Let $a = a_1$ and $R = \Gamma$. Then, $\mathtt{Env}_{\mathfrak{A},\Sigma,\Gamma}(a, \emptyset, R) = \{a_1, a_2, a_3\}$. We also have that $\mathtt{Env}_{\mathfrak{B},\Sigma,\Gamma_{df}}(a, \emptyset, \Gamma_{df}) = \{a_1, a_2, a_3, b_1\}$: These are the elements that coincide with a *exactly* on the first and the on the second data value when we dismiss the diagonal relation. Of course, as we consider $\mathfrak{B}$, this includes b_1, which we have to exclude. Thus, $\mathtt{Env}_{\mathfrak{A},\Sigma,\Gamma}(a, \emptyset, R) = \mathtt{Env}_{\mathfrak{B},\Sigma,\Gamma_{df}}(a, \emptyset, \Gamma_{df}) \setminus P_{\mathsf{ed}}$.

(6) Let $a = a_4$ and $R = \{(1, 1)\}$. We have $\mathtt{Env}_{\mathfrak{A},\Sigma,\Gamma}(a, \emptyset, R) = \{a_8\}$. Looking at $\mathfrak{B}$ and discarding the diagonal relation would also include b_3 and any element with data-value pair $(3, 3)$. Discarding P_{eq}, we obtain $\mathtt{Env}_{\mathfrak{B},\Sigma,\Gamma_{df}}(a, \emptyset, \{(1, 1)\}) \setminus P_{\mathsf{eq}} = \{a_8, b_3\} \setminus \{b_3\} = \{a_8\}$.

(7) Let $a = a_7$ and $R = \{(1, 2)\}$. Then, $\mathtt{Env}_{\mathfrak{A},\Sigma,\Gamma}(a, \emptyset, R) = \{a_4, a_5\}$, which is the set of elements whose second data value is 1 and whose first data value is different from 1. The idea is now to change the reference point. Take the unique $d \in P_{\mathsf{ed}}$ such that $d \;{}_1{\sim}_1^{\mathfrak{B}}\; a$. Thus, $d = b_1$. The set $\mathtt{Env}_{\mathfrak{B},\Sigma,\Gamma_{df}}(b_1, \emptyset, \{(2, 2)\})$ gives us exactly the elements that have 1 as the second data value and a first value different from 1, as desired. ⌟

Let us wrap up: By Lemmas 11 and 17, we end up with checking counting constraints in an extended data structure without using the diagonal relation.

Step 4: Counting in Two-Variable Logic

The next step is to express these constraints using two-variable formulas. Counting in two-variable logic is established using further unary predicates. These additional predicates allow us to define a partitioning of the universe of a structure into so-called *intersections*. Suppose $\mathfrak{A} = (A, f_1, f_2, (P_\sigma)) \in \mathrm{Data}[\Theta \cup \{\mathsf{eq}, \mathsf{ed}\}]$, where $\Sigma \subseteq \Theta$. Let $a \in A \setminus P_{\mathsf{ed}}$ and define $\ell_\Sigma(a) = \{\sigma \in \Sigma \mid a \in P_\sigma\}$. The *intersection* of a in $\mathfrak{A}$ is the set $\{b \in A \setminus P_{\mathsf{ed}} \mid a \;{}_1{\sim}_1\; b \wedge a \;{}_2{\sim}_2\; b \wedge \ell_\Sigma(a) = \ell_\Sigma(b)\}$. A set is called an *intersection* in $\mathfrak{A}$ if it is the intersection of some $a \in A \setminus P_{\mathsf{ed}}$.

▶ **Example 19.** Consider Figure 4 and suppose $\Sigma = \{\mathsf{p}\}$. The intersections of the given data structure are gray-shaded. ⌟

Let us introduce the various unary predicates, which will be assigned to *non-diagonal* elements. There are three types of them (for the first two types, also see Figure 4):

1. The unary predicates $\Lambda_M^\gamma = \{\gamma_1, \ldots, \gamma_M\}$ have the following intended meaning: For all intersections I and $i \in \{1, \ldots, M\}$, we have $|I| \geq i$ iff there is $a \in I$ such that $a \in P_{\gamma_i}$. In other words, the presence (or absence) of γ_i in an intersection I tells us whether $|I| \geq i$.
2. The predicates $\Lambda_M^\alpha = \{\alpha_i^j \mid i \in \{1, \ldots, M\}$ and $j \in \{1, \ldots, M + 2\}\}$ have the following meaning: If a is labeled with α_i^j, then (i) there are at least j intersections sharing the same first value and the same label set $\ell_\Sigma(a)$, and (ii) the intersection of a has i elements if $i \leq M - 1$ and at least M elements if $i = M$. Hence, in α_i^j, index i counts the elements inside an intersection, and j labels up to $M + 2$ different intersections. We need to go beyond M due to Lemma 17: When we remove certain elements (e.g., P_{eq}) from an environment, we must be sure to still have sufficiently many to be able to count until M.

3. Labels from $\Lambda_M^{\beta} = \{\beta_i^j \mid i \in \{1,\ldots,M\}$ and $j \in \{1,\ldots,M+1\}\}$ will play a similar role as those in Λ_M^{α} but consider the second values of the elements instead of the first ones.

▶ **Example 20.** A suitable labeling for types γ and α is illustrated in Figure 4 for $M = 3$. $\lrcorner$

Let $\Lambda_M = \Lambda_M^{\alpha} \cup \Lambda_M^{\beta} \cup \Lambda_M^{\gamma}$ denote the set of all these unary predicates. It is relatively standard to come up with sentences $\varphi_\alpha, \varphi_\beta, \varphi_\gamma \in \mathrm{FO}^2[\Theta \cup \{\mathsf{eq},\mathsf{ed}\} \cup \Lambda_M; \Gamma_{df}]$ that guarantee the respective properties. In particular, they make use of the formula $x\ {}_1{\sim}_1\ y \wedge x\ {}_2{\sim}_2\ y \wedge \bigwedge_{\sigma\in\Sigma} \sigma(x) \leftrightarrow \sigma(y)$ saying that two (non-diagonal) elements x and y are in the same intersection.

Now that we can count on a consistent labeling with predicates from Λ_M, let us see how we can exploit it to express $\lbag U, R, m \rbag \in \mathtt{C}_M$, with additional help from Lemma 17, as a formula $\varphi_{U,R,m}(x) \in \mathrm{FO}^2[\Theta \cup \{\mathsf{eq},\mathsf{ed}\} \cup \Lambda_M; \Gamma_{df}]$ applied to *non-diagonal* elements (outside P_{ed}). Let us look at two sample cases according to the case distinction done in Lemma 17. Hereby, we will use, for $U \subseteq \Sigma$, the formula $\varphi_U(y) = \bigwedge_{\sigma\in U} \sigma(y) \wedge \bigwedge_{\sigma\in\Sigma\setminus U} \neg\sigma(y)$.

(1) In this simple case with $R = \{(1,1),(2,2),(1,2)\}$, we need to say that (i) the element a under consideration is in P_{eq}, and (ii) there is an intersection of size at least m (i..e., it contains a γ_m-labeled element) whose elements b satisfy $a\ {}_1{\sim}_1\ b$, $a\ {}_2{\sim}_2\ b$, and $\ell_\Sigma(b) = U$:

$$\varphi_{U,R,m}(x) \ := \ \mathsf{eq}(x) \wedge \exists y.\big(\varphi_U(y) \wedge x\ {}_1{\sim}_1\ y \wedge x\ {}_2{\sim}_2\ y \wedge \gamma_m(y)\big)$$

(6) For $R = \{(1,1)\}$, we first need an extra definition. Given $m \in \{1,\ldots,M\}$, we define the set $\mathcal{S}_{\alpha,m}$ of subsets of Λ_M^{α} as follows: $\mathcal{S}_{\alpha,m} = \{\{\alpha_{i_1}^{j_1},\ldots,\alpha_{i_k}^{j_k}\} \mid i_1 + \ldots + i_k \geq m$ and $j_1 < j_2 < \ldots < j_k\}$. It corresponds to the sets of elements α_i^j whose sum of i is greater than or equal to m. We can then translate the constraint according to Lemma 17 as follows:

$$\varphi_{U,R,m}(x) \ := \bigvee_{S\in\mathcal{S}_{\alpha,m}} \bigwedge_{\alpha\in S} \exists y.\big(\varphi_U(y) \wedge \alpha(y) \wedge \neg\mathsf{eq}(y) \wedge x\ {}_1{\sim}_1\ y \wedge \neg(x\ {}_2{\sim}_2\ y)\big)$$

Finally, it remains to say that all elements are labeled with the suitable counting constraints. So we let $\varphi_{cc} = \forall x.\neg\mathsf{ed}(x) \rightarrow \bigwedge_{\lbag U,R,m\rbag \in \mathtt{C}_M} \lbag U,R,m \rbag(x) \leftrightarrow \varphi_{U,R,m}(x)$.

▶ **Lemma 21.** *Let* $\mathfrak{A} = (A, f_1, f_2, (P_\sigma)) \in \mathrm{Data}[\Sigma \cup \{\mathsf{eq}\} \cup \mathtt{C}_M \cup \Lambda_M]$ *be eq-respecting. If* $\mathfrak{A} + \mathsf{ed} \models \varphi_\alpha \wedge \varphi_\beta \wedge \varphi_\gamma \wedge \varphi_{cc}$*, then* $\mathfrak{A}$ *is cc-respecting.*

Step 5: Putting it All Together

Let $\mathsf{All} = \Sigma \cup \{\mathsf{eq},\mathsf{ed}\} \cup \mathtt{C}_M \cup \Lambda_M$ denote the set of all the unary predicates that we have introduced so far. Recall that, after Step 1, we were left with $M \geq 1$ and a formula $\varphi \in \mathrm{FO}[\Sigma \cup \{\mathsf{eq}\} \cup \mathtt{C}_M; \emptyset]$. The question is now whether φ has a well-typed model (i.e., a model that is eq-respecting and cc-respecting). Altogether, we get the following reduction:

▶ **Proposition 22.** *Let* $\varphi \in \mathrm{FO}[\Sigma \cup \{\mathsf{eq}\} \cup \mathtt{C}_M; \emptyset]$*. Then,* φ *has a well-typed model iff* $\widehat{\varphi} := \llbracket\varphi\rrbracket_{+\mathsf{ed}} \wedge \xi_{\mathsf{ed}}^{\mathsf{All}\setminus\{\mathsf{eq},\mathsf{ed}\}} \wedge \varphi_\alpha \wedge \varphi_\beta \wedge \varphi_\gamma \wedge \varphi_{cc} \in \text{ext-}\mathrm{FO}^2[\mathsf{All}; \Gamma_{df}]$ *is satisfiable.*

Proof. Suppose $\widehat{\varphi}$ is satisfiable. Then, there is $\mathfrak{B} \in \mathrm{Data}[\mathsf{All}]$ such that $\mathfrak{B} \models \widehat{\varphi}$. By Lemma 14, there exists an eq-respecting data structure $\mathfrak{A} \in \mathrm{Data}[\Sigma \cup \{\mathsf{eq}\} \cup \mathtt{C}_M \cup \Lambda_M]$ such that $\mathfrak{A} + \mathsf{ed} \models \llbracket\varphi\rrbracket_{+\mathsf{ed}} \wedge \varphi_\alpha \wedge \varphi_\beta \wedge \varphi_\gamma \wedge \varphi_{cc}$. Using Lemma 21, we deduce that $\mathfrak{A}$ is cc-respecting and, thus, well-typed. Furthermore, by Lemma 16, we have $\mathfrak{A} \models \varphi$. Note that $\mathfrak{A}$ belongs to $\mathrm{Data}[\Sigma \cup \{\mathsf{eq}\} \cup \mathtt{C}_M \cup \Lambda_M]$. However, by removing the unary predicates in Λ_M, we still have a model of φ from $\mathrm{Data}[\Sigma \cup \{\mathsf{eq}\} \cup \mathtt{C}_M]$ as required. Hence, φ has a well-typed model.

Assume now that there exists a well-typed data structure $\mathfrak{A} \in \mathrm{Data}[\Sigma \cup \{\mathtt{eq}\} \cup \mathtt{C}_M]$ such that $\mathfrak{A} \models \varphi$. Using Lemma 16, we have that $\mathfrak{A} + \mathtt{ed} \models [\![\varphi]\!]_{+\mathtt{ed}}$. Furthermore, using the fact that $\mathfrak{A}$ is well-typed, we can add the unary predicates from Λ_M to $\mathfrak{A} + \mathtt{ed}$ to obtain a data structure $\mathfrak{A}'$ in $\mathrm{Data}[\mathsf{All}]$ such that $\mathfrak{A}' \models \varphi_\alpha \wedge \varphi_\beta \wedge \varphi_\gamma \wedge \varphi_{cc}$. Note that $\mathfrak{A}'$ is well-diagonalized. We deduce that $\mathfrak{A}' \models \widehat{\varphi}$. ◀

▶ **Theorem 23.** DataSat(1-Loc-FO, $\{(1,1),(2,2),(1,2)\}$) *is decidable.*

Proof. Let $\psi \in$ 1-Loc-FO$[\Sigma; (1,1),(2,2),(1,2)]$. Using Lemma 11, we can effectively compute $M \in \mathbb{N}$ and $\varphi \in \mathrm{FO}[\Sigma \cup \{\mathtt{eq}\} \cup \mathtt{C}_M; \emptyset]$ such that ψ is satisfiable iff φ has a well-typed model. By Proposition 22, φ has a well-typed model iff $\widehat{\varphi}$ is satisfiable. Since $\widehat{\varphi}$ belongs to ext-FO$^2[\mathsf{All}; \Gamma_{df}]$, we conclude using Proposition 6. ◀

4 Undecidability Results

Let us show that extending the neighborhood radius yields undecidability. We rely on a reduction from the domino problem [9] and use a specific technique presented in [25].

The Tiling Problem

A *domino system* $\mathcal{D}$ is a triple (D, H, V) where D is a finite set of dominoes and $H, V \subseteq D \times D$ are two binary relations. Let $\mathfrak{G}_m$ denote the standard grid on an $m \times m$ torus, i.e., $\mathfrak{G}_m = (G_m, H_m, V_m)$ where H_m and V_m are two binary relations defined as follows: $G_m = \mathbb{Z} \bmod m \times \mathbb{Z} \bmod m$, $H_m = \{((i,j),(i',j)) \mid i' - i \equiv 1 \mod m\}$, and $V_m = \{((i,j),(i,j')) \mid i' - i \equiv 1 \mod m\}$. In the sequel, we will suppose $\mathbb{Z} \bmod m = \{0, \ldots, m-1\}$ using the least positive member to represent residue classes.

A *bi-binary structure* is a triple (A, R_1, R_2) where A is a finite set and R_1, R_2 are subsets of $A \times A$. Domino systems and $\mathfrak{G}_m$ for any m are examples of bi-binary structures. For two bi-binary structures $\mathfrak{G} = (G, H, V)$ and $\mathfrak{G}' = (G', H', V')$, we say that $\mathfrak{G}$ is *homomorphically embeddable* into $\mathfrak{G}'$ if there is a morphism $\pi : \mathfrak{G} \to \mathfrak{G}'$, i.e., a mapping π such that, for all $a, a' \in G$, $(a, a') \in H \Rightarrow (\pi(a), \pi(a')) \in H'$ and $(a, a') \in V \Rightarrow (\pi(a), \pi(a')) \in V'$. For instance, $\mathfrak{G}_{k \cdot m}$ is homomorphically embeddable into $\mathfrak{G}_m$ through reduction mod m. For a domino system $\mathcal{D}$, a *periodic tiling* is a morphism $\tau : \mathfrak{G}_m \to \mathcal{D}$ for some m and we say that $\mathcal{D}$ *admits a periodic tiling* if there exists a periodic tiling of $\mathcal{D}$.

The problem Tiles (or *periodic tiling problem*), which is well known to be undecidable [9], is defined as follows: Given a domino system $\mathcal{D}$, does $\mathcal{D}$ admit a periodic tiling?

To use Tiles in our reductions, we first use some specific bi-binary structures, which we call grid-like and which are easier to manipulate in our context to encode domino systems. A bi-binary structure $\mathfrak{G} = (A, H, V)$ is said to be *grid-like* if some $\mathfrak{G}_m$ is homomorphically embeddable into $\mathfrak{G}$. The logic FO *over bi-binary structures* refers to the first-order logic on two binary relations H, V, and we write $\mathsf{H}xy$ to say that x and y are in relation for H. Consider the two following FO formulas over bi-binary structures: $\varphi_{complete} = \forall x.\forall y.\forall x'.\forall y'.((\mathsf{H}xy \wedge \mathsf{V}xx' \wedge \mathsf{V}yy') \to \mathsf{H}x'y')$ and $\varphi_{progress} = \forall x.(\exists y.\mathsf{H}xy \wedge \exists y.\mathsf{V}xy)$. The following lemma, first stated and proved in [25], shows that these formulas suffice to characterize grid-like structures:

▶ **Lemma 24** ([25]). *Let $\mathfrak{G} = (A, H, V)$ be a bi-binary structure. If $\mathfrak{G}$ satisfies $\varphi_{complete}$ and $\varphi_{progress}$, then $\mathfrak{G}$ is grid-like.*

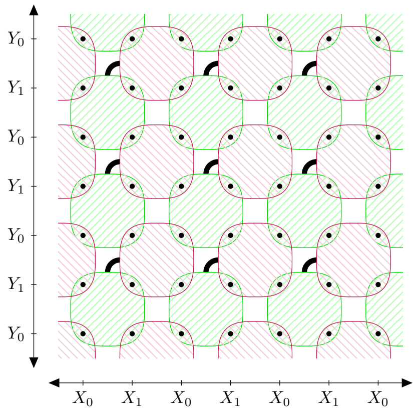


Figure 5 The local pattern of $\mathfrak{A}_{2m}$. Dots denote elements. Two dots are in the same ${}_1{\sim}_1$-equivalence class (resp. ${}_2{\sim}_2$) iff they are in the same green (resp. purple) area. The thick black lines represent the relation ${}_1{\sim}_2$ in the following way: if a ${}_1{\sim}_1$-equivalence class C_1 and a ${}_2{\sim}_2$-equivalence class C_2 are connected with a thick black line, then for any $a \in C_1$ and $b \in C_2$, we have $a \; {}_1{\sim}_2 \; b$.

$$\begin{aligned}
\varphi_H^{00} &= X_0(x) \wedge X_1(y) \wedge Y_0(x) \wedge Y_0(y) \wedge x \; {}_1{\sim}_1 \; y & \varphi_V^{00} &= X_0(x) \wedge X_0(y) \wedge Y_0(x) \wedge Y_1(y) \wedge x \; {}_1{\sim}_1 \; y \\
\varphi_H^{10} &= X_1(x) \wedge X_0(y) \wedge Y_0(x) \wedge Y_0(y) \wedge x \; {}_2{\sim}_2 \; y & \varphi_V^{10} &= X_1(x) \wedge X_1(y) \wedge Y_0(x) \wedge Y_1(y) \wedge x \; {}_1{\sim}_1 \; y \\
\varphi_H^{01} &= X_0(x) \wedge X_1(y) \wedge Y_1(x) \wedge Y_1(y) \wedge x \; {}_1{\sim}_1 \; y & \varphi_V^{01} &= X_0(x) \wedge X_0(y) \wedge Y_1(x) \wedge Y_0(y) \wedge x \; {}_2{\sim}_2 \; y \\
\varphi_H^{11} &= X_1(x) \wedge X_0(y) \wedge Y_1(x) \wedge Y_1(y) \wedge x \; {}_2{\sim}_2 \; y & \varphi_V^{11} &= X_1(x) \wedge X_1(y) \wedge Y_1(x) \wedge Y_0(y) \wedge x \; {}_2{\sim}_2 \; y \\
\varphi_H &= \varphi_H^{00} \vee \varphi_H^{10} \vee \varphi_H^{01} \vee \varphi_H^{11} & \varphi_V &= \varphi_V^{00} \vee \varphi_V^{10} \vee \varphi_V^{01} \vee \varphi_V^{11}
\end{aligned}$$

Figure 6 Link between $\mathfrak{A}_{2m}$ and $\mathfrak{G}_{2m}$.

Given $\mathfrak{A} = (A, f_1, f_2, (P_\sigma)) \in \text{Data}[\Sigma]$ and $\varphi(x, y) \in \text{FO}[\Sigma; \Gamma]$, we define the binary relation $[\![\varphi]\!]_{\mathfrak{A}} = \{(a, b) \in A \times A \mid \mathfrak{A} \models_{I[x/a][y/b]} \varphi(x, y)$ for some interpretation function $I\}$. Thus, given two $\text{FO}[\Sigma; \Gamma]$ formulas $\varphi_1(x, y), \varphi_2(x, y)$ with two free variables, $(A, [\![\varphi_1]\!]_{\mathfrak{A}}, [\![\varphi_2]\!]_{\mathfrak{A}})$ is a bi-binary structure.

As we want to reason on data structures, we build a data structure $\mathfrak{A}_{2m}$ that corresponds to the grid $\mathfrak{G}_{2m} = (G_{2m}, H_{2m}, V_{2m})$. This structure is depicted locally in Figure 5. To define $\mathfrak{A}_{2m}$, we use four unary predicates given by $\Sigma_{grid} = \{X_0, X_1, Y_0, Y_1\}$. They give us access to the coordinate modulo 2. We then define $\mathfrak{A}_{2m} = (G_{2m}, f_1, f_2, (P_\sigma)) \in \text{Data}[\Sigma_{grid}]$ as follows: For $k \in \{0, 1\}$, we have $P_{X_k} = \{(i, j) \in G_{2m} \mid i \equiv k \mod 2\}$ and $P_{Y_k} = \{(i, j) \in G_{2m} \mid j \equiv k \mod 2\}$. For all $i, j \in \{0, \ldots, 2m - 1\}$, we set $f_1(i, j) = ((i/2) \mod m) + m * ((j/2) \mod m)$ (where $/$ stands for the Euclidian division). Finally, for all $i, j \in \{1, \ldots, 2m\}$, set $f_2(i \mod (2m), j \mod (2m)) = f_1(i - 1, j - 1)$.

In Figure 6, we define quantifier free formulas $\varphi_H(x, y)$ and $\varphi_V(x, y)$ from the logic $\text{FO}[\Sigma_{grid}; (1, 1), (2, 2)]$ with two free variable. These formulas allow us to make the link between the data structure $\mathfrak{A}_{2m}$ and the grid $\mathfrak{G}_{2m}$, and we will use them later on to ensure that a data structure has a shape 'similar' to $\mathfrak{A}_{2m}$.

▶ Remark 25. Note that, using the definitions of G_{2m} and of $\mathfrak{A}_{2m}$ we can show that, if $\mathfrak{G}$ is the bi-binary structure $(G_{2m}, [\![\varphi_H]\!]_{\mathfrak{A}_{2m}}, [\![\varphi_V]\!]_{\mathfrak{A}_{2m}})$, then $\mathfrak{G}_{2m} = \mathfrak{G}$.

The Reduction from Radius 3

We first use the previously introduced notions to show that DataSat(3-Loc-FO, $\{(1,1),(2,2)\}$) is undecidable, hence we assume now that $\Gamma = \{(1,1),(2,2)\}$. The first step in our reduction from Tiles consists in defining $\varphi^{\textit{3-loc}}_{\textit{grid}} \in$ 3-Loc-FO$[\Sigma_{grid};(1,1),(2,2)]$ to check that a data structure corresponds to a grid ($\oplus$ stands for exclusive or):

$$\begin{aligned}
\varphi^{\textit{3-loc}}_{\textit{complete}} &= \forall x.\langle\!\langle \forall y.\forall x'.\forall y'.\varphi_H(x,y) \wedge \varphi_V(x,x') \wedge \varphi_V(y,y') \rightarrow \varphi_H(x',y')\rangle\!\rangle^3_x \\
\varphi^{\textit{3-loc}}_{\textit{progress}} &= \forall x.\langle\!\langle \exists y.\varphi_H(x,y) \wedge \exists y.\varphi_V(x,y)\rangle\!\rangle^3_x \\
\varphi^{\textit{3-loc}}_{\textit{grid}} &= \varphi^{\textit{3-loc}}_{\textit{complete}} \wedge \varphi^{\textit{3-loc}}_{\textit{progress}} \wedge \forall x.\langle\!\langle (X_0(x) \oplus X_1(x)) \wedge (Y_0(x) \oplus Y_1(x))\rangle\!\rangle^3_x
\end{aligned}$$

▶ **Lemma 26.** *We have $\mathfrak{A}_{2m} \models \varphi^{\textit{3-loc}}_{\textit{grid}}$. Moreover, for all $\mathfrak{A} = (A, f_1, f_2, (P_\sigma))$ in* Data$[\Sigma_{grid}]$*, if $\mathfrak{A} \models \varphi^{\textit{3-loc}}_{\textit{grid}}$, then $(A, [\![\varphi_H]\!]_{\mathfrak{A}}, [\![\varphi_V]\!]_{\mathfrak{A}})$ is grid-like.*

Given a domino system $\mathcal{D} = (D, H_{\mathcal{D}}, V_{\mathcal{D}})$, we now provide a formula $\varphi_{\mathcal{D}}$ from the logic 3-Loc-FO$[D;(1,1),(2,2)]$ that guarantees that, if a data structure corresponding to a grid satisfies $\varphi_{\mathcal{D}}$, then it can be embedded into $\mathcal{D}$:

$$\begin{aligned}
\varphi_{\mathcal{D}} \;:=\; & \forall x.\langle\!\langle \textstyle\bigvee_{d\in D}\Big(d(x) \wedge \bigwedge_{d\neq d'\in D} \neg(d(x) \wedge d'(x))\Big)\rangle\!\rangle^3_x \\
& \wedge \forall x.\langle\!\langle \forall y.\varphi_H(x,y) \rightarrow \textstyle\bigvee_{(d,d')\in H_{\mathcal{D}}} d(x) \wedge d'(y)\rangle\!\rangle^3_x \\
& \wedge \forall x.\langle\!\langle \forall y.\varphi_V(x,y) \rightarrow \textstyle\bigvee_{(d,d')\in V_{\mathcal{D}}} d(x) \wedge d'(y)\rangle\!\rangle^3_x
\end{aligned}$$

▶ **Proposition 27.** *Given $\mathcal{D} = (D, H_{\mathcal{D}}, V_{\mathcal{D}})$ a domino system, $\mathcal{D}$ admits a periodic tiling iff the* 3-Loc-FO$[\Sigma_{grid} \uplus D;(1,1),(2,2)]$ *formula $\varphi^{\textit{3-loc}}_{\textit{grid}} \wedge \varphi_{\mathcal{D}}$ is satisfiable.*

As a corollary of the proposition, we obtain the main result of this section.

▶ **Theorem 28.** DataSat(3-Loc-FO, $\{(1,1),(2,2)\}$) *is undecidable.*

We can also reduce Tiles to DataSat(2-Loc-FO, $\{(1,1),(2,2),(1,2)\}$). In that case, it is a bit more subtle to build a formula similar to the formula $\varphi_{complete}$ as we have only neighborhood of radius 2, but we use the diagonal binary relation $(1,2)$ to overcome this.

▶ **Theorem 29.** DataSat(2-Loc-FO, $\{(1,1),(2,2),(1,2)\}$) *is undecidable.*

5 Future Work

There are some interesting open questions. For example, we leave open whether our main decidability result holds for two diagonal relations. Recall that, when comparing the expressiveness, two-variable first-order logic can be embedded in our logic. We do not know yet whether the converse holds. Until now our work has focused on the satisfiability problem. Another next step would be to see how our logic can be used to verify practical distributed algorithms.

References

1 C. Aiswarya, B. Bollig, and P. Gastin. An automata-theoretic approach to the verification of distributed algorithms. *Inf. Comput.*, 259(Part 3):305–327, 2018.

2 B. Bednarczyk and P. Witkowski. A Note on C^2 Interpreted over Finite Data-Words. In *27th International Symposium on Temporal Representation and Reasoning, TIME 2020, September 23-25, 2020, Bozen-Bolzano, Italy*, volume 178 of *LIPIcs*, pages 17:1–17:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.TIME.2020.17.

3 H. Björklund and M. Bojanczyk. Shuffle expressions and words with nested data. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 750–761. Springer, 2007.

4 R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification.* Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.

5 M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011.

6 M. Bojanczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3), 2009.

7 B. Bollig, P. Bouyer, and F. Reiter. Identifiers in registers - describing network algorithms with logic. In *FOSSACS'19*, volume 11425 of *LNCS*, pages 115–132. Springer, 2019.

8 B. Bollig and D. Kuske. An optimal construction of hanf sentences. *J. Appl. Log.*, 10(2):179–186, 2012. doi:10.1016/j.jal.2012.01.002.

9 E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer, 1997.

10 N. Decker, P. Habermehl, M. Leucker, and D. Thoma. Ordered navigation on multi-attributed data words. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 497–511. Springer, 2014.

11 E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.

12 J. Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *STACS'14)*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.

13 W. Fokkink. *Distributed Algorithms: An Intuitive Approach.* MIT Press, 2013.

14 S. Grumbach and Z. Wu. Logical locality entails frugal distributed computation over graphs (extended abstract). In *WG'09*, volume 5911 of *LNCS*, pages 154–165. Springer, 2009.

15 W. Hanf. Model-theoretic methods in the study of elementary logic. In J.W. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*, pages 132–145. North Holland, 1965.

16 A. Janiczak. Undecidability of some simple formalized theories. *Fundamenta Mathematicae*, 40:131–139, 1953.

17 A. Kara, T. Schwentick, and T. Zeume. Temporal logics on words with multiple data values. 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 481–492. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.

18 E. Kieronski. Results on the guarded fragment with equivalence or transitive relations. In C.-H. Luke Ong, editor, *CSL'05*, volume 3634 of *LNCS*, pages 309–324. Springer, 2005.

19 E. Kieronski, J. Michaliszyn, I. Pratt-Hartmann, and L. Tendera. Two-variable first-order logic with equivalence closure. In *LICS'12*, pages 431–440. IEEE, 2012.

20 E. Kieronski and L. Tendera. On finite satisfiability of two-variable first-order logic with equivalence relations. In *LICS'09*, pages 123–132. IEEE, 2009.

21 I. V. Konnov, H. Veith, and J. Widder. What you always wanted to know about model checking of fault-tolerant distributed algorithms. In *PSI'15 in Memory of Helmut Veith*, volume 9609 of *LNCS*, pages 6–21. Springer, 2015.

22 L. Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

23 N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., 1996.

24 A. Manuel and T. Zeume. Two-variable logic on 2-dimensional structures. 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 484–499. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

25 M. Otto. Two-variable first-order logic over ordered domains. *Journal of Symbolic Logic*, 66:685–702, 2001.

26 L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL'06*, volume 4207 of *LNCS*, pages 41–57. Springer, 2006.

27 T. Tan. Extending two-variable logic on data trees with order on data values and its automata. *ACM Trans. Comput. Log.*, 15(1):8:1–8:39, 2014.

Diagrammatic Polyhedral Algebra

Filippo Bonchi
University of Pisa, Italy

Alessandro Di Giorgio
University of Pisa, Italy

Paweł Sobociński
Tallinn University of Technology, Estonia

Abstract

We extend the theory of Interacting Hopf algebras with an order primitive, and give a sound and complete axiomatisation of the prop of polyhedral cones. Next, we axiomatise an affine extension and prove soundness and completeness for the prop of polyhedra.

2012 ACM Subject Classification Theory of computation → Categorical semantics; Theory of computation → Concurrency

Keywords and phrases String diagrams, Polyhedral cones, Polyhedra

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.40

Related Version *Full Version*: `https://arxiv.org/abs/2105.10946` [5]

Funding *Filippo Bonchi*: Supported by the Ministero dell'Università e della Ricerca of Italy under Grant No. 201784YSZ5, PRIN2017 – ASPRA (Analysis of Program Analyses).
Alessandro Di Giorgio: Supported by the Ministero dell'Università e della Ricerca of Italy under Grant No. 201784YSZ5, PRIN2017 – ASPRA (Analysis of Program Analyses).
Paweł Sobociński: Supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001) and the Estonian Research Council grant PRG1210.

Acknowledgements The authors have benefited, at the early stage of this work, of enlightening comments and exciting discussions with Guillaume Boisseau. In particular, Guillaume proposed several simplifications to the axiomatisations, showed us the first proof of the generalised spider and the one for its polar.

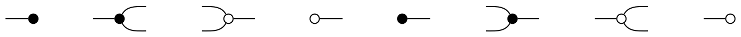
1 Introduction

Engineers and scientists of different fields often rely on diagrammatic notations to model systems of various sorts but, to perform a rigorous analysis, diagrams usually need to be translated to more traditional mathematical language. Indeed diagrams have the advantage to be quite intuitive, highlight connectivity, distribution and communication topology of systems but they usually have an informal meaning and, even when equipped with a formal semantics, diagrams cannot be easily manipulated like standard mathematical expressions. *Compositional network theory* [3, 24] is a multidisciplinary research program studying diagrams as first class citizens: diagrammatic languages come equipped with a formal semantics, which has the key feature to be compositional; moreover diagrams can be manipulated like ordinary symbolic expressions if an appropriate equational theory–ideally characterising semantic equality–can be identified. This approach has been shown effective in various settings like for instance, digital [20] and electrical circuits [4, 24], quantum protocols [14, 15], linear dynamical systems [2, 32], Petri nets [7], Bayesian networks [23] and query languages [19, 10].

The common technical infrastructure is provided by *string diagrams* [31]: arrows of a symmetric monoidal category freely generated by a monoidal signature. Intuitively, the signature is a set of generators and diagrams are simply obtained by composing in series

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 40; pp. 40:1–40:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(horizontally) and in parallel (vertically) generators plus some basic wires (—— and ⨯). The following set of generators is common to (most of) the aforementioned systems and, surprisingly enough, (almost) the same algebraic laws hold in the various settings.

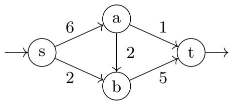


It is convenient to give an intuition of the intended meaning of such generators by relying on the semantics from [32] that, amongst the aforementioned works, is the most relevant for the present paper: the copier —●< receives one value on the left and emits two copies on the right; the discharger —● receives one value on the left and nothing on the right; the adder >○— receives two values on the left and emits their sum on the right; the zero ○— receives nothing on the left and constantly emits 0 on the right. The behaviour of the remaining four generators is the same but left and right are swapped. Here values are meant to be rational numbers. To deal with values from an arbitrary fields k, one has to add a generator —$\boxed{k}$— for each $k \in \mathsf{k}$; its intended meaning is the one of an amplifier: the value received on the left is multiplied by k and emitted on the right.

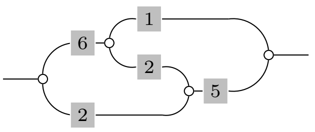
This semantics has two crucial properties: first, it enjoys a sound and complete axiomatisation called the theory of Interacting Hopf Algebras ($\mathbb{IH}$); second, it can express exactly *linear relations*, namely relations forming vector spaces over k. In other words, diagrams modulo the laws of $\mathbb{IH}$ are in one to one correspondence with linear relations.

In this paper we extend $\mathbb{IH}$ in order to express exactly relations that are *polyhedra*, rather than mere vector spaces. Indeed, polyhedra allow the modeling of bounded spaces which are ubiquitous in computer science. For instance, in abstract interpretation [17] polyhedra represent bounded sets of possible values of variables; in concurrency theory and linear optimisation one always deals with systems having a bounded amounts of resources.

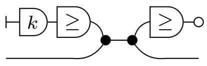
To catch a glimpse of our result, consider the flow network [1] in (1): edges are labeled with a positive real number representing their maximum capacity; the flow enters in the source (the node s) and exits from the sink (the node t).



(1)



(2)



(3)

The network in (1) is represented within our diagrammatic language as in (2) where —$\boxed{k}$— is syntactic sugar for the diagram in (3). Here —$\boxed{\geq}$— and ⊢— are the two novel generators that we need to add to Interacting Hopf Algebras to express exactly polyhedra: —$\boxed{\geq}$— constrains the observation on the left to be greater or equal to the one on the right; ⊢— constantly emits 1 on the right. Observe that (3) forces the values on the left and on the right to be equal and to be in the interval $[0, k]$; the use of >○— and —○< for the nodes forces the sum of the flows entering on the left to be equal to the sum of the flows leaving from the right.

An important property of flow networks is the maximum flow that can enter in the source and arrive to the the sink. The sound and complete axiomatisation that we introduce allows to compute their maximum flow by mean of intuitive graphical manipulations: for instance, the diagram in (2) can be transformed in —$\boxed{5}$—, meaning that its maximum flow is exactly 5. We will come back to flow networks at the end of §5 (Example 31).

The remainder of the paper is organised as follows. We recall the basic categorical tools for string diagrams in §2 and the theory of Interacting Hopf Algebras in §3. In §4, we extend the syntax of Interacting Hopf Algebras with the generator —$\boxed{\geq}$—. On the semantic side, this allows to move from linear relations to *polyhedral cones*, for which we give a sound and fully complete axiomatisation in terms of the diagrammatic syntax. The proof of completeness involves a diagrammatic account of Fourier-Motzkin elimination, two normal forms leading to the Weyl-Minkowski theorem, and a simple, inductive account of the notion of polar cone.

The results in §4 represent our main technical effort. Indeed, to pass from polyhedral cones to polyhedra in §5, it is enough to add the generator ⊢—, originally introduced in [9] to move from linear to affine relations, and one extra axiom. The proof substantially exploits the homogenization technique to reduce completeness for polyhedra to the just proved completeness for polyhedral cones.

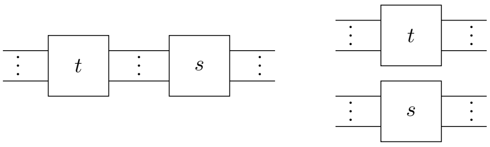
Finally, in §6, we conclude by showing a stateful extension of our diagrammatic calculus. By simply adding a register —$\boxed{x}$— we obtain a complete axiomatisation for stateful polyhedral processes: these are exactly all transition systems where both states and labels are vectors from some vector spaces and the underlying transition relation forms a polyhedron. Stateful polyhedral processes seem to be a sweet spot in terms of expressivity: on the one hand, they properly generalise signal flow graphs [29], on the other, as illustrated in §6, they allow us to give a compositional account of continuous Petri nets [18].

2 Props and Symmetric Monoidal Theories

The diagrammatic languages studied in network theory, e.g., [16, 30, 21, 7], can be treated formally using the category theoretic notion of prop [28, 26] (product and permutation category). A *prop* is a symmetric strict monoidal (ssm) category with objects natural numbers, where the monoidal product $\oplus$ on objects is addition. Morphisms between props are ssm functors that act as identity on objects. The usual methodology is to use two props: Syn, the arrows of which are the diagrammatic terms of the language, and Sem, the arrows of which are the intended semantics. A morphism $[\![\cdot]\!] : \mathsf{Syn} \to \mathsf{Sem}$ assigns semantics to terms, with the functoriality of $[\![\cdot]\!]$ guaranteeing compositionality.

The syntactic prop Syn is usually freely generated from a *monoidal signature* Σ, namely a set of generators $o\colon n \to m$ with arity $n \in \mathbb{N}$ and coarity $m \in \mathbb{N}$. Intuitively, the arrows of Syn are diagrams wired up from the generators. A way of giving a concrete description is via Σ-terms. The set of Σ-*terms* is obtained by composing generators in Σ, the identities $id_0\colon 0 \to 0$, $id_1\colon 1 \to 1$ and the symmetry $\sigma_{1,1}\colon 2 \to 2$ with ; and $\oplus$. This is a purely formal process: given Σ-terms $t\colon k \to l$, $u\colon l \to m$, $v\colon m \to n$, one constructs Σ-terms $t;u\colon k \to m$ and $t \oplus v\colon k + n \to l + n$. Now, the *prop freely generated by a signature* Σ, hereafter denoted by P_Σ, has as its arrows $n \to m$ the set of Σ-terms $n \to m$ modulo the laws of ssm categories.

There is a well-known, natural graphical representation for arrows of a freely generated prop as string diagrams, which we now sketch. A Σ-term $n \to m$ is pictured as a box with n ordered wires on the left and m on the right. Composition via ; and $\oplus$ are rendered graphically by horizontal and vertical juxtaposition of boxes, respectively.



(4)

Moreover $id_1\colon 1 \to 1$ is pictured as —, the symmetry $\sigma_{1,1}\colon 1 + 1 \to 1 + 1$ as ⤫, and the unit object for $\oplus$, that is, $id_0\colon 0 \to 0$ as the empty diagram ⬚. Arbitrary identities id_n and symmetries $\sigma_{n,m}$ are generated according to (4) and drawn as $\overset{n}{\text{—}}$ and ${}^{n}_{m}⤫^{m}_{n}$, respectively.

Given a diagrammatic language Syn and a morphism $[\![\cdot]\!] : \mathsf{Syn} \to \mathsf{Sem}$, a useful task is to identify a sound and (ideally) complete set of characterising equations E: $[\![c]\!] = [\![d]\!]$ iff c and d are equal in $\overset{E}{=}$, the smallest congruence (w.r.t. ; and $\oplus$) containing E. Formally, the set E consists of pairs $(t, t' \colon n \to m)$ of Σ-terms with the same arity and coarity. Then Σ together with E form a *symmetric monoidal theory* (smt), providing a calculus of diagrammatic reasoning. Any smt (Σ, E) yields a prop $\mathsf{P}_{\Sigma,E}$, obtained by quotienting the P_Σ by $\overset{E}{=}$.

Another issue is expressivity: one would like to characterise the image of Syn through $[\![\cdot]\!]$, namely a subprop IM of Sem consisting of exactly those arrows d of Sem for which there exists some c in Syn such that $[\![c]\!] = d$. When this is possible and a sound and complete axiomatization is available, the semantics map $[\![\cdot]\!]$ factors as follows:

$$\begin{tikzcd}
\mathsf{Syn} = \mathsf{P}_\Sigma \arrow[r, two heads, "q"] \arrow[rrr, bend left, "{[\![\cdot]\!]}"] & \mathsf{P}_{\Sigma,E} \arrow[r, "\cong"] & \mathsf{IM} \arrow[r, tail, "\iota"] & \mathsf{Sem} \, .
\end{tikzcd}$$

The morphism q quotients P_Σ by $\overset{E}{=}$, ι is the inclusion of IM in Sem and $\cong$ is an iso between $\mathsf{P}_{\Sigma,E}$ and IM. In this case we say that (Σ, E) is the (symmetric monoidal) theory of IM.

Let k be an ordered field. In this paper Sem is fixed to be the following prop.

▶ **Definition 1.** Rel_k *is the prop where arrows* $n \to m$ *are relations* $R \subseteq \mathsf{k}^n \times \mathsf{k}^m$*.*

- *Composition is relational: given* $R \colon n \to m$ *and* $S \colon m \to o$*,*

$$R \,;\, S = \{\, (u, v) \in \mathsf{k}^n \times \mathsf{k}^o \;\mid\; \exists w \in \mathsf{k}^m.\ (u, w) \in S \wedge (w, v) \in R \,\}$$

- *The monoidal product is cartesian product: given* $R \colon n \to m$ *and* $S \colon o \to p$*,*

$$R \oplus S = \{\, (\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}, \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}) \in \mathsf{k}^{n+o} \times \mathsf{k}^{m+p} \;\mid\; (u_1, v_1) \in R \wedge (v_1, v_2) \in S \,\}$$

- *The symmetries* $\sigma_{n,m} \colon n + m \to m + n$ *are the relations* $\{\, (\begin{pmatrix} u \\ v \end{pmatrix}, \begin{pmatrix} v \\ u \end{pmatrix}) \;\mid\; u \in \mathsf{k}^n, v \in \mathsf{k}^m \,\}$

For IM, we will consider the following three props.

▶ **Definition 2.** *We define three sub-props of* Rel_k*. Arrows* $n \to m$

- *in* $\mathsf{LinRel}_\mathsf{k}$ *are vector spaces* $\{(x, y) \in \mathsf{k}^n \times \mathsf{k}^m \mid A \begin{pmatrix} x \\ y \end{pmatrix} = 0\}$ *for some matrix* A*;*
- *in* PC_k *are polyhedral cones* $\{(x, y) \in \mathsf{k}^n \times \mathsf{k}^m \mid A \begin{pmatrix} x \\ y \end{pmatrix} \geq 0\}$ *for some matrix* A*;*
- *in* P_k *are polyhedra* $\{(x, y) \in \mathsf{k}^n \times \mathsf{k}^m \mid A \begin{pmatrix} x \\ y \end{pmatrix} + b \geq 0\}$ *for some matrix* A *and* $b \in \mathsf{k}^p$*.*

Identities, permutations, composition and monoidal product are defined as in Rel_k*.*

▶ Remark 3. In Definition 2, A is a matrix with $n + m$ columns and p rows, for some $p \in \mathbb{N}$. Observe that the matrix A gives rise also to arrows $n' \to m'$ with n', m' different from n, m but, such that $n' + m' = n + m$. This is justified by the isomorphism of $\mathsf{k}^n \times \mathsf{k}^m$ and $\mathsf{k}^{n'} \times \mathsf{k}^{m'}$. Note therefore that the left and the right boundaries should not be confused with inputs and outputs. This is a common feature in diagrammatic approaches relying on a notion of relational composition which is unbiased.

Showing that the above are well-defined – e.g. that the composition of polyhedral cones is a polyhedral cone – requires some well-known results, which are given in [5, Appendix A]. In §3, we recall the theory of $\mathsf{LinRel}_\mathsf{k}$, in §4 we identify the theory of PC_k and, in §5, that of P_k.

2.1 Ordered Props and Symmetric Monoidal Inequality Theories

As relations $R, S\colon n \to m$ in $\mathsf{Rel_k}$ carry the partial order of inclusion $\subseteq$, it is useful to be able to state when $\llbracket c \rrbracket \subseteq \llbracket d \rrbracket$ for some c, d in Syn (see e.g. [6] for motivating examples). In order to consider such inclusions, it is convenient to look at $\mathsf{Rel_k}$ as an ordered prop.

▶ **Definition 4.** *An* ordered prop *is a prop enriched over the category of posets: a symmetric strict monoidal 2-category with objects the natural numbers, monoidal product on objects given by addition, where each set of arrows $n \to m$ is a poset, with composition and monoidal product monotonic. Similarly, a* pre-ordered prop *is a prop enriched over the category of pre-orders. A morphism of (pre-)ordered props is an identity-on-objects symmetric strict 2-functor.*

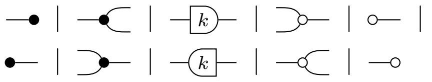
Just as SMTs yield props, *Symmetric Monoidal Inequalities Theories* [6] (SMITs) give rise to ordered props. A SMIT is a pair (Σ, I) where Σ is a signature and I is a set of inequations: as for equations, the underlying data is a pair $(t, t'\colon n \to m)$ of Σ-terms with the same arity and coarity. Unlike equations, however, we understand this data as directed: $t \leq t'$.

To obtain the free ordered prop from an SMIT, first, we construct the free pre-ordered prop: arrows are Σ-terms. The homset orders, hereafter denoted by $\overset{I}{\subseteq}$, are determined by closing I by reflexivity, transitivity, ; and $\oplus$: this is the smallest precongruence (w.r.t. ; and $\oplus$) containing I. Then, we obtain the free ordered prop by quotienting the free pre-ordered prop by the equivalence induced by $\overset{I}{\subseteq}$, i.e. quotienting wrt anti-symmetry.

Any prop can be regarded as an ordered prop with the discrete ordering. Moreover any SMT (Σ, E) gives rise to a canonical SMIT (Σ, I) where each equation is replaced by two inequalities $I = E \cup E^{op}$ in the obvious way. In the remainder of this paper, we always consider SMITs but, for the sake of readability, we generically refer to them just as *theories*. Such theories consist of both inequalities and equations, that are generically called *axioms*. Similarly, all props considered in the paper, their morphisms and isomorphisms are ordered.

3 The theory of Linear relations

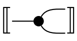
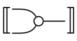
In this section, we recall from [11, 2, 32] the theory of Interacting Hopf algebras. The signature consists of the following set of generators, where k ranges over a fixed field k.

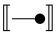
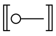
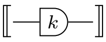


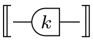
(5)

(6)

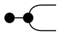
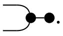
For each generator, its arity and coarity are given by the number of dangling wires on the left and, respectively, on the right. For instance has arity 1 and coarity 0. We call Circ the prop freely generated by this signature and we refer to its arrows as circuits. We use Circ as the *syntax* of our starting diagrammatic language. The semantics is given as the prop morphism $\llbracket\cdot\rrbracket : \mathsf{Circ} \to \mathsf{Rel_k}$ defined for the generators in (5) as

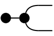
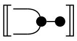
$\llbracket$  $\rrbracket$ $= \{(x, \begin{pmatrix} x \\ x \end{pmatrix}) \mid x \in \mathsf{k}\}$ $\llbracket$  $\rrbracket$ $= \{(\begin{pmatrix} x \\ y \end{pmatrix}, x+y) \mid x, y \in \mathsf{k}\}$ (7)

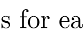
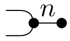
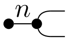
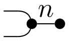
$\llbracket$  $\rrbracket$ $= \{(x, \bullet) \mid x \in \mathsf{k}\}$ $\llbracket$  $\rrbracket$ $= \{(\bullet, 0)\}$ $\llbracket$  $\rrbracket$ $= \{(x, k\cdot x) \mid x \in \mathsf{k}\}$

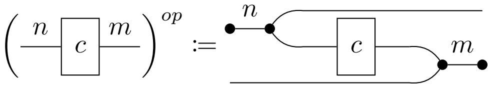
and, symmetrically, for the generators in (6). For instance, $\llbracket$  $\rrbracket$ $= \{(k \cdot x, x) \mid x \in \mathsf{k}\}$. The semantics of the identities, symmetries and compositions is given by the *functoriality* of $\llbracket\cdot\rrbracket$, e.g., $\llbracket c\,;d \rrbracket = \llbracket c \rrbracket \,;\, \llbracket d \rrbracket$. Above we used $\bullet$ for the unique element of the vector space k^0.

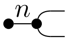
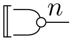
We call $\overrightarrow{\mathsf{Circ}}$ the prop freely generated from the generators in (5) and $\overleftarrow{\mathsf{Circ}}$ the one freely generated from (6). The semantics of circuits in $\overrightarrow{\mathsf{Circ}}$ can be thought of as functions taking inputs on left ports and giving output on the right ports, with the intuition for the generators as given in the Introduction. Symmetrically, the semantics of circuits in $\overleftarrow{\mathsf{Circ}}$ are functions with inputs on the right ports and outputs on the left. The semantics of an arbitrary circuit in Circ is, in general, a relation.

▶ **Example 5.** Two circuits will play a special role in our exposition:  and . Using the definition of $[\![\cdot]\!]$, it is immediate to see that their semantics forces the two ports on the right (resp. left) to carry the same value.

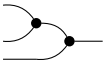
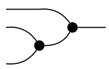
$[\![$  $]\!] = \{(\bullet, \begin{pmatrix} x \\ x \end{pmatrix}) \mid x \in \mathsf{k}\}$ $[\![$  $]\!] = \{(\begin{pmatrix} x \\ x \end{pmatrix}, \bullet) \mid x \in \mathsf{k}\}$

Using these diagrams (along with and) one defines for each $n \in \mathbb{N}$, $: 0 \to n+n$ and $: n+n \to 0$ with semantics $\{(\begin{pmatrix} x \\ x \end{pmatrix}, \bullet) \mid x \in \mathsf{k}^n\}$ and $\{(\bullet, \begin{pmatrix} x \\ x \end{pmatrix}) \mid x \in \mathsf{k}^n\}$. These circuits give rise, modulo the axioms that we will illustrate later, to a self-dual compact closed structure. See [13, Sec.5.1] for full details. As for identities and symmetries, also for  and  we will sometimes omit n for readability. Given an arbitrary circuit $c\colon n \to m$, its *opposite* circuit $c^{op}\colon m \to n$ is defined as illustrated below. It is easy to see that c^{op} denotes the opposite relation of $[\![c]\!]$, i.e., $[\![c^{op}]\!] = \{(y, x) \in \mathsf{k}^m \times \mathsf{k}^n \mid (x, y) \in [\![c]\!]\}$.



As for  above, one can define the n-version of each of the generators in (5) and (6) (as well as generators (8) and (10) that we shall introduce later). For instance $[\![$  $]\!] = \{(\begin{pmatrix} x \\ y \end{pmatrix}, x+y) \mid x, y \in \mathsf{k}^n\}$. When clear from the context, we will omit the n.

A sound and complete axiomatisation for semantic equality was developed in [11, 2, 32], and in [6] for inclusion. The above signature together with the axioms, recalled in Figure 1, form the theory of Interacting Hopf Algebras. The resulting prop is denoted by $\mathbb{IH}_{\mathsf{k}}$.

▶ Remark 6. Thanks to the compact closed structure, each of the axioms and laws that we prove in the text can be read both as $c \overset{\mathbb{IH}}{=} d$ and $c^{op} \overset{\mathbb{IH}}{=} d^{op}$. For example, by •*–coas* we also know that  $\overset{\mathbb{IH}}{=}$ .

▶ **Theorem 7.** *For all circuits c, d in* Circ*, $[\![c]\!] \subseteq [\![d]\!]$ if and only if $c \overset{\mathbb{IH}}{\subseteq} d$.*

We now come to expressivity: which relations in $\mathsf{Rel}_{\mathsf{k}}$ are expressed by Circ? The answer is that Circ captures exactly $\mathsf{LinRel}_{\mathsf{k}}$ (see Definition 2).

▶ **Theorem 8.** $\mathbb{IH}_{\mathsf{k}} \cong \mathsf{LinRel}_{\mathsf{k}}$.

The above result means that $\mathbb{IH}_{\mathsf{k}}$ is the theory of linear relations. It is convenient to recall from [27] a useful fact: circuits in $\overrightarrow{\mathsf{Circ}}$ express exactly k-matrices, as illustrated below:

▶ **Example 9.** Consider the circuit $c\colon 3 \to 4$ below and its representation as a 4×3 matrix. Note that $A_{ij} = k$ whenever k is the scalar encountered on the path from the ith port to the jth port. If there is no path, then $A_{ij} = 0$. It is easy to check that $[\![c]\!] = \{(x, y) \in \mathsf{k}^3 \times \mathsf{k}^4 \mid y = Ax\}$.

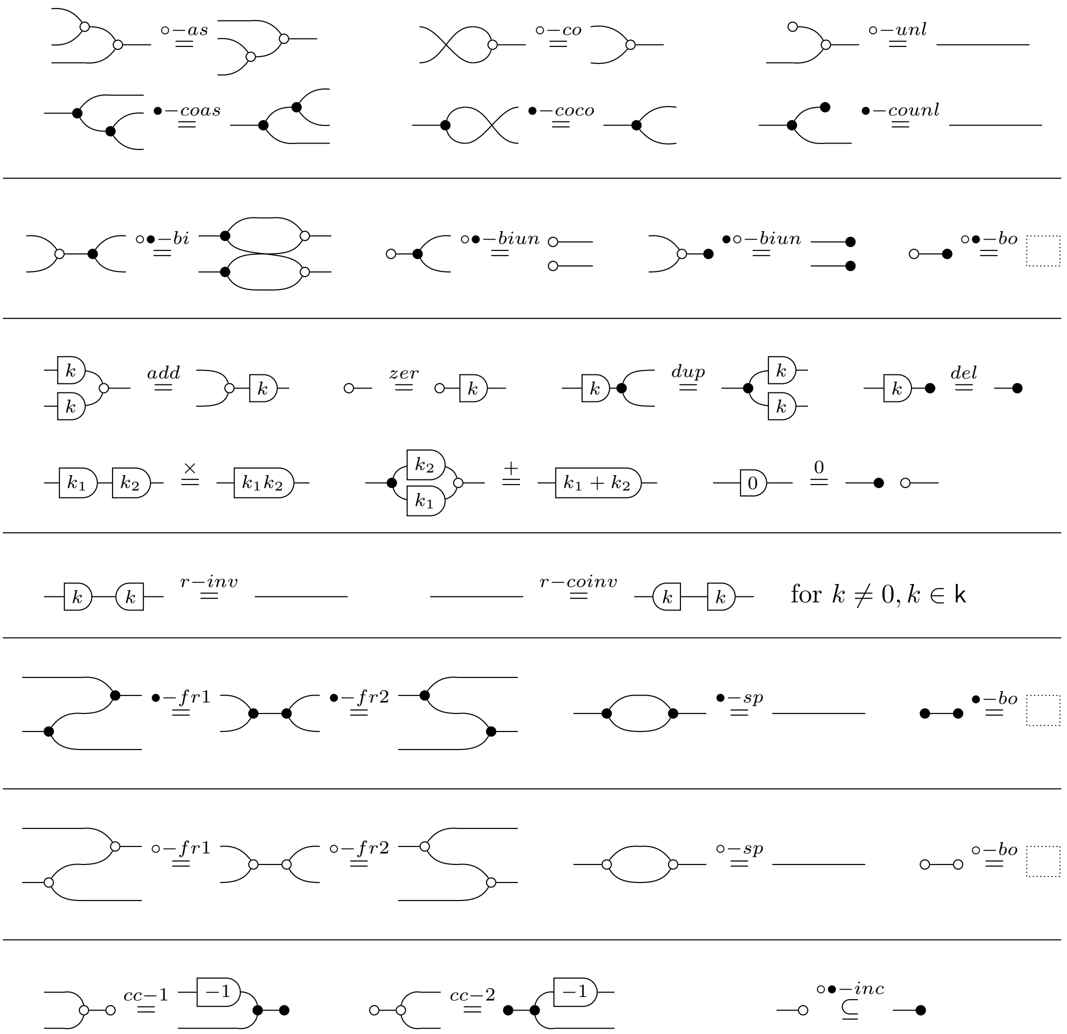
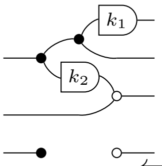
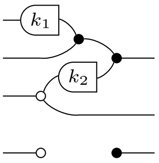


Figure 1 Axioms of Interacting Hopf Algebras ($\mathbb{IH}_{\mathsf{k}}$).

$c =$ 

$$A = \begin{pmatrix} k_1 & 0 & 0 \\ 1 & 0 & 0 \\ k_2 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$d =$ 

Dually, circuits in $\overleftarrow{\mathsf{Circ}}$ are "reversed" matrices: inputs on the right and outputs on the left. For instance $d\colon 4 \to 3$ again encodes A, but its semantics is $[\![d]\!] = \{(y, x) \in \mathsf{k}^4 \times \mathsf{k}^3 \mid y = Ax\}$.

4 The Theory of Polyhedral cones

Hereafter, we assume k to be an *ordered field*, namely a field equipped with a total order $\leq$ such that for all $i, j, k \in \mathsf{k}$: (a) if $i \leq j$, then $i + k \leq j + k$; (b) if $0 \leq i$ and $0 \leq j$, then $0 \leq ij$.

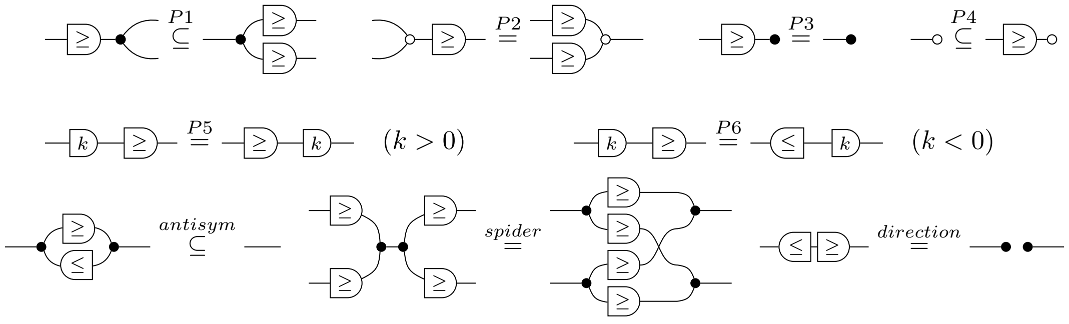
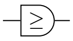
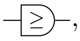
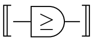
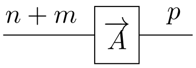


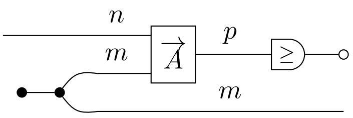
Figure 2 Axioms of $\mathbb{IH}_{\mathsf{k}}^{\geq}$.

We extend the signature in (5) and (6) with the following generator

 (8)

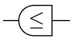
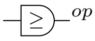
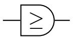
and denote the resulting free prop $\mathsf{Circ}^{\geq}$. The morphism $[\![\cdot]\!] : \mathsf{Circ}^{\geq} \to \mathsf{Rel}_{\mathsf{k}}$ behaves as (7) for the generators in (5) and (6), whereas for , it is defined as hinted by our syntax: $[\![$$]\!] = \{\, (x, y) \mid x, y \in \mathsf{k}, x \geq y \,\}$.

▶ **Example 10.** Let  be a diagram in $\overrightarrow{\mathsf{Circ}}$ denoting some matrix A (see Example 9). Consider the following circuit in $\mathsf{Circ}^{\geq}$

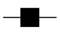
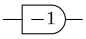


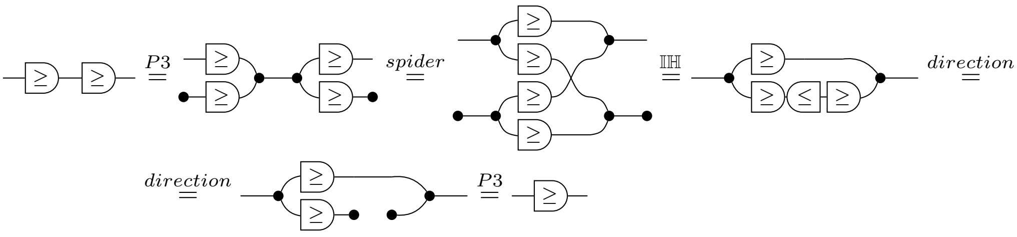
(9)

It easy to check that its semantics is the relation $C = \{(x, y) \in \mathsf{k}^n \times \mathsf{k}^m \mid A \begin{pmatrix} x \\ y \end{pmatrix} \geq 0\}$. Thus (9) denotes a *polyhedral cone*, i.e., the set of solutions of some system of linear inequations.

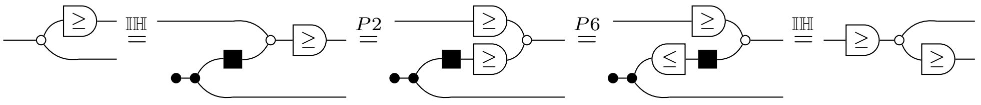
We denote by $\mathbb{IH}_{\mathsf{k}}^{\geq}$ the prop generated by the theory consisting of this signature (namely, (5), (6) and (8)), the axioms of $\mathbb{IH}_{\mathsf{k}}$ and the axioms in Figure 2, where  is just  (see Example 5). The first two rows of axioms describe the interactions of  with the generators in (5). The third row asserts that $\geq$ is antisymmetric and satisfies an appropriate spider condition. In the last axiom, the right-to-left inclusion states that for all $k, l \in \mathsf{k}$, there exists an upper bound u, i.e. $u \geq k$, $u \geq l$. The left-to-right inclusion is redundant.

The axioms are perhaps surprising: e.g. reflexivity and transitivity are not included. As a taster for working with the diagrammatic calculus, we prove these properties below.

▶ Remark 11. We will often use the alternative antipode notation  for the scalar .

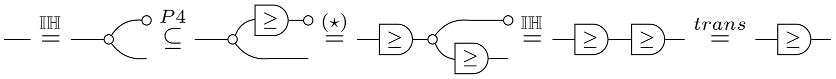


The derivation above proves transitivity. The following derivation



(⋆)

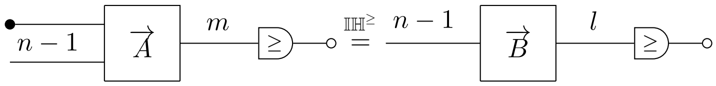
is used below to show that $\geq$ is reflexive:



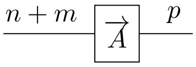
▶ Remark 12. When we annotate equalities with $\mathbb{IH}$, we are making use of multiple unmentioned derived laws presented in related works. These can be seen to hold also by appealing to Theorem 7.

Routine computations confirm that all the axioms are sound. To prove completeness, we give diagrammatic proofs of several well-known results.

▶ **Proposition 13** (Fourier-Motzkin elimination). *For each arrow $A\colon n \to m$ of $\overrightarrow{\mathsf{Circ}}$, there exists an arrow $B\colon n-1 \to l$ of $\overrightarrow{\mathsf{Circ}}$ such that*

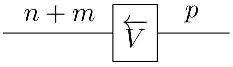


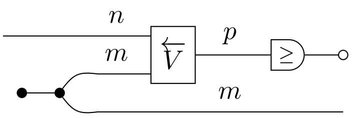
The proof, in [5, Appendix B.1], mimics the Fourier-Motzkin elimination, an algorithm for eliminating variables from a system of linear inequations (i.e. projecting a polyhedral cone).

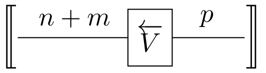
The next step is a normal form theorem. A circuit $c\colon n \to m$ of $\mathsf{Circ}^{\geq}$ is said to be in *polyhedral normal form* if there is an arrow  of $\overrightarrow{\mathsf{Circ}}$, such that $c =$ (9).

▶ **Theorem 14** (First Normal Form). *For each arrow $c\colon n \to m$ of $\mathsf{Circ}^{\geq}$, there is another arrow $d\colon n \to m$ of $\mathsf{Circ}^{\geq}$ in polyhedral normal form such that $c \overset{\mathbb{IH}^{\geq}}{=} d$.*

The proof is by induction on the structure of $\mathsf{Circ}^{\geq}$. The only challenging case is sequential composition, which uses the Fourier-Moztkin elimination. Details are in [5, Appendix B.2].

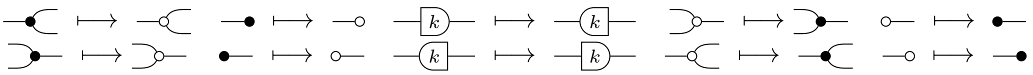
Diagrams in $\mathsf{Circ}^{\geq}$ enjoy a second normal form: an arrow $c\colon n \to m$ of $\mathsf{Circ}^{\geq}$ is said to be in *finitely generated normal form* if there is an arrow  of $\overleftarrow{\mathsf{Circ}}$, such that

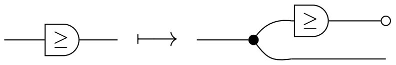
$c =$ 

As recalled in Example 9, the semantics of the circuit in $\overleftarrow{\mathsf{Circ}}$ is  $=$ $\{(u,z) \in \mathsf{k}^{n+m} \times \mathsf{k}^p \mid u = Vz\}$ for some $(n+m) \times p$ matrix V. Then, it is easy to check that $[\![c]\!] = \{(x,y) \in \mathsf{k}^n \times \mathsf{k}^m \mid \exists z \in \mathsf{k}^p \text{ s.t. } \begin{pmatrix} x \\ y \end{pmatrix} = Vz, z \geq 0\}$. The matrix V can be regarded as a set of column vectors $\{v_1, \ldots, v_p\}$ and $[\![c]\!]$ as the *conic combination* of those vectors, defined as $\mathsf{cone}(V) = \{z_1 v_1 + \ldots + z_p v_p \mid z_i \in \mathsf{k},\, z_i \geq 0\}$. Sets of vectors generated in this way are known as *finitely generated cones*, which justifies the name of the normal form.

To prove the existence of this normal form, we introduce the polar operator, an important construction in convex analysis which, in our approach, has a simple inductive definition.

▶ **Definition 15.** *The polar operator* $\cdot^\circ \colon \mathsf{Circ}^{\geq} \to \mathsf{Circ}^{\geq}$ *is the functor inductively defined as:*



 $(c\,;d)^\circ = c^\circ\,; d^\circ \quad (c \oplus d)^\circ = c^\circ \oplus d^\circ$

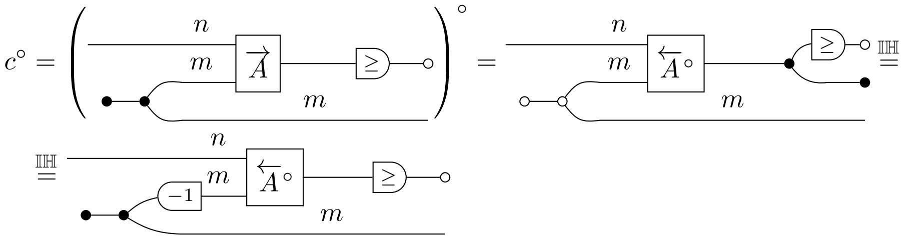
The polar operator enjoys the following useful properties.

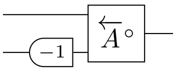
▶ **Proposition 16.** *For all arrows* $c, d \colon n \to m$ *in* $\mathsf{Circ}^{\geq}$*, it holds that*

1. *if* $c \overset{\mathbb{IH}^{\geq}}{\subseteq} d$ *then* $(d)^\circ \overset{\mathbb{IH}^{\geq}}{\subseteq} (c)^\circ$*;*
2. $(c^\circ)^\circ \overset{\mathbb{IH}^{\geq}}{=} c$*;*
3. *if* c *is an arrow of* $\overrightarrow{\mathsf{Circ}}$*, then* c° *is an arrow of* $\overleftarrow{\mathsf{Circ}}$*.*

▶ **Proposition 17.** *For each arrow* $c \colon n \to m$ *of* $\mathsf{Circ}^{\geq}$ *in polyhedral normal form there is an arrow* $d \colon n \to m$ *of* $\mathsf{Circ}^{\geq}$ *in finitely generated normal form, such that* $(c)^\circ \overset{\mathbb{IH}^{\geq}}{=} d$*.*

Proof. Let c as in (9). Then, by applying the definition of $\cdot^\circ$



which is in finitely generated normal form, since  is by Proposition 16 in $\overleftarrow{\mathsf{Circ}}$. ◀

▶ **Theorem 18** (Second Normal Form)**.** *For each arrow* $c \colon n \to m$ *of* $\mathsf{Circ}^{\geq}$*, there is an arrow* $d \colon n \to m$ *of* $\mathsf{Circ}^{\geq}$ *in finitely generated normal form such that* $c \overset{\mathbb{IH}^{\geq}}{=} d$*.*

Proof. By Theorem 14, there exists an arrow $p \colon n \to m$ of $\mathsf{Circ}^{\geq}$ in polyhedral normal form, such that $c^\circ \overset{\mathbb{IH}^{\geq}}{=} p$. By Proposition 16.1, $p^\circ \overset{\mathbb{IH}^{\geq}}{=} (c^\circ)^\circ$ and, by Proposition 16.2 , $(c^\circ)^\circ \overset{\mathbb{IH}^{\geq}}{=} c$, thus $p^\circ \overset{\mathbb{IH}^{\geq}}{=} c$. Since p is in polyhedral normal form, by Proposition 17, there exists a circuit d in finitely generated normal form such that $d \overset{\mathbb{IH}^{\geq}}{=} p^\circ \overset{\mathbb{IH}^{\geq}}{=} c$. ◀

An immediate consequence of the two normal form theorems is the well-known Weyl-Minkowski theorem, which states that every polyhedral cone is finitely generated and, vice-versa, every finitely generated cone is polyhedral. It is worth emphasising that neither the polyhedral nor the finitely generated normal form are unique: different matrices may give rise to the same cone. However, with the finitely generated normal form, proving completeness requires only a few more lemmas, which are given in [5, Appendix B.4].

▶ **Theorem 19** (Completeness)**.** *For all circuits* $c, d \in \mathsf{Circ}^{\geq}$*, if* $[\![c]\!] \subseteq [\![d]\!]$ *then* $c \overset{\mathbb{IH}^{\geq}}{\subseteq} d$*.*

We now come to the problem of expressivity: what is the image of $\mathsf{Circ}^{\geq}$ through $[\![\cdot]\!]$? It turns out that $\mathsf{Circ}^{\geq}$ denotes exactly the arrows of $\mathsf{PC_k}$ (see Definition 2).

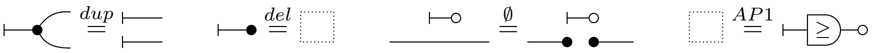


Figure 3 Axioms of $\mathsf{a}\mathbb{IH}^{\geq}_{\mathsf{k}}$.

▶ **Proposition 20** (Expressivity). *For each arrow $C\colon n \to m$ in PC_{k} there exists a circuit $c\colon n \to m$ of $\mathsf{Circ}^{\geq}$, such that $C = [\![c]\!]$. Vice-versa, for each circuit $c\colon n \to m$ of $\mathsf{Circ}^{\geq}$ there exists an arrow $C\colon n \to m$ of PC_{k}, such that $[\![c]\!] = C$.*

By Theorem 19 and Proposition 20 it follows that

▶ **Corollary 21.** $\mathbb{IH}^{\geq}_{\mathsf{k}} \cong \mathsf{PC}_{\mathsf{k}}$

The above allows us to conclude that $\mathbb{IH}^{\geq}_{\mathsf{k}}$ is the theory of polyhedral cones.

5 The theory of Polyhedra

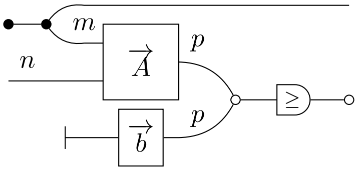
In [9], the signature of Circ was extended with an additional generator

(10)

with semantics $[\![$⊢—$]\!] = \{(\bullet, 1)\}$. The three leftmost equations in Figure 3 provide a complete axiomatisation for semantic equality. In terms of expressivity, the resulting calculus expresses exactly *affine spaces*, namely sets of solutions of $Ax + b = 0$ for some matrix A and vector b.

Here we extend $\mathsf{Circ}^{\geq}$ with (10). Let $\mathsf{ACirc}^{\geq}$ be the prop freely generated by (5), (6), (8) and (10). The circuits of $\mathsf{ACirc}^{\geq}$ can denote *polyhedra*, namely sets $P = \{\, x \in \mathsf{k}^n \mid Ax{+}b \geq 0 \,\}$. Observe that the empty set $\emptyset$ is a polyhedron, but not a polyhedral cone.

▶ **Example 22.** Let $\overset{n+m}{\text{——}}\boxed{\overrightarrow{A}}\overset{p}{\text{—}}$ and ⊢$\boxed{\overrightarrow{b}}\overset{p}{\text{—}}$ be circuits in $\overrightarrow{\mathsf{Circ}}$ denoting, respectively some matrix A and some vector b. Consider the following circuit in $\mathsf{ACirc}^{\geq}$.

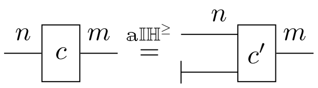
 (11)

It is easy to check that its semantics is the relation $P = \{\, (x, y) \in \mathsf{k}^n \times \mathsf{k}^m \mid A \begin{pmatrix} y \\ x \end{pmatrix} + b \geq 0 \,\}$. Another useful circuit is ⊢—○: $[\![$⊢—○$]\!] = \{(\bullet, 1)\};\{(0, \bullet)\} = \emptyset$. Intuitively, it behaves as a logical false, since for any relation R in $\mathsf{Rel}_{\mathsf{k}}$, $R \oplus \emptyset = \emptyset = \emptyset \oplus R$.

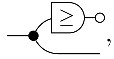
In order to obtain a complete axiomatisation, it is enough to add to the three axioms in [9], only one axiom: $AP1$ in Figure 3. Intuitively, $AP1$ states that $1 \geq 0$. The prop freely generated by (5), (6), (8), (10) and the axioms in Figures 1, 2 and 3 is denoted by $\mathsf{a}\mathbb{IH}^{\geq}_{\mathsf{k}}$.

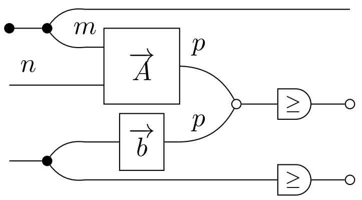
With these axioms, any circuit in $\mathsf{ACirc}^{\geq}$ can be shown equivalent to one of the form (11). This is shown using the first normal form (Theorem 14) for $\mathsf{Circ}^{\geq}$ and the following lemma.

▶ **Lemma 23.** *For any $c\colon n \to m$ of $\mathsf{ACirc}^{\geq}$, there exists $c'\colon n+1 \to m$ of $\mathsf{Circ}^{\geq}$ such that*



▶ **Theorem 24.** *For all c of* $\mathsf{ACirc}^{\geq}$*, there exist d in the form of* (11) *such that* $c \overset{\mathrm{a}\mathbb{IH}^{\geq}}{=} d$.

To prove completeness, the notion of homogenization is pivotal. The *homogenization* of a polyhedron $P = \{x \in \mathsf{k}^n \mid Ax + b \geq 0\}$ is the cone $P^H = \{(x, y) \in \mathsf{k}^{n+1} \mid Ax + by \geq 0, y \geq 0\}$. Diagrammatically, this amounts to replace the $\vdash\!\!-$ in (11) with , obtaining the following diagram



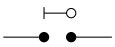
(12)

▶ **Lemma 25.** *Let $P_1, P_2 \subseteq \mathsf{k}^n$ be two non-empty polyhedra. Then, $P_1 \subseteq P_2$ iff $P_1^H \subseteq P_2^H$.*

Using Theorem 24 and Lemma 25, we can reduce completeness for *non-empty* polyhedra to completeness of polyhedral cones.

▶ **Theorem 26.** *Let $c, d \colon n \to m$ in* $\mathsf{ACirc}^{\geq}$ *denote non-empty polyhedra. If $[\![c]\!] \subseteq [\![d]\!]$, then* $c \overset{\mathrm{a}\mathbb{IH}^{\geq}}{\subseteq} d$.

Completeness for empty polyhedra requires a few additional lemmas, given in [5, Appendix D].

▶ **Theorem 27.** *For all circuits c in* $\mathsf{ACirc}^{\geq}$*, if $[\![c]\!] = \emptyset$ then* $c \overset{\mathrm{a}\mathbb{IH}^{\geq}}{=}$ 

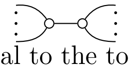
▶ **Corollary 28** (Completeness)**.** *For all circuits c, d in* $\mathsf{ACirc}^{\geq}$*, if $[\![c]\!] \subseteq [\![d]\!]$ then* $c \overset{\mathrm{a}\mathbb{IH}^{\geq}}{\subseteq} d$.

Finally, we characterise the semantic image of circuits in $\mathsf{ACirc}^{\geq}$.

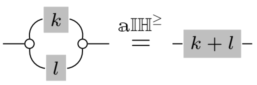
▶ **Proposition 29** (Expressivity)**.** *For each arrow $P \colon n \to m$ in* $\mathsf{P_k}$ *there exist a circuit $c \colon n \to m$ in* $\mathsf{ACirc}^{\geq}$*, such that $P = [\![c]\!]$. Vice-versa, for each circuit $c \colon n \to m$ of* $\mathsf{ACirc}^{\geq}$ *there exists an arrow $P \colon n \to m$ of* $\mathsf{P_k}$*, such that $[\![c]\!] = P$.*

Indeed, $\mathrm{a}\mathbb{IH}^{\geq}_{\mathsf{k}}$ is the theory of polyhedra:

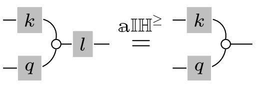
▶ **Corollary 30.** $\mathrm{a}\mathbb{IH}^{\geq}_{\mathsf{k}} \cong \mathsf{P_k}$

▶ **Example 31** (Flow networks)**.** Consider again flow networks, previously mentioned in the Introduction: edges with capacity k can be expressed in $\mathsf{ACirc}^{\geq}$ by the diagram in (3) hereafter referred as $-\boxed{k}-$. Observe that $[\![-\boxed{k}-]\!] = \{(x, x) \mid 0 \leq x \leq k\}$ is exactly the expected meaning of an edge in a flow network. Nodes with n incoming edges and m outgoing edges can be encoded by the diagram n  m. Again the semantics is the expected one: the total incoming flow must be equal to the total outgoing flow. For an example of the encoding, check the flow network in (1) and the corresponding diagram in (2).

The axioms in $\mathrm{a}\mathbb{IH}^{\geq}_{\mathsf{k}}$ can be exploited to compute the maximum flow of a network. By using the following two derived laws (proved in [5, Appendix E])

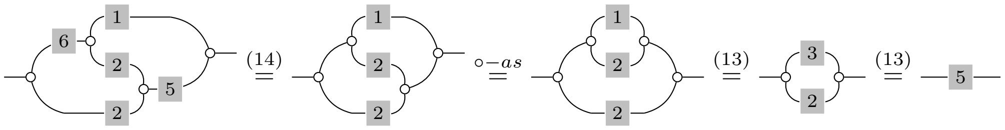


(13)



$(k + q \leq l)$ (14)

one can transform the diagram in (2) into $-\boxed{5}-$



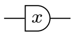
meaning that $[\![(2)]\!] = \{(x,x) \mid 0 \leq x \leq 5\}$, i.e., the maximum flow of (2) is exactly 5.

6 Adding states to polyhedra

We have shown that $\mathsf{ACirc}^{\geq}$ with its associated equational theory $\mathsf{a}\mathbb{IH}^{\geq}_{\mathsf{k}}$ provides a sound and complete calculus for polyhedra. In this section, we extend the calculus with a canonical notion of *state*. Our development follows step-by-step the general recipe illustrated in [8, §4].

6.1 The Calculus of Stateful Polyhedral Processes

We call SPP the prop freely generated by (5), (6), (8), (10) and the following.

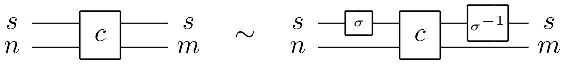


(15)

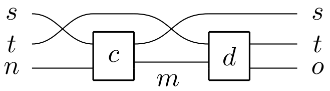
Intuitively, the register $-\boxed{x}-$ is a synchronous buffer holding a value $k \in \mathsf{k}$: when it receives $l \in \mathsf{k}$ on the left port, it emits k on the right one and stores l. To give a formal semantics to such behaviour we exploit a "state bootstrapping" technique that appears in several places in the literature, e.g. in the setting of cartesian bicategories [25] and geometry of interaction [22].

▶ **Definition 32** (Stateful processes [25]). *Let* T *be a prop. Define* $\mathsf{St}(\mathsf{T})$ *as the prop where:*

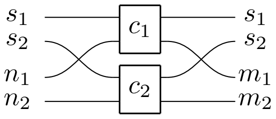
- *morphisms* $n \to n$ *are pairs* (s, c) *where* $s \in \mathbb{N}$ *and* $c \colon s + n \to s + m$ *is a morphism of* T*, quotiented by the smallest equivalence relation including every instance of*



for a permutation $\sigma{:}s \to s$*; the order is defined as* $(s,c) \overset{\mathsf{St}(\mathsf{T})}{\subseteq} (s,d)$ *if and only* $c \overset{\mathsf{T}}{\subseteq} d$*.*
- *the composition of* $(s,c) \colon n \to m$ *and* $(t,d) \colon m \to o$ *is* $(s+t, e)$ *where* e *is the arrow of* T *given by*



- *the monoidal product of* $(s_1, c_1) \colon n_1 \to m_1$ *and* $(s_2, c_2) \colon n_2 \to m_2$ *is* $(s_1 + s_2, e)$ *where* e *is given by*



- *the identity on* n *is* $(0, id_n)$ *and the symmetry of* n, m *is* $(0, \sigma_{n,m})$*.*

We use $\mathsf{St}(\mathsf{Rel}_{\mathsf{k}})$ as our semantic domain: in an arrow $(s, R) \colon n \to m$, s records the number of registers while $R \colon s + n \to s + m$ is a relation $R \subseteq \mathsf{k}^s \times \mathsf{k}^n \times \mathsf{k}^s \times \mathsf{k}^m$ containing quadruples (u, l, v, r) representing transitions: u and v are the starting and arrival state (namely vectors in k^s, holding a value in k to each of the s registers), while l and r are vectors of values occurring on the left and the right ports. The equivalence relation $\sim$ ensures that registers remains anonymous: it equates arrows that only differ by a bijective relabelling of their lists of registers. This is, therefore, a syntactic form of equivalence similar in flavour to α-equivalence, since it discards intentional details not relevant for the dynamics of processes.

We can now give the semantics of SPP as the morphism $\langle\!\langle\cdot\rangle\!\rangle \colon \mathsf{SPP} \to \mathsf{St}(\mathsf{Rel}_\mathsf{k})$ defined:

$$\langle\!\langle -\boxed{x}- \rangle\!\rangle = (1, \{(k,l,l,k) \mid l,k \in \mathsf{k}\}) \quad \text{and} \quad \langle\!\langle o \rangle\!\rangle = (0, [\![o]\!])$$

for all generators o in (5), (6), (8), (10). For instance $\langle\!\langle -\!\bullet \rangle\!\rangle = (0, \{(\bullet, k) \mid k \in \mathsf{k}\})$. The semantics of $-\boxed{x}-$ is the expected behaviour: from any state k (the stored value), it makes a transition to state l when l is on the left port and k is on the right. This can be restated as a structural operational semantics (sos) axiom $(-\boxed{x}-, k) \xrightarrow[k]{l} (-\boxed{x}-, l)$ where the labels above and under the arrow stand, respectively, for the values on the left and right ports.

Theorem 30 in [8] ensures that no other data is needed for an axiomatisation: let $\mathbb{S}\mathbb{a}\mathbb{IH}^{\geq}$ be the prop generated by (5), (6), (8), (10), (15) and the axioms in Figures 1, 2 and 3.

▶ **Theorem 33.** *For all c, d in* SPP*, if $\langle\!\langle c \rangle\!\rangle \subseteq \langle\!\langle d \rangle\!\rangle$ then $c \overset{\mathbb{S}\mathbb{a}\mathbb{IH}^{\geq}}{\subseteq} d$. Moreover $\mathbb{S}\mathbb{a}\mathbb{IH}^{\geq} \cong \mathsf{St}(\mathsf{P}_\mathsf{k})$.*

Proof of Theorem 33. We make more clear the correspondence with [8]. Considering the following diagram.

$$\begin{tikzcd}
\mathsf{SPP} \arrow[r, two heads, "q"] \arrow[rrrr, bend left, "\langle\!\langle\cdot\rangle\!\rangle"] & \mathbb{S}\partial\mathbb{IH}^{\geq} \arrow[r, "F"] & \mathsf{St}(\mathbb{a}\mathbb{IH}^{\geq}_{\mathsf{k}}) \arrow[r, "\mathsf{St}(\cong)"] & \mathsf{St}(\mathsf{P}_\mathsf{k}) \arrow[r, tail, "\mathsf{St}(\iota)"] & \mathsf{St}(\mathsf{Rel}_\mathsf{k})
\end{tikzcd}$$

The morphism $\mathsf{St}(\iota)$ is just the obvious extension of the inclusion $\iota \colon \mathsf{P}_\mathsf{k} \to \mathsf{Rel}_\mathsf{k}$. Similarly, $\mathsf{St}(\cong)$ is the extension of the isomorphism shown in Corollary 30. The morphism q is just the obvious quotient from SPP to $\mathbb{S}\mathbb{a}\mathbb{IH}^{\geq}$. The interesting part is provided by the morphism $F \colon \mathbb{S}\mathbb{a}\mathbb{IH}^{\geq} \to \mathsf{St}(\mathbb{a}\mathbb{IH}^{\geq}_{\mathsf{k}})$ defined in [8, §4.1]: take T as $\mathbb{a}\mathbb{IH}^{\geq}_{\mathsf{k}}$ and $\mathsf{T} + \mathsf{X}$ as $\mathbb{S}\mathbb{a}\mathbb{IH}^{\geq}$. By Theorem 30 in [8], since $\mathbb{a}\mathbb{IH}^{\geq}_{\mathsf{k}}$ is compact closed, then F is an isomorphism of props. To see that it is an isomorphism of ordered props, it is immediate to check that both F and its inverse G defined in [8, §4.1] preserves the order. ◀

We conclude by observing the semantics can be presented with intuitive sos rules. Indeed, the same rules as in [8, §2] – interpreted over a field rather than the naturals – and:

$$\frac{x \geq y}{(-\boxed{\geq}-, \bullet) \xrightarrow[y]{x} (-\boxed{\geq}-, \bullet)} \qquad (\vdash\!\!-, \bullet) \xrightarrow[1]{\bullet} (\vdash\!\!-, \bullet)$$

This diagrammatic language is, therefore, similar in flavour to traditional process calculi, and we call it the *calculus of stateful polyhedral processes*. Theorem 33 affirms that it expresses exactly the stateful polyhedral processes.

6.2 Bounded Continuous Petri Nets

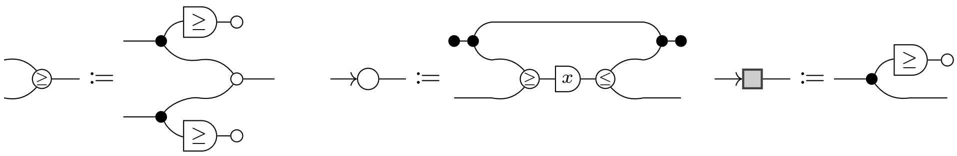
Hereafter k is fixed to be the real numbers $\mathbb{R}$ and the set of non-negative reals is denoted by $\mathbb{R}_+ = \{\, r \in \mathbb{R} \mid r \geq 0 \,\}$. A *continuous* Petri net [18] differs from a (discrete) Petri net in that:

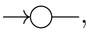
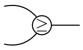
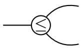
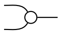
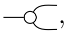
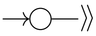
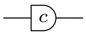
- markings are real valued – that is, places hold a non-negative real number of tokens,
- transitions can consume and produce non-negative real numbers of tokens,
- transitions can be fired a non-negative real number amount of times – for example a transition can be fired 0.5 times, producing and consuming half the tokens.

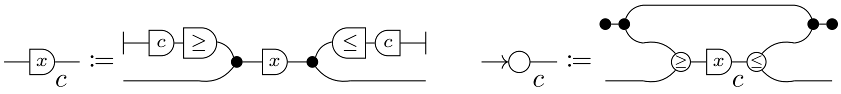
▶ **Definition 34** (Continuous Petri nets and their semantics). *A Petri net $\mathcal{P} = (P, T, {}^\circ-, -^\circ)$ consists of a finite set of places P, a finite set of transitions T, and functions ${}^\circ-, -^\circ : T \to {\mathbb{R}_+}^P$. Given $\mathbf{y}, \mathbf{z} \in {\mathbb{R}_+}^P$, we write $\mathbf{y} \to \mathbf{z}$ if there exists $\mathbf{t} \in {\mathbb{R}_+}^T$ such that ${}^\circ\mathbf{t} \leq \mathbf{y}$ and $\mathbf{z} = \mathbf{y} - {}^\circ\mathbf{t} + \mathbf{t}^\circ$, where ${}^\circ\mathbf{t}$ and $\mathbf{t}^\circ$ are the evident liftings of ${}^\circ()$ and $()^\circ$, e.g. ${}^\circ\mathbf{t}(p) = \sum_{s \in T} \mathbf{t}(s) \cdot {}^\circ s(p)$. The (step) operational semantics of $\mathcal{P}$ is the relation $\langle \mathcal{P} \rangle = \{(\mathbf{y}, \mathbf{z}) \mid \mathbf{y} \to \mathbf{z}\} \subseteq {\mathbb{R}_+}^P \times {\mathbb{R}_+}^P$.*

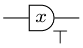
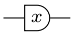
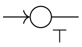
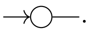
As for ordinary Petri nets, one can consider *bounded nets*: each place has a maximum capacity $c \in \mathbb{R}_+ \cup \{\top\}$: a place with capacity $\top$ is unbounded. The above definition is therefore extended with a boundary function $\mathbf{b} \in (\mathbb{R}_+ \cup \{\top\})^P$ and the transition relation $\mathbf{y} \to \mathbf{z}$ is modified by additionally requiring that $\mathbf{y}, \mathbf{z} \leq \mathbf{b}$. Since $r \leq \top$ for all $r \in \mathbb{R}_+$, continuous Petri nets are instances of bounded continuous nets where every place is unbounded.

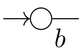
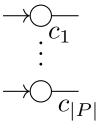
To encode continuous Petri nets and their bounded variant as stateful polyhedral processes, it is convenient to introduce syntactic sugar: the circuit below left is an adder that takes only positive values as inputs, the central circuit models a place, and the last one a transition.

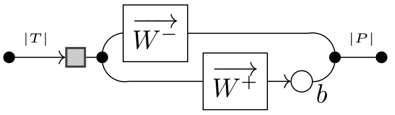


Observe that for , it is essential the use of  and and its opposite . Indeed, replacing them by ordinary adders  and , would give as semantics the whole space $\mathbb{R}^2 \times \mathbb{R}^2$, while as defined above $\langle\!\langle$  $\rangle\!\rangle = (1, \{(m, i, m - o + i, o) \mid i, o, m \in \mathbb{R}_+,\ o \geq m\})$, modelling exactly the expected behaviour of a place. In the diagrams below  is either a scalar $r \in \mathbb{R}_+$ or $\top =$ —●●—.



The leftmost diagram models a buffer with capacity c, while the rightmost a place with capacity c. Since $\top =$ —●●—, it holds that  $\overset{\mathbb{S}\mathsf{a}\mathbb{IH}^{\geq}}{=}$  and  $\overset{\mathbb{S}\mathsf{a}\mathbb{IH}^{\geq}}{=}$ .

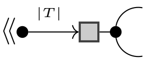
By choosing an ordering on places and transitions, the functions ${}^\circ-, -^\circ : T \to {\mathbb{R}_+}^P$ can be regarded as $\mathbb{R}_+$-matrices of type $|T| \to |P|$ and thus can be encoded as $\overrightarrow{\mathsf{Circ}}$ circuits, hereafter denoted by respectively W^- and W^+. The ordering on P also makes the boundary function b a vector $\begin{pmatrix} c_1 \\ \vdots \\ c_{|P|} \end{pmatrix}$ in $(\mathbb{R}_+ \cup \{\top\})^{|P|}$: we write  for  . Any bounded Continuous Petri net $\mathcal{P}$ can be encoded as the following circuit $d_{\mathcal{P}} : 0 \to 0$ in SPP.



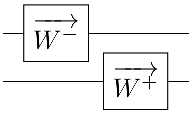
It is easy to show that $\mathcal{P}$ and $d_{\mathcal{P}}$ have the same semantics.

▶ **Proposition 35.** *For all bounded continuous Petri net $\mathcal{P}$, $\langle \mathcal{P} \rangle \sim \langle\!\langle d_{\mathcal{P}} \rangle\!\rangle$.*

Proof of Proposition 35. In order to compute $\langle\!\langle d_{\mathcal{P}} \rangle\!\rangle$, it is convenient to cut $d_{\mathcal{P}}$ in three parts. The leftmost part of $d_{\mathcal{P}}$ has the following semantics

$\langle\!\langle$  $\rangle\!\rangle = \quad (0, \{(\bullet, \bullet, \bullet, \begin{pmatrix} t \\ t \end{pmatrix}) \mid t \in {\mathbb{R}_+}^{|T|}\})$

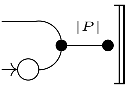
The central part

$\langle\!\langle$  $\rangle\!\rangle$ $= \quad (0\{(\bullet, \begin{pmatrix}x_1\\x_2\end{pmatrix}, \bullet, \begin{pmatrix}y_1\\y_2\end{pmatrix}) \mid W^- x_1 = y_1,\ W^+ x_2 = y_2\})$

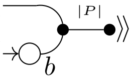
By definition of $\langle\!\langle\cdot\rangle\!\rangle$, the composition of the two semantics above is the pair

$$(0, \{(\bullet, \bullet, \bullet, \begin{pmatrix}y_1\\y_2\end{pmatrix}) \mid \exists t \in {\mathbb{R}_+}^{|T|} \text{ s.t. } W^- t = y_1,\ W^+ t = y_2\})$$

The right-most part

$[\![$  $]\!]$ $= \quad \{(y, \begin{pmatrix}i\\o\end{pmatrix}, y - o + i, \bullet) \mid i, o, y \in {\mathbb{R}_+}^{|T|},\ o \geq y\}$

The semantics of rightmost part is the pair

$\langle\!\langle$  $\rangle\!\rangle$ $= \quad (|P|, \{(y, \begin{pmatrix}o\\i\end{pmatrix}, y - o + i, \bullet) \mid i, o, y \in {\mathbb{R}_+}^{|P|},\ o \geq y,\ y - o + i \leq b,\ y \leq b\})$

By composing everything we obtain $(|P|, \{(y, \bullet, y - i + o, \bullet) \mid \exists t \in {\mathbb{R}_+}^{|T|}$ s.t. $i, o, y \in {\mathbb{R}_+}^{|P|},\ o \geq y,\ y - o + i \leq b,\ y \leq b,\ W^- t = o,\ W^+ t = i\})$ that is $\langle\!\langle d_{\mathcal{P}} \rangle\!\rangle = (|P|, \{(y, \bullet, z, \bullet) \mid \exists t \in {\mathbb{R}_+}^{|T|}$ s.t. $W^- t \geq y,\ y - W^- t + W^+ t = z \leq b,\ y \leq b\})$.

Since the equivalence is stated modulo $\sim$, then it is safe to fix an ordering on P and T. Thus, rather than considering $(y, z) \in \langle P \rangle$ as functions in ${\mathbb{R}_+}^P$ they can be regarded as vectors in ${\mathbb{R}_+}^{|P|}$. One can thus conclude by observing that $y \to z$ if and only if there exists $t \in {\mathbb{R}_+}^{|T|}$ such that $W^- t \geq y$, $y - W^- t + W^+ t = z$ and $y, z \leq b$. ◀

7 Conclusions and Future Work

We have introduced the theories of polyhedral cones and the one of polyhedra. In other words, we have identified suitable sets of generators and axioms for which we proved completeness and expressivity. As side results, we get an inductive definition of the notion of polar cone, as well as an understanding of Weyl-Minkowski theorem as a normal form result.

As shown by Example 31, the theory of polyhedra allows us to represent networks with bounded resources, not expressible in $\mathbb{IH}$, and to manipulate them as symbolic expressions.

Indeed the passage from linear relations to polyhedra is a reflection of the fact that, operationally, we are able to consider several patterns of computations important in computer science, as opposed to purely linear patterns, traditionally studied in system/control theory.

For instance, as shown in §6, the addition to $\mathsf{a}\mathbb{IH}^{\geq}$ of a single generator, $-\!\boxed{x}\!-$, directly gives us a concurrent extension of the signal flow calculus [2, 12], introduced as a compositional account for linear dynamical systems, that is expressive enough to encode continuous Petri nets [18].

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., USA, 1993.

2 John Baez and Jason Erbele. Categories in control. *Theory and Applications of Categories*, 30:836–881, 2015. arXiv:1405.6881.

3 John C. Baez. Network theory, 2014. URL: `http://math.ucr.edu/home/baez/networks/`.

4 John C Baez and Brendan Fong. A compositional framework for passive linear networks. *arXiv preprint*, 2015. `arXiv:1504.05625`.

5 Filippo Bonchi, Alessandro Di Giorgio, and Pawel Sobocinski. Diagrammatic polyhedral algebra, 2021. `arXiv:2105.10946`.

6 Filippo Bonchi, Joshua Holland, Dusko Pavlovic, and Paweł Sobociński. Refinement for signal flow graphs. In *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*, pages 24:1–24:16, 2017. `doi:10.4230/LIPIcs.CONCUR.2017.24`.

7 Filippo Bonchi, Joshua Holland, Robin Piedeleu, Pawel Sobocinski, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. *Proc. ACM Program. Lang.*, 3(POPL):25:1–25:28, 2019. `doi:10.1145/3290338`.

8 Filippo Bonchi, Joshua Holland, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. *Proceedings of the 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 3:1–28, 2019.

9 Filippo Bonchi, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Graphical affine algebra. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2019.

10 Filippo Bonchi, Jens Seeber, and Pawel Sobocinski. Graphical Conjunctive Queries. In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:23, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.CSL.2018.13`.

11 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A categorical semantics of signal flow graphs. In *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR)*, pages 435–450. Springer, 2014.

12 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Full abstraction for signal flow graphs. In *Proceedings of the 42nd Annual ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 515–526, 2015.

13 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. The calculus of signal flow diagrams I: linear relations on streams. *Information and Computation*, 252:2–29, 2017.

14 Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011.

15 Bob Coecke, Ross Duncan, Aleks Kissinger, and Quanlong Wang. Strong complementarity and non-locality in categorical quantum mechanics. In *LiCS 2012*, pages 245–254, 2012.

16 Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes - A first course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017.

17 Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252. ACM, 1977. `doi:10.1145/512950.512973`.

18 René David and Hassane Alla. *Discrete, Continuous, and Hybrid Petri Nets*. Springer, Berlin, 2 edition, 2010. `doi:10.1007/978-3-642-10669-9`.

19 Brendan Fong and David Spivak. String diagrams for regular logic (extended abstract). In John Baez and Bob Coecke, editors, *Applied Category Theory 2019*, volume 323 of *Electronic Proceedings in Theoretical Computer Science*, pages 196–229. Open Publishing Association, September 2020. `doi:10.4204/eptcs.323.14`.

20 Dan R Ghica and Achim Jung. Categorical semantics of digital circuits. In *Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 41–48, 2016.

21 Nathan Haydon and Paweł Sobociński. Compositional diagrammatic first-order logic. In *11th International Conference on the Theory and Application of Diagrams (DIAGRAMS 2020)*, 2020.

22 Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 52. ACM, 2014.

23 Bart Jacobs and Fabio Zanasi. The logical essentials of bayesian reasoning. *CoRR*, abs/1804.01193, 2018. `arXiv:1804.01193`.

24 Franciscus Rebro John C. Baez, Brandon Coya. Props in network theory. *CoRR*, abs/1707.08321, 2017. `arXiv:1707.08321`.

25 P Katis, N Sabadini, and RFC Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115(2):141–178, 1997.

26 Stephen Lack. Composing PROPs. *Theory and Application of Categories*, 13(9):147–163, 2004.

27 Yves Lafont. Towards an algebraic theory of Boolean circuits. *Journal of Pure and Applied Algebra*, 184(2–3):257–310, 2003.

28 Saunders Mac Lane. Categorical algebra. *Bulletin of the American Mathematical Society*, 71:40–106, 1965.

29 Samuel J Mason. *Feedback Theory: I. Some Properties of Signal Flow Graphs*. MIT Research Laboratory of Electronics, 1953.

30 Robin Piedeleu and Fabio Zanasi. A string diagrammatic axiomatisation of finite-state automata. In *FoSSaCS 2021*, 2021.

31 Peter Selinger. A survey of graphical languages for monoidal categories. *Springer Lecture Notes in Physics*, 13(813):289–355, 2011.

32 Fabio Zanasi. *Interacting Hopf Algebras: the theory of linear systems*. PhD thesis, Ecole Normale Supérieure de Lyon, 2015.

From Local to Global Determinacy in Concurrent Graph 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

In general, finite concurrent two-player reachability games are only determined in a weak sense: the supremum probability to win can be approached via stochastic strategies, but cannot be realized.

We introduce a class of concurrent games that are determined in a much stronger sense, and in a way, it is the largest class with this property. To this end, we introduce the notion of *local interaction* at a state of a graph game: it is a *game form* whose outcomes (i.e. a table whose entries) are the next states, which depend on the concurrent actions of the players. By definition, a game form is *determined* iff it always yields games that are determined via deterministic strategies when used as a local interaction in a Nature-free, one-shot reachability game. We show that if all the local interactions of a graph game with Borel objective are determined game forms, the game itself is determined: if Nature does not play, one player has a winning strategy; if Nature plays, both players have deterministic strategies that maximize the probability to win. This constitutes a clear-cut separation: either a game form behaves poorly already when used alone with basic objectives, or it behaves well even when used together with other well-behaved game forms and complex objectives.

Existing results for positional and finite-memory determinacy in turn-based games are extended this way to concurrent games with determined local interactions (CG-DLI).

2012 ACM Subject Classification Theory of computation

Keywords and phrases Concurrent games, Game forms, Local interaction

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.41

Related Version *Full Version*: https://arxiv.org/abs/2107.04081

1 Introduction

We consider games that involve two players and that are played on infinite (unless otherwise stated) graphs. On such games, we consider several flavors of determinacy properties. Specifically, the existence of a winning deterministic strategy for either of the players, of optimal deterministic strategies for both players, of almost-sure stochastic strategies for either of the players, and of ϵ-optimal stochastic strategies for both of the players. Generic determinacy results have been established on many classes of games. We illustrate these notions on turn-based and concurrent games with either a deterministic or stochastic Nature on Figures 1, 2, 3 and 4 (see [21] for the introduction of the most general setting, the stochastic concurrent games). In all cases the game starts in q_0 and the goal for Player A is to see y at some point while Player B wins if y never occurs.

Consider the turn-based game in Figure 1. There, Player A chooses either the self-loop or the edge to q_1; a symbol x, called a color, is seen in either case; then the game proceeds to state q_0 or q_1. In q_1 Player B chooses either the y-labeled self-loop or the x-labeled edge to q_0. This generates an infinite sequence over $\{x, y\}$. Player B has a winning strategy, which consists in never using the self-loop in q_1: however Player A may play, the generated sequence

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 41; pp. 41:1–41:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is x^ω. Thus, the game is said to be determined, in a very strong sense, and many sorts of objectives enjoy similar properties on such turn-based games. More generally, Martin [14, 15] proved that turn-based games with Borel objective are deterministically determined.

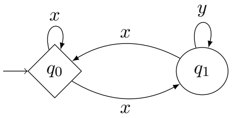


Figure 1 A turn-based game w/o Nature with diamond-shaped nodes for Player A, circle-shaped nodes for Player B and color labels on edges.

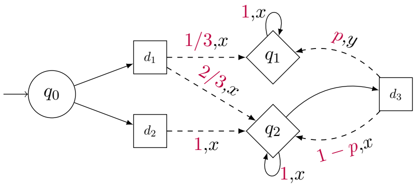


Figure 2 A turn-based game with Nature with probabilities displayed in purple on Nature-to-player edges, and colors in black on the same edges for convenience.

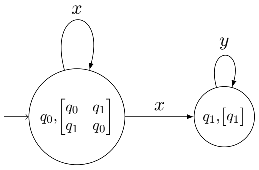


Figure 3 A concurrent game w/o Nature, with two actions for each player: Player A chooses a row, Player B chooses a column.

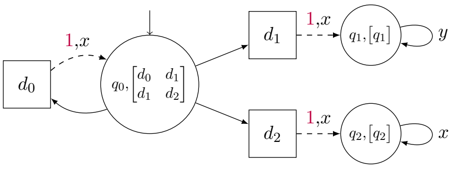


Figure 4 A concurrent game with Nature (albeit deterministic) with probabilities displayed in purple on Nature-to-player edges.

Now consider the turn-based game with (stochastic) Nature in Figure 2. In q_0 Player B moves to Nature state d_1 or d_2. In d_1 Nature goes to q_1 and q_2 with probability $\frac{1}{3}$ and $\frac{2}{3}$, respectively; in d_2 with probability 1 to q_2. In q_1 there is only a self-loop, which is a shorthand for an edge towards a Nature state that goes back to q_1 with probability 1. In q_2 Player A stays in q_2 or moves to d_3. In d_3 Nature goes to q_1 and q_2 with probabilities $p, 1-p \in]0,1[$. The edge (q_2, q_1) is labeled with y, and the other edges between the q_i are labeled with x. From q_0, Player B has a strategy that minimizes the probability to see y (to $\frac{2}{3}$), namely to play towards d_1; and Player A maximizes this probability (to the same value $\frac{2}{3}$) by playing d_3 when in q_2. Note that from q_2, this Player A's strategy wins almost surely but not surely. These optimal strategies of the Players are deterministic. The game is said to be determined, in a sense that is rather strong but weaker than above without Nature, and several objectives (though fewer than above) enjoy similar properties on turn-based games with Nature. More generally, it was proved [4, 23] that turn-based parity games played on finite graphs with stochastic Nature have deterministic optimal strategies.

Consider the game in Figure 3. The table depicted within state q_0 records the concurrent interaction between the two players at q_0: Player A chooses a row of the table while Player B independently chooses a column of the table; depending on the two choices, the game proceeds either to state q_0 again (first row first column, or second row second column) or to state q_1. In the two cases x is seen. In q_1 the interaction is trivial, i.e. each player has only one option, and y is seen. It is easy to see that Player A has no deterministic winning strategy, but a stochastic strategy that wins almost surely: in q_0, she picks each row with probability one half.

In the game from [8, 5] in Figure 4, Player A has no stochastic strategy that wins almost surely, but for all $\epsilon \in]0,1]$, she has a stochastic strategy that wins with a probability at least $1-\epsilon$: in q_0, she chooses the second row with probability ϵ. More generally, Martin [16] proved that such a weak determinacy holds in games with Borel objective if the local interactions involve finitely many rows and columns.

The above examples and existing results suggest that what prevents the existence of optimal strategies is more the structure of the local interaction rather than the presence of a stochastic Nature. This article substantiates this impression.

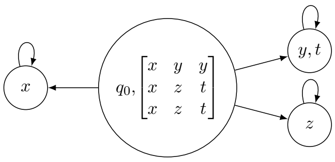

x
$q_0,$
$\begin{bmatrix} x & y & y \\ x & z & t \\ x & z & t \end{bmatrix}$
y,t
z

Figure 5 A concurrent game reachability game that is determined for every set of states as target set for either of the players.

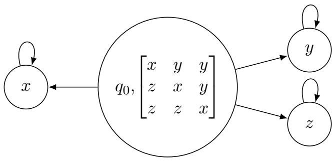

x
$q_0,$
$\begin{bmatrix} x & y & y \\ z & x & y \\ z & z & x \end{bmatrix}$
y
z

Figure 6 A concurrent reachability game that is not determined if a player tries to reach the set of states $T = \{x\}$.

Our contribution. A game form is a table whose entries are called outcomes, see e.g. Figures 5, 6, 7. By definition, it is determined if replacing each outcome with 1 or 0 yields a table with a row full of 1 (Player A wins) or a column full of 0 (Player B wins). It is easy to show that it is determined iff every "one-shot" reachability game using it as local interaction is deterministically determined. E.g. consider the one-shot reachability arena in Figure 5, involving a determined game form. Setting any subset of $\{x, y, z, t\}$ as target for either of the players yields a deterministically determined game. However, the game form in Figure 6 is non-determined, e.g. by setting $x := 1$ and $y, z := 0$. Equivalenty, setting the target of Player A to $\{x\}$ yields a game with no winning strategies. Thus, the determinacy of a game form amounts to its good behavior when used individually as local interaction in very simple games. We will show that individually well-behaved game forms are collectively well-behaved. More specifically, we extend various determinacy results from turn-based [14, 3, 4, 23, 11] to concurrent games with determined local interactions (CG-DLI). Fix a set K of colors. Each edge of our games is labeled with some color, and the winning objective is expressed as a subset of K^ω. We prove the following:

1. In all CG-DLI with Borel (parity) objective, one player has a (positional) winning strategy.
2. In all CG-DLI with Borel (parity) objective and stochastic Nature, both players have (positional) optimal strategies.
3. Let $\mathcal{M}$ be a memory skeleton (DFA on K, explained later). The following are equivalent.
   - W and $\mathsf{K}^\omega \setminus W$ are $\mathcal{M}$-monotone and $\mathcal{M}$-selective (notions recalled later).
   - All CG-DLI with finitely many states and actions, and objective W has a finite-memory winning strategy implemented via $\mathcal{M}$.

Moreover in the three statements above, the winning/optimal strategies can be chosen both deterministic and dependent only on the history of observed colors, rather than visited states.

Conversely, let G be any non-determined game form. As hinted at above, one can show that for all Borel objectives $\emptyset \subsetneq W \subsetneq \mathsf{K}^\omega$, there is a Nature-free game with one single non-trivial state whose local interaction is G, and no deterministic optimal (or winning) strategy, even one that would depend on the history of visited states. A similar result holds for finite-memory strategies. Hence, these results provide a clear-cut separation: determined game forms are well-behaved basic bricks that collectively build well-behaved-only concurrent games, while non-determined game forms are ill-behaved already when used alone.

A large part of the proofs of the above extensions is factored out by our following theorem: a CG-DLI is (finite-memory, positionnaly, "plainly") determined (via winning deterministic or optimal stochastic strategies) if and only if its sequential version is. The sequential version of the game is obtained by letting one player (whichever, but keep the convention) act first at each state of the game, and the opponent act second. Although most of the extensions are straightforward applications of this theorem, the finite-memory case is different: the result in [3] requires the objective to satisfy specific properties, and it is rather long to prove that these properties satisfy the assumptions of the theorem.

Outline. Section 2 contains notations; Section 3 recalls the notion of game form; Section 4 presents the game-theoretic formalism; Section 5 defines the sequentialization and parallelization, and proves related preservation results; Section 6 presents determinacy extensions.

Additional details and complete proofs are available in the arXiv version of this paper [1].

2 Preliminaries

Consider a non-empty set D. We denote by $D^{\uparrow} := D^* \cup D^{\omega}$ the set of finite or infinite sequences in D. For a sequence $\pi = \pi_0\pi_1 \ldots \pi_n \in D^*$, we denote by $\mathsf{lt}(\pi)$ the last element of the sequence: $\mathsf{lt}(\pi) = \pi_n$.

For a function $\mathsf{f} : E \to F$ and $F' \subseteq F$, the notation $\mathsf{f}^{-1}[F']$ refers to the preimage $\{e \in E \mid \mathsf{f}(e) \in F'\}$ of F' by the function f. Furthermore, a function $\mathsf{f} : E \to F$ can be lifted into a function $\bar{\mathsf{f}} : E^{\uparrow} \to F^{\uparrow}$ defined by: $\bar{\mathsf{f}}(\varepsilon) = \varepsilon$, $\bar{\mathsf{f}}(e) = \mathsf{f}(e)$ for all $e \in E$, and $\bar{\mathsf{f}}(\pi \cdot \pi') = \bar{\mathsf{f}}(\pi) \cdot \bar{\mathsf{f}}(\pi')$ for all $\pi \in E^*$ and $\pi' \in E^{\uparrow}$. For a set $E' \subseteq E$, we define the projection function $\phi_{E,E'} : E^{\uparrow} \to E'^{\uparrow}$ such that $\phi_{E,E'}(e) = e$ if $e \in E'$, $\phi_{E,E'}(e) = \varepsilon$ otherwise and $\phi_{E,E'}(\pi \cdot \pi') = \phi_{E,E'}(\pi) \cdot \phi_{E,E'}(\pi')$ for all $\pi \in E^*$ and $\pi' \in E^{\uparrow}$. For a set Q and a function $\mathsf{f} : Q \times Q \to T$, we denote by $\mathsf{tr}_{\mathsf{f}} : Q^+ \to T^* \times Q$ the function that associates to a sequence $\pi \in Q^+$, its trace $\mathsf{tr}_{\mathsf{f}}(\pi) = (\bar{\mathsf{f}}(\pi), \mathsf{lt}(\pi))$. For instance, $\mathsf{tr}_{\mathsf{f}}(a \cdot b \cdot c) = (f(a,b) \cdot f(b,c), c)$.

Let us now recall the definition of cylinder sets. For a non-empty set Q, for all $\pi \in Q^*$, the cylinder set $\mathsf{Cyl}(\pi)$ generated by π is the set $\mathsf{Cyl}(\pi) = \{\pi \cdot \rho \in Q^{\omega} \mid \rho \in Q^{\omega}\}$. We denote by Cyl_Q the set of all cylinder sets on Q^{ω}. The open sets of Q^{ω} are the sets equal to an arbitrary union of cylinder sets. The set of Borel sets on Q^{ω}, denoted $\mathsf{Borel}(Q)$, is then equal to the smallest set containing all open sets that is closed under complementation and countable union. Recall that, considering two probability measures $\nu, \nu' : \mathsf{Borel}(Q) \to [0,1]$, if for all $C \in \mathsf{Cyl}_Q$, we have $\nu(C) = \nu'(C)$, then $\nu = \nu$.

3 Game Forms and Win/Lose Games

Informally, game forms (used in [10, 12]) are games without objectives, see Definition 1 and examples in Figure 7. They are similar to what is sometimes called arena, but are presented in normal form, i.e. by ignoring their possible underlying graph or tree structure.

▶ **Definition 1** (Game form and win/lose game). *A* game form *is a tuple $\mathcal{F} = \langle \mathsf{S_A}, \mathsf{S_B}, \mathsf{O}, \varrho \rangle$ where $\mathsf{S_A}$ (resp. $\mathsf{S_B}$) is the non-empty set of strategies available to Player A (resp. B), O is a non-empty set of possible outcomes, and $\varrho : \mathsf{S_A} \times \mathsf{S_B} \to \mathsf{O}$ is a function that associates an outcome to each pair of strategies. A* win/lose game *is a pair $\mathcal{G} = \langle \mathcal{F}, \mathcal{V} \rangle$ where $\mathcal{F}$ is a game form and $\mathcal{V} \subseteq \mathsf{O}$ is the set of winning outcomes for Player A whereas $\mathsf{O} \setminus \mathcal{V}$ is the set of winning outcomes for Player B.*

As in Definition 1, a player wins if she obtains an outcome that makes her win, hence winning for Player A means reaching an outcome in $\mathcal{V}$, whereas winning for Player B means reaching an outcome in $\mathsf{O} \setminus \mathcal{V}$. So, one player wins if and only if the other player loses, hence the terminology. In the context of win/lose games, we can define the notion of winning strategy, that is, a strategy for a player that ensures winning regardless of his opponent's strategy. The definition of determinacy follows.

$$I_1 = \begin{bmatrix} x & y \\ y & x \end{bmatrix} \quad I_2 = \begin{bmatrix} x & x \\ y & z \end{bmatrix} \quad I_3 = \begin{bmatrix} x & x & z \\ x & y & y \\ z & y & z \end{bmatrix} \quad I_4 = \begin{bmatrix} x & y & z \\ y & x & z \\ z & z & z \end{bmatrix} \quad I_5 = \begin{bmatrix} x & x & z \\ x & z & y \\ z & y & y \end{bmatrix}$$

Figure 7 Five game forms: I_1 and I_5 are not determined, whereas I_2, I_3, and I_4 are.

▶ **Definition 2** (Winning Strategies and Determinacy)**.** *Consider a game form* $\mathcal{F} = \langle \mathsf{S_A}, \mathsf{S_B}, \mathsf{O}, \varrho \rangle$ *and a subset of outcomes* $\mathcal{V} \subseteq \mathsf{O}$*. In the win/lose game* $\mathcal{G} = \langle \mathcal{F}, \mathcal{V} \rangle$*, a* winning strategy $\mathsf{s_A} \in \mathsf{S_A}$ *(resp.* $\mathsf{s_B} \in \mathsf{S_B}$*) for Player* A *(resp.* B*) is a strategy such that, for all* $\mathsf{s_B} \in \mathsf{S_B}$ *(resp.* $\mathsf{s_A} \in \mathsf{S_A}$*), we have* $\varrho(\mathsf{s_A}, \mathsf{s_B}) \in \mathcal{V}$ *(resp.* $\mathsf{O} \setminus \mathcal{V}$*). We write* $\mathcal{W}_\mathsf{A}(\mathcal{F}, \mathcal{V})$ *(resp.* $\mathcal{W}_\mathsf{B}(\mathcal{F}, \mathsf{O} \setminus \mathcal{V})$*) the set of winning strategies for Player* A *(resp. Player* B*) with objective* $\mathcal{V}$ *(resp.* $\mathsf{O} \setminus \mathcal{V}$*). The win/lose game* $\mathcal{G}$ *is* determined *if either of the players has a winning strategy:* $\mathcal{W}_\mathsf{A}(\mathcal{F}, \mathcal{V}) \cup \mathcal{W}_\mathsf{B}(\mathcal{F}, \mathsf{O} \setminus \mathcal{V}) \neq \emptyset$*. The game form* $\mathcal{F}$ *is said to be* determined *if, for all* $\mathcal{V} \subseteq \mathsf{O}$*, the win/lose game* $\mathcal{G} = \langle \mathcal{F}, \mathcal{V} \rangle$ *is determined. We denote by* $\mathsf{Det_{GF}}$ *the set of determined game forms.*

▶ **Example 3.** Consider the game forms represented in Figure 7. We argue that I_2, I_3, and I_4 are determined, while I_1 and I_5 are not. Consider any subset $\mathcal{V}$ of the outcomes and, in I_2, I_3, and I_4, replace each occurence of outcome in $\mathcal{V}$ with $\mathsf{w_A}$ (indicating winning outcomes for Player A) and the others with $\mathsf{w_B}$ (indicating winning outcomes for Player B). There is always a row of $\mathsf{w_A}$ or a column of $\mathsf{w_B}$, so these game forms are determined. However, rewriting x with $\mathsf{w_A}$ and y with $\mathsf{w_B}$ in I_1 yields the well-known matching-penny game, which clearly has no winning strategies. Similarly, rewriting z with $\mathsf{w_A}$ and x, y with $\mathsf{w_B}$ in I_5 leads to no row full of $\mathsf{w_A}$ and no column full of $\mathsf{w_B}$.

As we shall see, determined game forms are exactly the game forms that share enough similarities with "turn-based interactions", so that our determinacy transfer may hold. Hence, we may ask whether the determined game forms are nothing but turn-based interactions in disguise. Of course, the answer depends on what we mean by "in disguise". For a natural notion of being similar to a turn-based interaction, the answer is negative. Thus, determined game forms are more than turn-based interactions.

In addition to the toy examples in Figure 7, let us exemplify that determined game forms arise naturally in computer science. A parity game ([7, 17, 22]) is defined on a priority arena, i.e. a graph where each vertex is controlled by one player and every edge is labeled with a natural number less than a fixed bound. The outcome of an infinite run in such an arena is the maximum of all the numbers that occur infinitely often during the run. If the priorities are seen not as concrete numbers but as abstract outcomes, the priority arena can be seen as a game form. By a slight generalization of [7, 17, 22] described, e.g., in [19, Corollary 3.8], it is moreover a determined game form. So, as we shall see, choosing the next state following a local interaction given by a parity game will be a *well-behaved* interaction.

Finally, consider the complexity of deciding if a given game form is determined (the corresponding decision problem is denoted DetGF). It is straightforwardly in coNP since proving that a game form is not determined amounts to exhibiting a $\{\mathsf{w_A}, \mathsf{w_B}\}$-valuation for which there is neither a row full of $\mathsf{w_A}$s nor a column full of $\mathsf{w_B}$s, which can be checked in polynomial time. In fact, in [2] (where determinacy is refered to as tightness), the authors mentioned that DetGF could be solved in quasi-polynomial time via a reduction to the dualization of monotone CNF formulae (denoted $\mathsf{MonotoneDual}$), which can be solved in quasi-polynomial time [9]. Note that it is an open problem if $\mathsf{MonotoneDual}$ is in P or is coNP-complete [6]. In fact, we can show that DetGF is equivalent (modulo polytime reduction) to $\mathsf{MonotoneDual}$ thus showing that answering if DetGF is in P or coNP-complete directly answers the same question for $\mathsf{MonotoneDual}$.

4 Concurrent Graph Games and Strategies

Colored stochastic win/lose concurrent graph games. Informally, a stochastic concurrent game is played on a graph as follows: from a given state, both players simultaneously choose an action, and the next state is set according to a probability distribution that depends on the two actions. We want to consider the ways the two players interact at each state (which we call the local interactions of the game) as game forms. To facilitate this, we decouple the concurrent interaction of the players from the stochastic choice of Nature; we therefore add intermediate states belonging to Nature, and ensure that they do not impact winning conditions by assigning colors to ordered pairs of player states, thus hiding the Nature states that are visited. To sum up, the outcome of an interaction of the players is a Nature state from which the next (relevant) state of the game is chosen via a probability distribution.

▶ **Definition 4** (Stochastic concurrent games). *A colored* stochastic concurrent graph arena $\mathcal{C}$ *is a tuple* $\langle A, B, Q, q_0, \mathsf{D}, \delta, \mathsf{dist}, \mathsf{K}, \mathsf{col}\rangle$ *where* A *(resp.* B*) is the non-empty set of actions available to Player* A *(resp.* B*),* Q *is the (non-empty) set of states,* $q_0 \in Q$ *is the initial state,* D *is the set of Nature states,* $\delta : Q \times A \times B \to \mathsf{D}$ *is the transition function,* $\mathsf{dist} : \mathsf{D} \to \mathsf{Dist}(Q)$ *is the distribution function,* K *is a non-empty set of colors, and* $\mathsf{col} : Q \times Q \to \mathsf{K}$ *is a coloring function. The composition of the transition and distribution functions* $\mathsf{dist} \circ \delta : Q \times A \times B \to \mathsf{Dist}(Q)$ *will be denoted* Δ*. A* win/lose concurrent graph game *is a pair* $\langle \mathcal{C}, W\rangle$ *where* $W \in \mathsf{Borel}(\mathsf{K})$ *is the set of winning sequences of colors (for Player* A*).*

In the following, the arena $\mathcal{C}$ will always refer to the tuple $\langle A, B, Q, q_0, \mathsf{D}, \delta, \mathsf{dist}, \mathsf{K}, \mathsf{col}\rangle$ unless otherwise stated. In section 6, we will be able to apply some of our results only to finite arenas, i.e. when A, B and $Q \cup \mathsf{D}$ are finite.

Strategies and their values. We consider two kinds of strategies: those that only depend on the sequence of colors seen (and the current state) and that output a specific action – called chromatic strategies [13] – and those that may depend on the sequence of states seen and that output a distribution over the available actions – called state strategies.

▶ **Definition 5** (State and chromatic strategies). *Let* $\mathcal{C}$ *be an arena.*

- *A state strategy, for Player* A *is a function* $\mathsf{s_A} : Q^+ \to \mathsf{Dist}(A)$ *and the set of all such strategies in arena* $\mathcal{C}$ *for that player is denoted* $\mathsf{StaSt}^{\mathsf{A}}_{\mathcal{C}}$*.*
- *A chromatic strategy for Player* A *is a function* $\mathsf{s_A} : \mathsf{K}^* \times Q \to A$ *and the set of all such strategies in arena* $\mathcal{C}$ *for that player is denoted* $\mathsf{ColSt}^{\mathsf{A}}_{\mathcal{C}}$*. From a chromatic strategy* $\mathsf{s_A} \in \mathsf{ColSt}^{\mathsf{A}}_{\mathcal{C}}$*, we can extract the state strategy* $\widetilde{\mathsf{s_A}} : Q^+ \to \mathsf{Dist}(A)$ *defined by* $\widetilde{\mathsf{s_A}} = \mathsf{s_A} \circ \mathsf{tr_{col}}$*.*

The definitions are likewise for Player B*. Two state strategies* $\mathsf{s_A}$ *and* $\mathsf{s_B}$ *for Players* A *and* B *then induce a probability of occurrence of finite paths and, following, of cylinder sets. This, in turn, induces a probability measure* $\mathbb{P}^{\mathcal{C}}_{\mathsf{s_A},\mathsf{s_B}}$ *over all Borel sets.*

In a game $\langle \mathcal{C}, W\rangle$, Player A tries to maximize the probability to be in the set W whereas Player B tries to minimize it. We will show that the concurrent games we consider are determined and that chromatic strategies are sufficient to play optimally. However, since the games considered are stochastic, for a strategy to be optimal, it has to achieve the optimal value against all strategies – i.e. state strategies – of the antagonist player. For convenience in the proofs, we give below this assymetric definition of values, where one player plays with chromatic strategies while the other is allowed to use state strategies. This is without restriction as we will be able to prove that the color values of the two players coincide.

▶ **Definition 6** (Value of strategies and color value of the game)**.** *Let $\mathcal{C}$ be an arena. The corresponding winning set for Player* A *to a Borel set $W \subseteq \mathsf{K}^\omega$ is equal to $U_W = \overline{\mathsf{col}}^{-1}[W] \subseteq Q^\omega$, which is also Borel.*[1] *Let $\mathsf{s_A} \in \mathsf{ColSt}^{\mathsf{A}}_{\mathcal{C}}$ be a chromatic strategy for Player* A*. The* value *of strategy $\mathsf{s_A}$ is equal to $\chi^{\mathcal{C}}_{\mathsf{s_A}}[W] := \inf_{\mathsf{s_B} \in \mathsf{StaSt}^{\mathsf{B}}_{\mathcal{C}}} \mathbb{P}^{\mathcal{C}}_{\mathsf{s_A},\mathsf{s_B}}[U_W]$. The* color value *$\chi^{\mathcal{C}}_{\mathsf{A}}$ of the game for Player* A *is: $\chi^{\mathcal{C}}_{\mathsf{A}}[W] := \sup_{\mathsf{s_A} \in \mathsf{ColSt}^{\mathsf{A}}_{\mathcal{C}}} \chi^{\mathcal{C}}_{\mathsf{s_A}}[W]$. The definitions are likewise for Player* B*, by reversing the supremum and infimum.*

A win/lose stochastic concurrent graph game $\langle \mathcal{C}, W \rangle$ is limit-determined *if $\chi^{\mathcal{C}}_{\mathsf{A}}[W] = \chi^{\mathcal{C}}_{\mathsf{B}}[W]$. If in addition there are strategies $\mathsf{s_A} \in \mathsf{ColSt}^{\mathsf{A}}_{\mathcal{C}}$ and $\mathsf{s_B} \in \mathsf{ColSt}^{\mathsf{B}}_{\mathcal{C}}$ such that $\chi^{\mathcal{C}}_{\mathsf{s_A}}[W] = \chi^{\mathcal{C}}_{\mathsf{A}}[W]$ and $\chi^{\mathcal{C}}_{\mathsf{s_B}}[W] = \chi^{\mathcal{C}}_{\mathsf{B}}[W]$, we say that the game is* determined*. In this case, such strategies are called* optimal *strategies.*

Let us look at what the *local determinacy* of a concurrent game refers to, which will yield the definition of locally determined stochastic concurrent games.

▶ **Definition 7** (Local interactions)**.** *The* local interaction *in a stochastic concurrent graph arena $\mathcal{C}$ at state $q \in Q$ is the game form $\mathcal{F}_q = \langle A, B, \mathsf{D}, \delta(q,\cdot,\cdot) \rangle$ where the strategies available for Player* A *(resp.* B*) are the actions in A (resp. B) and the outcomes are the Nature states. For a set of game forms $\mathcal{I}$, we say that a concurrent arena $\mathcal{C} = \langle A, B, Q, q_0, \mathsf{D}, \delta, \mathsf{dist}, \mathsf{K}, \mathsf{col} \rangle$ is* built on *$\mathcal{I}$ if, for all $q \in Q$, we have $\mathcal{F}_q \in \mathcal{I}$ (up to a renaming of the outcomes). A stochastic concurrent graph arena/game is* locally determined *if it is built on* $\mathsf{Det_{GF}}$.

Turn-based games. Usually, turn-based games and concurrent games are described in two different formalisms. Indeed, in a turn-based game, a player plays only in the states that she controls, whereas in a concurrent game, in each state both players play an action and subsequently the next (Nature) state is reached. However, turn-based games can be seen as a special case of concurrent games, where at each state, the next (Nature) state is chosen regardless of one of the player's action. We choose the second option .

Section 5 will translate locally determined concurrent games into turn-based games, then transfer existing determinacy results on turn-based games back into extension results for the more general locally determined concurrent games.

Chromatic strategy implementations. We recall the notion of memory skeleton (see, for instance, [3]) and we see how it can implement the chromatic strategies. For a set of colors K and a set of states Q, a memory skeleton on K is a triple $\mathcal{M} = \langle M, m_{\text{init}}, \mu \rangle$, where M is a non-empty set called the memory, $m_{\text{init}} \in M$ is the initial state of the memory and $\mu : M \times \mathsf{K} \to M$ is the update function. An action map with memory M is a function $\lambda : M \times Q \to T$ for a non-empty set T. Note that T is a set of possible decisions that can be made. Here, T will be instantiated with the set of actions of either of the players. In fact, a memory skeleton and an action map implement a chromatic strategy.

▶ **Definition 8** (Implementation of strategies)**.** *Consider a concurrent colored arena $\mathcal{C}$, a player $p \in \{\mathsf{A}, \mathsf{B}\}$ and the corresponding set of actions $T \in \{A, B\}$. A memory skeleton $\mathcal{M} = \langle M, m_{\text{init}}, \mu \rangle$ on K and an action map $\lambda : M \times Q \to T$ implement the chromatic strategy $\mathsf{s} : \mathsf{K}^* \times Q \to T$ that is defined by $\mathsf{s}(\rho, q) = \lambda(\overline{\mu}(m_{\text{init}}, \rho), q) \in T$ for all $(\rho, q) \in \mathsf{K}^* \times Q$.*

A strategy s is finite memory *if there exists a memory skeleton $\mathcal{M} = \langle M, m_{\text{init}}, \mu \rangle$, with M finite, and an action map λ implementing s. If M is reduced to a singleton, s is* positional*, aka memoryless. The amount of memory used to implement the strategy s is $|M|$.*

[1] As the preimage of a Borel set by the continuous function $\overline{\mathsf{col}}$.

Note that any chromatic strategy $\mathsf{s} : \mathsf{K}^* \times Q \to T$ can be implemented with a (possibly infinite) memory skeleton and an action map: consider the memory skeleton $\mathcal{M} = \langle \mathsf{K}^*, \epsilon, \mu \rangle$ where $\mu : \mathsf{K}^* \times \mathsf{K} \to \mathsf{K}^*$ is defined by $\mu(\rho, k) = \rho \cdot k$ for all $\rho \in \mathsf{K}^*$ and $k \in \mathsf{K}$.

▶ **Definition 9** (Finite-memory determinacy). *A game is said to be* finite-memory *(resp.* positionally*) determined if it is determined and optimal strategies can be found among finite-memory (resp. positional) strategies.*

5 Sequentialization of Games

In this section, we explain how we sequentialize a concurrent graph game. We then show "correctness" of this sequentialization in a sense that we will make precise.

Sequential version of a concurrent graph game. The sequential version of an arbitrary colored stochastic concurrent graph arena consists of a turn-based graph arena where Player A plays first and then Player B responds.

▶ **Definition 10** (Sequentialization of a concurrent arena and game). *Consider a concurrent arena $\mathcal{C} = \langle A, B, Q, q_0, \mathsf{D}, \delta, \mathsf{dist}, \mathsf{K}, \mathsf{col} \rangle$ and an objective $W \in \mathsf{Borel}(\mathsf{K})$.*

- *The* sequential version *of $\mathcal{C}$ is the turn-based arena $\mathsf{Seq}(\mathcal{C}) = \langle A, B, V, q_0, \mathsf{D}_\mathsf{A} \uplus \mathsf{D}_\mathsf{B}, \delta_\mathcal{C}, \mathsf{dist}_\mathcal{C}, \mathsf{K}_\mathcal{C}, \mathsf{col}_\mathcal{C} \rangle$ where $V = V_\mathsf{A} \uplus V_\mathsf{B}$ with $V_\mathsf{A} = Q$ and $V_\mathsf{B} = Q \times A$, $\mathsf{D}_\mathsf{A} = V_\mathsf{B}$ and $\mathsf{D}_\mathsf{B} = \mathsf{D}$. Furthermore, for all $q \in V_\mathsf{A}$, $a \in A$ and $b \in B$, we have $\delta_\mathcal{C}(q, a, b) = (q, a) \in V_\mathsf{B} = \mathsf{D}_\mathsf{A}$ and $\mathsf{dist}_\mathcal{C}((q, a))[(q, a)] = 1$. In addition, for all $d \in \mathsf{D}$, we have $\mathsf{dist}_\mathcal{C}(d) = \mathsf{dist}(d)$ and for all $a' \in A$, $b \in B$, and $(q, a) \in V_\mathsf{B}$ we have $\delta_\mathcal{C}((q, a), a', b) = \delta(q, a, b) \in \mathsf{D} = \mathsf{D}_\mathsf{B}$. Finally, we have $\mathsf{K}_\mathcal{C} = \mathsf{K} \cup \{k_\mathcal{C}\}$ for some fresh color $k_\mathcal{C} \notin \mathsf{K}$ and $\mathsf{col}_\mathcal{C}(q, (q, a)) = k_\mathcal{C}$ if $q \in V_\mathsf{A}$ and $(q, a) \in V_\mathsf{B}$ and $\mathsf{col}_\mathcal{C}((q, a), q') = \mathsf{col}(q, q')$ if $(q, a) \in V_\mathsf{B}$ and $q' \in V_\mathsf{A}$. The function $\mathsf{col}_\mathcal{C}$ is defined arbitrarily on other pairs of states.*
- *The* sequential version *of the concurrent game $\langle \mathcal{C}, W \rangle$ is the turn-based game $\langle \mathsf{Seq}(\mathcal{C}), \mathsf{Seq}(W) \rangle$, where $\mathsf{Seq}(W) = (\phi_{\mathsf{K}_\mathcal{C}, \mathsf{K}})^{-1}[W]$ is the preimage of the winning set W by the projection function $\phi_{\mathsf{K}_\mathcal{C}, \mathsf{K}} : \mathsf{K}_\mathcal{C}^\uparrow \to \mathsf{K}^\uparrow$.*

In the above definition, one can notice that the states in V_A belong to Player A whereas states in V_B belong to Player B.

▶ **Example 11.** Sequentialization of an arena is a rather simple operation that we illustrate in Figure 8. Note that the initial concurrent arena (from Figure 4) has deterministic Nature (all probabilities that appear equal 1), and the sequential version also does. From q_0, Player A selects either the first row (top choice in the figure) or the second row (bottom choice in the figure), then Player B selects one of the options, i.e. one of the next states offered in the subset – this corresponds to choosing a column in the game form. The fresh color $k_\mathcal{C}$ appear after the choice of Player A, and the original colors appear after the choice of Player B.

One can notice here that in the original concurrent game and its sequential version, the value of the game for the players are different: in the turn-based game, from q_0, Player B has a strategy to ensure never seeing the color y (which induces a value of 0 for Player B) whereas it is not the case in the original game. As we will see along that paper, this is due to the fact that the local interaction at q_0 is not determined.

We make several remarks: paths in a concurrent arena and in its sequential version relate via a projection and, if W is Borel, so is $\mathsf{Seq}(W)$ as the continuous preimage of a Borel set.

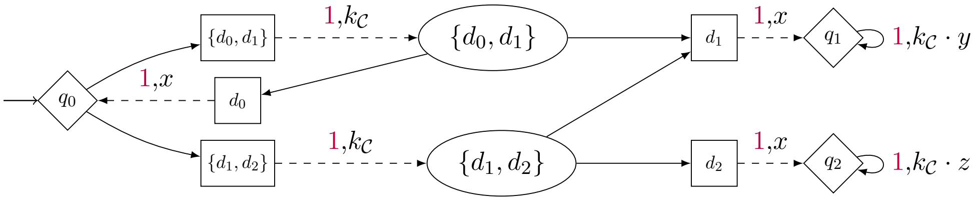


Figure 8 Sequentialization of the concurrent arena from Figure 4. Diamond-shaped nodes belong to Player A, ellipse-shaped ones belong to Player B and the rectangle-shaped are Nature states. On the edges, probabilities appear in purple and colors in black. The pairs in $Q \times A$ are represented as the corresponding set of states $\delta(q, a, B) \subseteq \mathcal{P}(\mathsf{D})$.

Main theorem. We now state the main result, and discuss it in the rest of Section 5.

▶ **Theorem 12.** *Consider a concurrent game $\langle \mathcal{C}, W\rangle$ and assume that it is locally determined. Then, it is (resp. finite-memory, resp. positionnaly) determined if and only if its sequential version $\langle \mathsf{Seq}(\mathcal{C}), \mathsf{Seq}(W)\rangle$ is (resp. finite-memory, resp. positionnaly) determined.*

We assume for the rest of the section that $\langle \mathcal{C}, W\rangle$ with $\mathcal{C} = \langle A, B, Q, q_0, \mathsf{D}, \delta, \mathsf{dist}, \mathsf{K}, \mathsf{col}\rangle$ is a concurrent graph game, and $\langle \mathsf{Seq}(\mathcal{C}), \mathsf{Seq}(W)\rangle$ with $\mathsf{Seq}(\mathcal{C}) = \langle A, B, V, q_0, \mathsf{D}_\mathsf{A} \uplus \mathsf{D}_\mathsf{B}, \delta_\mathcal{C}, \mathsf{dist}_\mathcal{C}, \mathsf{K}_\mathcal{C}, \mathsf{col}_\mathcal{C}\rangle$, is its sequential version.

The idea of the proof is to show that the same values are achieved by both players in the two games while preserving the memory which is used (memory skeletons have to be slightly adapted to take care of removing the new color $k_\mathcal{C}$ used in $\mathsf{Seq}(\mathcal{C})$). To do so:

- given a chromatic strategy s (for either of the players) in the concurrent game $\mathcal{C}$, build a sequentialized version $\mathsf{Seq}(\mathsf{s})$ (which is also a chromatic strategy) in $\mathsf{Seq}(\mathcal{C})$ that is at least as good in $\mathsf{Seq}(\mathcal{C})$ as s is in $\mathcal{C}$. For both players, this is not difficult to achieve since histories in $\mathsf{Seq}(\mathcal{C})$ give at least as much information for taking a decision as in $\mathcal{C}$; it is even the case that Player B has more information in $\mathsf{Seq}(\mathcal{C})$ since she plays second but she doesn't use it.
- given a chromatic strategy σ (for either of the players) in the sequential game $\mathsf{Seq}(\mathcal{C})$, build a parallelized version $\mathsf{Par}(\sigma)$ (which is also a chromatic strategy) in $\mathcal{C}$ that is at least as good in $\mathcal{C}$ as σ is in $\mathsf{Seq}(\mathcal{C})$. This is not difficult to prove (while preserving the same memory) for Player A. Indeed, she has the same information in both games on histories when she has to take a decision (removing $k_\mathcal{C}$'s). The case of Player B is very different since she plays second, hence has more information in the sequentialized version than in the original concurrent game. The next paragraph is dedicated to this case, highlighting the role of the local determinacy hypothesis.

Overall, we obtain that the values of the concurrent game $\langle \mathcal{C}, W\rangle$ and its sequential version $\langle \mathsf{Seq}(\mathcal{C}), \mathsf{Seq}(W)\rangle$ are equal for both players, which implies Theorem 12. Formally:

▶ **Theorem 13.** *If the game $\langle \mathcal{C}, W\rangle$ is locally determined, we have the following:*

- $\chi_\mathsf{A}^\mathcal{C}[W] = \chi_\mathsf{A}^{\mathsf{Seq}(\mathcal{C})}[\mathsf{Seq}(W)]$ *and* $\chi_\mathsf{B}^\mathcal{C}[W] = \chi_\mathsf{B}^{\mathsf{Seq}(\mathcal{C})}[\mathsf{Seq}(W)]$*;*
- *the game $\langle \mathcal{C}, W\rangle$ is limit-determined iff its sequential version $\langle \mathsf{Seq}(\mathcal{C}), \mathsf{Seq}(W)\rangle$ is;*
- *if a chromatic strategy s is optimal in $\langle \mathcal{C}, W\rangle$, so is $\mathsf{Seq}(s)$ is in $\langle \mathsf{Seq}(\mathcal{C}), \mathsf{Seq}(W)\rangle$;*
- *if a chromatic strategy σ is optimal in $\langle \mathsf{Seq}(\mathcal{C}), \mathsf{Seq}(W)\rangle$, so is $\mathsf{Par}(\sigma)$ is in $\langle \mathcal{C}, W\rangle$.*

Parallelization of the strategy of Player B. We now give arguments for the difficult case, the preservation of the value for Player B from $\mathsf{Seq}(\mathcal{C})$ back to $\mathcal{C}$.

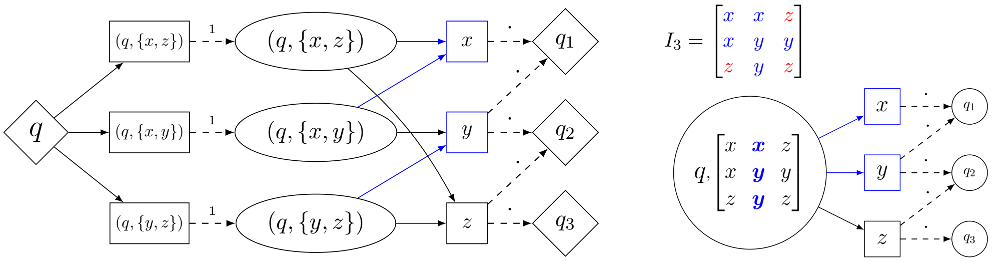

q
$(q,\{x,z\})$
$(q,\{x,y\})$
$(q,\{y,z\})$
1
1
1
$(q,\{x,z\})$
$(q,\{x,y\})$
$(q,\{y,z\})$
x
y
z
q_1
q_2
q_3
$I_3 = \begin{bmatrix} x & x & z \\ x & y & y \\ z & y & z \end{bmatrix}$
$q, \begin{bmatrix} x & x & z \\ x & y & y \\ z & y & z \end{bmatrix}$
x
y
z
q_1
q_2
q_3

Figure 9 On the left-hand side, we have a portion of a turn-based graph arena, with all the states reachable in at most two steps from the state q. This turn-based arena corresponds to the sequentialization of the portion of the concurrent arena on the right-hand side with the local interaction in state q being I_3 from Figure 7. Out of q, Player A has three choices (corresponding to the three rows), hence the three outgoing edges; leading to three Nature states from which a specific state belonging to Player B is reached with probability 1. From each of these three states, Player B has two choices, leading to two out of the three x, y and z Nature states. A strategy for Player B is represented in blue arrows in the turn-based arena, with the Nature states reachable with that strategy represented in blue. It is done similarly in the local interaction I_3, with the state that is not reachable, i.e. z, in red. Finally, in the concurrent arena, the blue states are the Nature states reachable if Player B opts for the second column, which is the winning strategy for Player B in the win/lose game obtained from the game form I_3 if she has $\{x, y\}$ as winning set.

Consider a strategy σ for Player B in $\mathsf{Seq}(\mathcal{C})$. Such a strategy takes a finite sequence of colors in $\mathsf{K}_\mathcal{C}$ and the current vertex $(q, a) \in Q \times A$ to make the next decision, where a is the last action played by Player A. We assume that σ is implemented by a memory skeleton $\mathcal{M} = \langle M, m_{\text{init}}, \mu\rangle$ on $\mathsf{K}_\mathcal{C}$ and an action map $\lambda : M \times V_\mathsf{B} \to B$. The parallelization $\mathsf{Par}(\sigma)$ will be implemented by the memory skeleton $\mathsf{Par}(\mathcal{M})$ and the action map $\mathsf{Par}(\lambda)$, where $\mathsf{Par}(\mathcal{M}) = \langle M, m_{\text{init}}, \mathsf{Par}(\mu)\rangle$ only adds occurrences of $k_\mathcal{C}$: $\mathsf{Par}(\mu)(m, k) = \mu(\mu(m, k_\mathcal{C}), k)$. The parallelization $\mathsf{Par}(\lambda)\colon M \times Q \to B$ of the action map $\lambda\colon M \times V_\mathsf{B} \to B$ is more difficult to define since λ has more information than is supposed to have $\mathsf{Par}(\lambda)$.

Since our goal is to ensure that the value of the game does not worsen, we want the new strategy to ensure that the Nature states reachable in $\mathcal{C}$ with the parallel version of the action map are also reachable in $\mathsf{Seq}(\mathcal{C})$ with the original action maps: that way, every path that can be generated with some probability in the concurrent game could also be generated (up to projection) with the same probability in the turn-based game.

We fix a memory state $m \in M$ and a state $q \in V_\mathsf{A} = Q$ in $\mathsf{Seq}(\mathcal{C})$. We define $\mathsf{Rch}^\sigma_{m,q} = \{\delta(q, a, \lambda(\mu(m, k_\mathcal{C}), (q, a))) \mid a \in A\} \subseteq \mathsf{D}$ the set of Nature states that can be reached in two steps when applying strategy σ from memory state m and state q in $\mathsf{Seq}(\mathcal{C})$, taking into account all possible choices of Player A. The crux of the construction relies Lemma 14.

▶ **Lemma 14.** *If the local interaction $\mathcal{F}_q$ is determined, $\mathcal{W}_\mathsf{B}(\mathcal{F}_q, \mathsf{Rch}^\sigma_{m,q}) \neq \emptyset$.*

Proof. Consider an action $a \in A$. There exists $b \in B$ such that $\delta(q, a, b) \in \mathsf{Rch}^\sigma_{m,q}$. Since this is true for all $a \in A$, it implies that Player A has no strategy to avoid the set $\mathsf{Rch}^\sigma_{m,q}$ in the game form $\mathcal{F}_q$, i.e. she has no winning strategy in the win/lose game $\langle \mathcal{F}_q, Q \setminus \mathsf{Rch}^\sigma_{m,q}\rangle$. In other words, $\mathcal{W}_\mathsf{A}(\mathcal{F}_q, Q \setminus \mathsf{Rch}^\sigma_{m,q}) = \emptyset$, which implies $\mathcal{W}_\mathsf{B}(\mathcal{F}_q, \mathsf{Rch}^\sigma_{m,q}) \neq \emptyset$ by determinacy of $\mathcal{F}_q$: Player B has a winning strategy in this game. ◀

The parallelization of the action map λ for Player B follows: any winning action in $\mathcal{W}_\mathsf{B}(\mathcal{F}_q, \mathsf{Rch}^\sigma_{m,q})$ will ensure that the strategy σ is correctly mimicked by $\mathsf{Par}(\sigma)$.

▶ **Example 15.** We illustrate the definition on the example on Figure 9. We keep the notations used before. We consider Player B strategy depicted in the left-hand side of Figure 9 from the state q and an arbitrary state of the memory m omitted on the figure. For each possible choice of Player A (which corresponds to the rows of the local interaction I_3), Player B reacts with his strategy and either x or y is reached (in blue on the left figure). Specifically, writing m' for $\mu(m, k_\mathcal{C})$, we have $\delta(q, a_1, \lambda(m', (q, a_1))) = x$, $\delta(q, a_2, \lambda(m', (q, a_2))) = x$ and $\delta(q, a_3, \lambda(m', (q, a_3))) = y$, where a_i represents the action for player A for the i-th row (similarly, b_i represents the action for Player B for the i-th column). Then, we must define the action for Player B to play in the concurrent game at state q, that is $\mathsf{Par}(\lambda)(m, q) \in B$, so that only the states x and y can be reached. To choose $\mathsf{Par}(\lambda)(m, q)$ we consider the local interaction $\mathcal{F}_q = I_3$. We know that for each action of Player A, there is one for Player B to reach the set $\{x, y\}$ (it is given by the strategy depicted in the turn-based arena). It follows that Player A has no winning strategy in the win/lose game I_3 with $\{x, y\}$ as winning set for Player B. Since the local interaction I_3 is determined, Player B has a winning strategy that ensures reaching a state in $\{x, y\}$. By opting for this strategy, which corresponds to choosing the second column in the local interaction, it follows that the states reachable in the concurrent arena from q are $\{x, y\}$ (depicted in blue). Hence, we set $\mathsf{Par}(\lambda)(m, q) = b_2$.

6 Applications

6.1 Games with deterministic Nature (i.e. without Nature)

We consider the special case of games $\mathcal{C}$ with a deterministic Nature (i.e. all probabilities are equal to 1: for all Nature states $d \in \mathsf{D}$, there is a state $q \in Q$ such that $\mathsf{dist}(d)(q) = 1$), as, on turn-based games, this setting enjoys more determinacy results than the stochastic one. In such a setting, it is relevant to consider infinite paths compatible with a chromatic strategy, not only their probabilities. A winning strategy is a strategy whose set of compatible paths is included in the winning set – U_W for Player A and $Q^\omega \setminus U_W$ for Player B.

A deterministic concurrent game $\langle \mathcal{C}, W \rangle$ is (resp. positionally, resp. finite-memory) *exactly-determined* if either of the player has a (resp. positional, resp. finite-memory) winning strategy. In the literature this notion is sometimes called "sure winning", while winning with probability 1 is called "almost-sure winning". However, in deterministic concurrent games with chromatic strategies (recall that they are deterministic strategies), we have an equivalence between the two notions. This immediately gives us:

▶ **Corollary 16.** *A deterministic CG-DLI is (resp. positionally, resp. finite-memory) exactly-determined if and only if it is (resp. positionally, resp. finite-memory) determined.*

In the following, the determinacy of a deterministic game will refer to exact-determinacy. We consider the transfer of determinacy results from turn-based games to CG-DLI.

Borel determinacy. We apply Theorem 12 and Corollary 16 to prove the Borel determinacy of CG-DLI. By rephrasing the famous result of Borel determinacy in our formalism, we have that a deterministic turn-based graph arena $\mathcal{C}$ is determined for all Borel winning set $W \subseteq \mathsf{Borel}(\mathsf{K})$. Note that this theorem is not directly given by the results proved by Martin in [14, 15, 16]. To obtain this theorem, we additionally need to apply a result from [20] since a strategy depends on color history instead of state history. We use this result to prove the determinacy of CG-DLI.

▶ **Theorem 17.** *For all Borel winning set W, for all locally determined deterministic concurrent graph arena $\mathcal{C}$, the concurrent game $\langle \mathcal{C}, W \rangle$ is determined. Conversely, for all non-trivial Borel winning set $\emptyset \subsetneq W \subsetneq \mathsf{K}^\omega$, for all non-determined game form $\mathcal{F}$, there exists a deterministic concurrent arena $\mathcal{C}$ with only $\mathcal{F}$ as a local interaction that is not determined such that the game $\langle \mathcal{C}, W \rangle$ is not determined.*

Finite-memory determinacy. The next application only applies to finite arenas. In [3], the authors proved an equivalence between the shape of a winning set and the existence of winning strategies that can be implemented with a given memory skeleton[2] $\mathcal{M}$. They defined the properties of $\mathcal{M}$-selectivity and $\mathcal{M}$-monotony and proved that for $\mathcal{M}$ a memory skeleton and $W \subseteq \mathsf{K}^\omega$, we have that W and $\mathsf{K}^\omega \setminus W$ are $\mathcal{M}$-monotone and $\mathcal{M}$-selective is equivalent to every finite deterministic turn-based game with W as winning set is determined with winning strategies for both players that can be found among strategies implemented with memory skeleton $\mathcal{M}$.

Let $\langle \mathcal{C}, W \rangle$ be a deterministic concurrent game, $\langle \mathsf{Seq}(\mathcal{C}), \mathsf{Seq}(W) \rangle$ be its sequential version, and $\mathcal{M}$ a memory skeleton on K. In fact, W is $\mathcal{M}$-monotone and $\mathcal{M}$-selective if and only if $\mathsf{Seq}(W)$ is $\mathsf{Seq}(\mathcal{M})$-monotone and $\mathsf{Seq}(\mathcal{M})$-selective. The proof of this fact, longer than the other applications, requires establishing algebraic properties of the projection function $\phi_{\mathsf{K}_\mathcal{C},\mathsf{K}} : \mathsf{K}_\mathcal{C}^\uparrow \to \mathsf{K}^\uparrow$. In turn, finite deterministic CG-DLI ensure the following theorem:

▶ **Theorem 18.** *Let $\mathcal{M}$ be a memory skeleton and $W \subseteq \mathsf{K}^\omega$. The following two assertions are equivalent:*

1. *every finite deterministic locally determined concurrent game $\langle \mathcal{C}, W \rangle$ with finite action sets is determined with winning strategies for both players that can be found among strategies implemented with memory skeleton $\mathcal{M}$;*
2. *W and $\mathsf{K}^\omega \setminus W$ are $\mathcal{M}$-monotone and $\mathcal{M}$-selective.*

As for Borel determinacy, local determinacy is somehow a necessary condition as a one-shot reachability game, with a non-determined initial local interaction, may not be determined (see Figures 5, 6). In fact, this theorem can written as a more involved equivalence.

6.2 Stochastic Games (i.e. with Nature)

There are fewer determinacy results on stochastic games, especially with deterministic strategies. Let us translate some of them into locally determined concurrent games. We consider parity objectives and the more general case of tail-objectives (a.k.a. prefix-independent).

Parity Objectives. As already mentioned in Section 3, parity objectives are defined as follows. For a set of colors $\mathsf{K} = [\![m, n]\!]$ for some $m, n \in \mathbb{N}$, a parity objective on K is the winning set $W = \{\rho \in \mathsf{K}^\omega \mid \max(\mathsf{n}_\infty(\rho)) \text{ is even } \}$ where $\mathsf{n}_\infty(\rho)$ is the set of colors seen infinitely often in ρ. A result from [4, 23] gives us that any finite turn-based parity game is positionally determinedThis result can be directly transferred to locally determined concurrent games thanks to Theorem 12. Note that, as in the two previous cases, the local determinacy assumption is somewhat necessary.

[2] In fact, they looked at the existence of Nash equilibria with antagonistic preference relations instead of winning sets. However, a winning set $W \subseteq \mathsf{K}^\omega$ can be directly translated into an equivalent preference relation $\prec_W \subseteq \mathsf{K}^\omega \times \mathsf{K}^\omega$ by $\rho \prec_W \rho' \Leftrightarrow \rho \notin W \wedge \rho' \in W$. In the following we will refer to the preference relation $\prec_W$ when mentionning the winning set W.

▶ **Theorem 19.** *Consider a (stochastic) locally determined finite concurrent graph arena $\mathcal{C}$ with color set* $\mathsf{K} = [\![m, n]\!]$ *for some* $m, n \in \mathbb{N}$*. For all parity objective* $W \in \mathsf{Borel}(\mathsf{K})$ *on* K*, the concurrent game* $\langle \mathcal{C}, W \rangle$ *is positionally determined.*

Tail Objectives. We consider more general objectives than the parity objectives. In particular, positional determinacy does not hold in the general case for these objectives (consider, for instance, the Muller objectives). A tail objective is a winning set that is closed by adding and removing finite prefixes, that is, for a set of colors K, a winning set $W \in \mathsf{Borel}(\mathsf{K})$ is a tail-objective if, for all $\rho \in \mathsf{K}^\omega$ and $\pi \in \mathsf{K}^*$, we have $\rho \in W \Leftrightarrow \pi \cdot \rho \in W$. In particular, a parity objective is a tail objective. In fact, we have that every finite turn-based game that is limit-determined with value 0 or 1 is determined.

This result can be directly transferred to locally determined concurrent games. As usual, the local determinacy is a somewhat necessary condition.

▶ **Theorem 20.** *Consider a (stochastic) locally determined finite concurrent graph arena $\mathcal{C}$ with finite action sets. Then, for all Borel winning set* $W \subseteq \mathsf{Borel}(\mathsf{K})$ *that is a tail objective, on* K*, if* $\chi_{\mathsf{A}}^{\mathcal{C}}[W] = 1$ *or* $\chi_{\mathsf{B}}^{\mathcal{C}}[W] = 0$*, then the game is determined.*

7 Future Work

Several applications from Section 6 can be generalized in the setting of non-antagonistic preferences instead of the win/lose setting that was used in this article. Apart from formatting bureaucracy, most of these generalizations are automatic corollaries of the combination of this paper's results and [18]. The latter is a general transfer result, from determinacy results in the win/lose setting, into existence of Nash equilibria in the setting of non-antagonistic preferences. We intend to state and detail these generalizations in the journal version of this paper.

References

1 Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux. From local to global determinacy in concurrent graph games. *CoRR*, abs/2107.04081, 2021. arXiv:2107.04081.

2 Endre Boros, Ondrej Cepek, and Vladimir Gurvich. Separable discrete functions: Recognition and sufficient conditions. *Discret. Math.*, 342(5):1275–1292, 2019. doi:10.1016/j.disc.2018.12.026.

3 Patricia Bouyer, Stéphane Le Roux, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhove. Games where you can play optimally with arena-independent finite memory. In *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, pages 24:1–24:22, 2020. doi:10.4230/LIPIcs.CONCUR.2020.24.

4 Krishnendu Chatterjee, Marcin Jurdzinski, and Thomas A. Henzinger. Quantitative stochastic parity games. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 121–130. SIAM, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982808.

5 Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 564–575. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743507.

6 Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational aspects of monotone dualization: A brief survey. *Discret. Appl. Math.*, 156(11):2035–2049, 2008. doi:10.1016/j.dam.2007.04.017.

7 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science FOCS, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377, 1991. doi:10.1109/SFCS.1991.185392.

8 Hugh Everett. Recursive games. *Annals of Mathematics Studies – Contributions to the Theory of Games*, 3:67–78, 1957.

9 Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms*, 21(3):618–628, 1996. doi:10.1006/jagm.1996.0062.

10 Allan Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41(4):587–601, 1973. URL: http://www.jstor.org/stable/1914083.

11 Hugo Gimbert and Florian Horn. Solving simple stochastic tail games. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 847–862. SIAM, 2010. doi:10.1137/1.9781611973075.69.

12 V.A. Gurvich. The solvability of positional games in pure strategies. *USSR Computational Mathematics and Mathematical Physics*, 15(2):74–87, 1975. doi:10.1016/0041-5553(75)90042-7.

13 Eryk Kopczyński. *Half-positional Determinacy of Infinite Games.* PhD thesis, Warsaw University, 2008.

14 Donald A Martin. Borel determinacy. *Annals of Mathematics*, pages 363–371, 1975.

15 Donald A Martin. A purely inductive proof of Borel determinacy. *Recursion theory (Ithaca, NY, 1982)*, 42:303–308, 1985.

16 Donald A Martin. The determinacy of Blackwell games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

17 Andrzej Włodzimierz Mostowski. *Games with forbidden positions.* Preprint – Uniwersytet Gdański. Instytut Matematyki. UG, 1991.

18 Stéphane Le Roux. From winning strategy to nash equilibrium. *Math. Log. Q.*, 60(4-5):354–371, 2014. doi:10.1002/malq.201300034.

19 Stéphane Le Roux. Concurrent games and semi-random determinacy. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, pages 40:1–40:15, 2018. doi:10.4230/LIPIcs.MFCS.2018.40.

20 Stéphane Le Roux. Time-aware uniformization of winning strategies. In *Beyond the Horizon of Computability - 16th Conference on Computability in Europe, CiE 2020, Fisciano, Italy, June 29 - July 3, 2020, Proceedings*, pages 193–204, 2020. doi:10.1007/978-3-030-51466-2_17.

21 Lloyd S. Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

22 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.

23 Wieslaw Zielonka. Perfect-information stochastic parity games. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2987 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 2004. doi:10.1007/978-3-540-24727-2_35.

Quantitative Verification on Product Graphs of Small Treewidth

Krishnendu Chatterjee
IST Austria, Klosterneuburg, Austria

Rasmus Ibsen-Jensen
University of Liverpool, UK

Andreas Pavlogiannis
Aarhus University, Denmark

Abstract

Product graphs arise naturally in formal verification and program analysis. For example, the analysis of two concurrent threads requires the product of two component control-flow graphs, and for language inclusion of deterministic automata the product of two automata is constructed. In many cases, the component graphs have constant treewidth, e.g., when the input contains control-flow graphs of programs. We consider the algorithmic analysis of products of two constant-treewidth graphs with respect to three classic specification languages, namely, (a) algebraic properties, (b) mean-payoff properties, and (c) initial credit for energy properties.

Our main contributions are as follows. Consider a graph G that is the product of two constant-treewidth graphs of size n each. First, given an idempotent semiring, we present an algorithm that computes the semiring transitive closure of G in time $\tilde{O}(n^4)$. Since the output has size $\Theta(n^4)$, our algorithm is *optimal* (up to polylog factors). Second, given a mean-payoff objective, we present an $O(n^3)$-time algorithm for deciding whether the value of a starting state is non-negative, improving the previously known $O(n^4)$ bound. Third, given an initial credit for energy objective, we present an $O(n^5)$-time algorithm for computing the minimum initial credit for all nodes of G, improving the previously known $O(n^8)$ bound. At the heart of our approach lies an algorithm for the efficient construction of strongly-balanced tree decompositions of constant-treewidth graphs. Given a constant-treewidth graph G' of n nodes and a positive integer λ, our algorithm constructs a binary tree decomposition of G' of width $O(\lambda)$ with the property that the size of each subtree decreases geometrically with rate $(1/2 + 2^{-\lambda})$.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Verification by model checking; Theory of computation → Graph algorithms analysis

Keywords and phrases graph algorithms, algebraic paths, mean-payoff, initial credit for energy

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.42

1 Introduction

Product graphs. Graphs are at the heart of formal analysis of reactive systems and programs. The nodes of the graph represent states of the system, edges represent transitions, and paths of the graph represent behaviors of the system. One graph problem that repeatedly arises in many applications is the analysis of product graphs (i.e., the synchronous product of two graphs). For example, in the analysis of two concurrent threads, the resulting graph for analysis is the product of two component control-flow graphs. Similarly, in language intersection or language inclusion between deterministic automata, the product of two automata is considered.

Specification languages. The analysis of programs and reactive systems is performed w.r.t. desired properties that are described as specification languages. We consider three classic specification languages: (i) algebraic properties w.r.t. a semiring, (ii) mean-payoff properties,

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 42; pp. 42:1–42:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and (iii) initial credit for energy properties. In algebraic properties for system analysis, (a) each transition is associated with a weight from a semiring; (b) the value of a path is the semiring product operation of the weights along the transitions of the path; and (c) path aggregation is performed using the semiring sum operation across the values of paths. In mean-payoff properties for system analysis, (a) each transition of the system is associated with an integer-valued weight; (b) the value of an infinite path is the long-run average of the weights of the transitions of the path; and (c) an optimal path is selected that has the minimum mean-payoff value among all paths. In initial credit for energy for system analysis, (a) each transition of the system is associated with an integer-valued weight; (b) the value of an infinite path is the smallest weight of all of its prefixes, and (c) for each node, an optimal path is selected that starts in that node and such that it has the largest value among all paths originating in that node.

Constant-treewidth graphs. One key structural property of graphs that appears in several contexts is that of *constant-treewidth.* Treewidth is a classic measure of closeness of a graph to a tree [52]. Besides its mathematical elegance, constant-treewidth graphs are of practical relevance in formal verification and program analysis. For example, (i) the control-flow graphs of typical programming languages (such as goto-free Algol, Pascal, and C programs) have constant treewidth [53], which has been exploited for fast static analyses [21, 15, 18], and in practice even control-flow graphs of Java programs have constant treewidth [37]; (ii) the analysis of constant-treewidth graphs in logic plays a crucial role, such as the celebrated result of Courcelle for MSO [26] and its subsequent extensions [2, 30, 8]; as well as for logics such as modal mu-calculus [48].

Significance of the problems. In this work we consider the algorithmic analysis of the product of two constant-treewidth graphs with respect to algebraic properties and mean-payoff properties. We discuss the significance of the problems we consider. First, as mentioned above, products of two constant-treewidth graphs arise naturally (a) in the analysis of concurrent programs, and (b) in model checking an implementation against a high-level specification, a task typically expressed as language inclusion. We now discuss the relevance of the specification languages we consider here.

- *Semiring properties.* Semiring (or algebraic) properties have been widely used as specification formalisms as weighted automata [28], or properties of programs [51, 1]. Semirings also form the basis of dataflow analysis of concurrent programs [36, 45, 32, 24, 41, 27, 19], where the underlying analysis is based on an algebraic “meet-over-all-paths” formulation. Finally, semirings also arise in concurrent Kleene algebras for the analysis of concurrent programs [38, 39, 42].
- *Mean-payoff properties.* Mean-payoff is a classic quantitative property in performance analysis [33, 49, 3]. It has applications in (a) automata theoretic formalisms [17, 16]; (b) weighted logic formalisms [9, 12, 29]; (c) synthesis of reactive systems [5, 14]; and (d) quantitative interprocedural analysis [22]; to name a few applications in verification and program analysis.
- *Initial credit for energy properties.* Initial credit is a useful quantitative property for expressing energy constraints [11, 10, 13]. The goal is to determine for each state of the system an initial energy supply so that the system can exhibit infinite behavior without running out of energy, and has numerous applications in planning [31, 40].

In many cases in verification, instead of having a graph with constant treewidth, the input is a graph G that is the product of two constant treewidth graphs G_1, G_2. For example, this holds when we analyze control-flow graphs of two threads running in parallel, or when we test for language inclusion and the system and specification automaton have constant-treewidth. Note that G does not have constant treewidth: a simple example would be grid graphs - they are trivial to get this way, but have tree-width equal to the minimum of height and width. Due to this fact, existing algorithms for constant-treewidth graphs give sub-optimal solutions. The question is thus whether better algorithms are possible given the fact that, although G does not have constant treewidth, its two components G_1 and G_2 do have constant treewidth. We address this challenge in this work.

Our contributions. Our main contributions are as follows.

1. *Mean-payoff properties.* Given a product G of two constant treewidth graphs G_1, G_2 of size n each and a mean-payoff objective, we present an $O(n^3)$-time algorithm for deciding whether the value of a starting state is non-negative. Note that constant-treewidth graphs have $O(n)$ edges. Existing algorithms (such as Karp's algorithm) solve this problem in $O(n \cdot m)$ time on a graph of n nodes and m edges, which results to $O(n^4)$ complexity on G (since G has n^2 nodes and edges). Hence our algorithm yields a factor-n improvement compared to existing approaches.
2. *Semiring properties.* Given an idempotent semiring and a product G of two constant-treewidth graphs G_1, G_2 of size n each, we present an algorithm that computes the semiring transitive closure of G in time $\tilde{O}(n^4)$. On the other hand, applying the classic cubic-time transitive closure algorithm on G yields an $O(n^6)$ bound. Although this naive bound has been improved to $O(n^{4+\epsilon})$, for fixed $\epsilon > 0$ [19], our work yields a further polynomial improvement to $\tilde{O}(n^4)$. Since G has n^2 nodes, the output has size $\Theta(n^4)$ and thus our algorithm is *optimal* (up to polylog factors).
3. *Initial credit for energy properties.* Given a product G of two constant treewidth graphs G_1, G_2 of size n each and an energy objective, we present an $O(n^5)$-time algorithm for computing the minimum initial credit for each node of G. The best existing solution is due to [20] which has quartic complexity in the size of the graph, and thus implies an $O(n^8)$ time bound on G (since G has n^2 nodes). Hence our algorithm yields a factor-n^3 improvement.
4. At the heart of our approach lies a new notion of (α, β) tree decompositions, as well as an efficient algorithm for their construction for constant-treewidth graphs. Given a constant-treewidth graph G of n nodes and an integer $\lambda \geq 2$, our algorithm constructs in $O(\lambda^2 \cdot n \cdot \log n)$ time a binary tree decomposition of G of width $O(\lambda)$ that has the following property: for each bag B of the tree decomposition at level i, the number of nodes of G contained in bags of the subtree rooted at B is at most $n \cdot (1/2 + 2^{-\lambda})^i$. Hence, for increasing values of λ, the number of contained nodes gets exponentially close to the optimal value of $n \cdot 2^{-i}$ (since the tree decomposition is binary). Note that for $\lambda = O(1)$, we obtain a tree decomposition with logarithmic depth and increased width by a constant factor. We complement this result with a lower bound stating that tree decompositions of logarithmic depth must, in general, incur a constant-factor increase in the width.

   Due to space restrictions, some proofs are relegated to the appendix.

Comparison to related existing work. The notion of balanced tree decompositions has long existed in the literature. The classic work of [50] presents the first algorithm to construct a balanced tree decomposition in time $O(n \cdot \log n)$, by finding balanced separators in the graph. Various works present parallel algorithms for constructing balanced tree decompositions in (poly-)logarithmic parallel time with $O(n)$ processors [46, 7]. The work of [30] constructs balanced tree decompositions in Logspace. More recently, the work of [35] constructs approximate balanced tree decompositions in time $O(f(t) \cdot n \cdot \log n)$), where $f(t)$ is a polynomial function of the treewidth t.

In all these cases, the balancing guarantee is that the tree decomposition has depth $\leq c \cdot \log n$, for some constant c (logarithms are on base 2). For binary tree decompositions, these algorithms yield $c \geq 2$. Crucially, the complexity of the algorithms for our results (1)–(3) depends on c, and $c \geq 2$ is prohibitively large. E.g., using the existing algorithms for balanced tree decompositions yields a bound for the semiring properties (result (2)) that is at least n^6, as opposed to our $\tilde{O}(n^4)$ bound.

Although [19] also considers a notion of strong balancing, the balancing factor achieved there for level i is, on average, $(1/2 + P(\lambda^{-1}))^i$, where P is a sub-linear function. Hence, compared to that algorithm, our new algorithm yields an exponential improvement in the balancing factor (i.e., $(1/2 + 2^{-\lambda})^i$). This improvement is necessary to arrive at our results. Moreover, our new techniques are quite different from previous ones, and might be of independent interest.

2 Preliminaries

In this section we set up our main notation and introduce our new notion of (α, β) tree decompositions. In the next sections we will show how to construct such decompositions efficiently, as well as how they can be used for developing algorithmic improvements on product graphs.

Graphs. We consider directed graphs $G = (V, E)$ where V is a set of n nodes and $E \subseteq V \times V$ is an edge relation. Two nodes $u, v \in V$ are called *neighbors* if $(u, v) \in E$. Given a set $X \subseteq V$, we denote by $G \restriction X$ the subgraph $(X, E \cap (X \times X))$ of G induced by the set of nodes X. A path $P : u \rightsquigarrow v$ is a sequence of nodes $(x_1, \dots, x_k)$ such that $u = x_1$, $v = x_k$, and for all $1 \leq i \leq k-1$ we have $(x_i, x_{i+1}) \in E$. The path P is *acyclic* if every node appears at most once in P. The length of P is $k-1$, and a single node is by itself a 0-length path. Given a path P we use the notation $u \in P$ to say that a node u appears in P, and $A \cap P$ to refer to the set of nodes that appear in both P and a set A.

Trees. A (rooted) tree $T = (I, F)$ is an undirected graph (the edge relation F is symmetric) without self loops and with a distinguished node r, which is the root of T, such that there is a unique acyclic path $P_u^v : u \rightsquigarrow v$ for each pair of nodes u, v. The *size* of T is $|I|$. Given a tree T with root r, the *level* $\mathsf{Lv}(u)$ of a node u is the length of the path P_u^r from u to the root r. Every node in P_u^r is an *ancestor* of u, and if v is an ancestor of u, then u is a *descendant* of v (u is both an ancestor and a descendant of itself). We call v a *strict ancestor* (resp., *strict descendant*) of u if $v \neq u$ and v is an ancestor (resp., descendant) of u. For a pair of nodes $u, v \in I$, the *lowest common ancestor (LCA)* of u and v is the common ancestor of u and v with the largest level. Given a node $v \neq r$, the *parent* u of v is the unique ancestor of v in level $\mathsf{Lv}(v) - 1$, and v is a *child* of u. We denote by $\mathsf{Parent}(v)$ the parent of node $v \neq r$. A *leaf* of T is a node with no children. For a node $u \in I$, we denote by $T(u)$ the subtree of T rooted in u (i.e., the tree consisting of all descendants of u). A node is k-ary if it has at most k children, and a tree is k-ary if every node is k-ary. The *depth* of T is $\max_u \mathsf{Lv}(u)$.

Connected components of trees. Let $T = (I, F)$ be a tree. A *connected component* $\mathcal{C} \subseteq I$ of T is such that for every pair of nodes $u, v \in \mathcal{C}$, the unique acyclic path P_u^v in T visits only nodes in $\mathcal{C}$. The *border* of a non-empty $\mathcal{C}$ is the set $\mathsf{Border}(\mathcal{C}) = \{u \in I \setminus \mathcal{C} : \exists v \in \mathcal{C} \text{ s.t. } (u, v) \in F\}$,

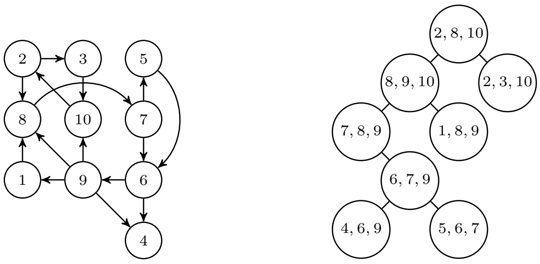


Figure 1 A graph with treewidth 2 (left) and a corresponding tree-decomposition (right).

i.e., it is the set of nodes of T that are adjacent to $\mathcal{C}$. For technical convenience, in this work we also make use of empty connected components of T. Empty connected components are also associated with a border, which is two endpoints of an edge of T. Hence we have $|F|$ many different empty connected components.

Balancing separators. Consider a connected component $\mathcal{C}$ of a tree. A *balancing separator* of $\mathcal{C}$ is a node $u \in \mathcal{C}$ such that removing u splits $\mathcal{C}$ into (at most) 3 connected components $\{\mathcal{C}_i\}_{1\leq i\leq 3}$, with $|\mathcal{C}_i| \leq |\mathcal{C}|/2$ for each $1 \leq i \leq 3$. The following lemma is well-known (e.g., [25]).

▶ **Lemma 1.** *Consider a binary tree T and a connected component $\mathcal{C}$ of T. A balancing separator of $\mathcal{C}$ can be computed in $O(|\mathcal{C}|)$ time.*

Tree decompositions. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{B}, T)$ where $T = (I, F)$ is a tree and $\mathcal{B} = \{B_i : \ i \in I\}$ is a family of subsets of V such that the following conditions hold.

1. $\bigcup_{i\in I} B_i = V$
2. For all $(u, v) \in E$ there exists $i \in I$ with $u, v \in B_i$.
3. For all $i, j, k \in I$, if $k \in P_i^j$ in T then $B_i \cap B_j \subseteq B_k$.

The sets B_i are called *bags* of the tree decomposition. Given a node $u \in V$, we denote by i_u the *root node* of u in T, which is the smallest-level node in T such that $u \in B_{i_u}$, and call B_{i_u} the *root bag* of u. Conditions Item 1 and Item 3 of tree decompositions guarantee that every node has a unique root bag. The *width* of the tree decomposition is $\max_{i\in I} |B_i| - 1$, i.e., it is the size of the largest bag of $\mathcal{B}$ minus 1. The *treewidth* of G is the smallest width of all tree decompositions of G. Given a node $i \in I$, we denote by $N^T_{\mathcal{B}}(i)$ the set of nodes of G that appear in bags of the subtree $T(i)$ (i.e., $u \in N^T_{\mathcal{B}}(i)$ iff i has a descendant j such that $u \in B_j$), and by $\mathcal{Y}^T_{\mathcal{B}}(i)$ the set of nodes of G that appear only in bags of the subtree $T(i)$ (i.e., $u \in \mathcal{Y}^T_{\mathcal{B}}(i)$ iff for the root bag B_j of u, j is a descendant of i). Observe that $N^T_{\mathcal{B}}(i) \subseteq \mathcal{Y}^T_{\mathcal{B}}(i) \cup B_i$. We assume w.l.o.g. that $\mathcal{Y}^T_{\mathcal{B}}(i) \neq \emptyset$ for every $i \in I$, as otherwise the subtree $T(i)$ can be removed from T and obtain a valid tree decomposition of G. For simplicity of exposition, we associate properties of the tree T with the tree decomposition $(\mathcal{B}, T)$, e.g., the depth of the tree decomposition is the depth of T, and we say that the tree decomposition is balanced if T is balanced. The following lemma states a well-known separator property of tree decompositions, which is a key property behind many efficient algorithms on low-treewidth graphs.

▶ **Lemma 2** (Lemma 3, [6]). *Consider a graph $G = (V, E)$, a tree-decomposition $(\mathcal{B}, T = (I, F))$ of G and a node $i \in I$. Let $\{\mathcal{C}_j\}_j$ be the connected components of T created by removing i from T. Consider two integers j_1, j_2 such that $j_1 \neq j_2$, and two nodes $i_1 \in \mathcal{C}_{j_1}$ and $i_2 \in \mathcal{C}_{j_2}$. For any two nodes $u \in B_{i_1}$ and $v \in B_{i_2}$, every path $P : u \rightsquigarrow v$ in G contains a node that appears in B_i.*

Approximate, balanced, and (α, β) tree decompositions. Consider a graph G of n nodes and treewidth t, and let $(\mathcal{B}, T = (I, F))$ be a tree decomposition of G. We refer to $(\mathcal{B}, T)$ as α-*approximate*, for some integer $\alpha \geq 1$, if the width of $(\mathcal{B}, T)$ is $\leq \alpha \cdot (t+1) - 1$. We refer to $(\mathcal{B}, T)$ as β-*balanced*, for some $0 < \beta < 1$, if for every node $i \in I$ we have that $\mathcal{Y}_{\mathcal{B}}^{T}(i) \leq n \cdot \beta^{\mathsf{Lv}(i)}$. If $(\mathcal{B}, T)$ is both α-approximate and β-balanced, it is called a (α, β) tree decomposition. Intuitively, a (α, β) tree decomposition approximates the treewidth of G to a factor α, and for every $i \in I$, the number of nodes of G contained in bags of the subtree $T(i)$ decreases geometrically with $\mathsf{Lv}(i)$ by a factor β. Note that if β is constant (i.e., independent of G) then T has depth $O(\log n)$ (hence it is balanced).

3 Construction of (α, β) tree decompositions

In this section we present our algorithm for constructing (α, β) tree decompositions, where α and β depend on some integer input $\lambda \geq 2$. In particular, we establish the following theorem.

▶ **Theorem 3.** *Consider a graph G of n nodes and treewidth t, and any integer $\lambda \geq 2$. Let $\mathcal{T}(G)$ be the time required to construct a tree decomposition of G with $\leq n$ bags and width t. A (α, β) tree decomposition of G can be constructed in $O(\mathcal{T}(G) + \lambda^2 \cdot n \cdot \log n)$ time, where $\alpha = 11 \cdot \lambda + 32$ and $\beta = 1/2 + 2^{-\lambda}$.*

Hence, for larger values of λ, the constructed tree decomposition is more strongly balanced, which also incurs a factor increase in its width.

The motivation behind Theorem 3 is as follows. The properties we consider later for product graphs (i.e., semiring, mean-payoff and initial credit for energy properties) are solved by existing algorithms that operate on a tree decomposition of the input graph. In high level, these algorithms iterate over every bag in the input tree decomposition and perform a polynomial-time computation in it. Because we deal with tree decompositions of product graphs, as we go down the tree decomposition, the number of bags in each level increases geometrically. By using Theorem 3, we ensure that the size of the bags reduces geometrically by an appropriate factor, which in turn ensures that the total time spent for all bags in each level of the tree decomposition stays bounded. The constant λ is chosen in each case to ensure this effect.

The high-level intuition behind Theorem 3 is as follows. We start with a tree decomposition of G, obtained using standard algorithms. The tree decomposition is then split recursively. Each recursive step operates on a part of the tree that consists of connected components, and splits this part into two sub-parts, in such a way that the overall number of nodes is balanced, meaning that the two parts are of approximately equal size. Since these parts shrink by a factor of $1/2$ in every step, after λ steps, the balance is only off by at most a factor of $2^{-\lambda}$. The key challenge is to perform the aforementioned splits in such a way that (i) the two constructed parts are approximately balanced, and (ii) the nodes appearing in each such part are separated from the rest of the nodes via a few "border" nodes.

We complement Theorem 3 by showing that, generally, balanced tree decompositions are approximate.

▶ **Theorem 4.** *For any n and $t = o(n/\log n)$, there exists a graph G_t^n that has treewidth at most $2 \cdot t - 1$, but any tree decomposition of G_t^n with depth $O(\log n)$ has width at least $3 \cdot t - 1$.*

In Section 3.1 we develop two operations on tree components that will be used later on. In Section 3.2 we develop our main algorithm which uses the operations of Section 3.1 to construct the (α, β) tree decomposition of Theorem 3. Finally, in Section 3.3 we prove the lower-bound of Theorem 4.

3.1 Operations on tree components

In this section we define set-components of trees and two operations on such set-components that will be used later for constructing (α, β) tree decompositions. In words, a set-component of a tree is simply a set of pairwise-disjoint connected components of the tree. The operations that we introduce here take as input a set-component. Each operation splits that set-component into multiple sub-components so that certain properties are met. In what follows, we fix a tree $T = (I, F)$.

Set-components of trees. A *set-component* of T is a set $\mathcal{X} = \{\mathcal{C}_i\}_i$ of connected components of T such that for each $i \neq j$ we have (i) $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ and (ii) $\mathcal{C}_i \cap \mathsf{Border}(\mathcal{C}_j) = \emptyset$. The *size* of $\mathcal{X}$ is $\mathsf{size}(\mathcal{X}) = \sum_i |\mathcal{C}_i|$, i.e., it is the total size of all of its connected components. The *border* of the set-component $\mathcal{X}$ is defined as $\mathsf{Border}(\mathcal{X}) = \bigcup_i \mathsf{Border}(\mathcal{C}_i)$, i.e., the border of $\mathcal{X}$ is the union of the borders of its connected components. Given two set-components $\mathcal{X}$ and $\mathcal{X}'$, we define $\mathcal{X} \sqsubseteq \mathcal{X}' = \bigcup_{\mathcal{C}_i \in \mathcal{X}} \mathcal{C}_i \subseteq \bigcup_{\mathcal{C}'_i \in \mathcal{X}'} \mathcal{C}'_i$ and $\mathcal{X} \sqcap \mathcal{X}' = (\bigcup_{\mathcal{C}_i \in \mathcal{X}} \mathcal{C}_i) \cap (\bigcup_{\mathcal{C}'_i \in \mathcal{X}'} \mathcal{C}'_i)$.

3.1.1 The operation BorderSplit

Intuitively, a component $\mathcal{X}$ is connected to the rest of the tree via the nodes of its border $\mathsf{Border}(\mathcal{X})$. Our balancing algorithm later needs that this border never gets too large. To achieve this, we define the operation $\mathsf{BorderSplit}$. In particular, consider a set-component $\mathcal{X} = \{\mathcal{C}_i\}_{1 \leq i \leq k}$ of T. We define the operation $\mathsf{BorderSplit}$ on $\mathcal{X}$ which returns (at most) three set-components$\{\mathcal{X}_i\}_{1 \leq i \leq 3}$ with $\mathcal{X}_i \sqsubseteq I$ for each $1 \leq i \leq 3$. In words, $\mathsf{BorderSplit}$ splits $\mathcal{X}$ into three set-components $\mathcal{X}_i$, possibly by removing a node $u \in \mathcal{X}$, such that $|\mathsf{Border}(\mathcal{X}_i)| \leq |\mathsf{Border}(\mathcal{X})|/2 + 1$ for each $1 \leq i \leq 3$. Formally, $\mathsf{BorderSplit}$ operates as follows (recall that the input tree T is binary). If $|\mathsf{Border}(\mathcal{X})| < 3$, then we simply return $\{\mathcal{X}\}$. Otherwise, we perform the following steps. First, we construct the LCA tree $T' = (I', F')$ of $\mathsf{Border}(\mathcal{X})$, defined as follows.

1. I' is the smallest subset of I such that (i) $\mathsf{Border}(\mathcal{X}) \subseteq I'$ and (ii) for every two nodes $u, v \in I'$, we have $w \in I'$, where w is the LCA of u and v. Hence, I' is the LCA-closure of $\mathsf{Border}(\mathcal{X})$.
2. For every pair of distinct nodes $u, v \in I'$, we have that $(u, v) \in F'$ iff v is the largest-level strict ancestor of u in T that appears in I'.

Then, we find a node u of T' such that the removal of u splits T' into three connected components $\{A_i\}_i$ with $|A_i \cap \mathsf{Border}(\mathcal{X})| \leq \lfloor |\mathsf{Border}(\mathcal{X})|/2 \rfloor$. This is easily done by a bottom-up pass of T'. For each $1 \leq i \leq 3$, we construct a set-component $\mathcal{X}_i$, as follows.

1. For every connected component $\mathcal{C}_j \in \mathcal{X}$ with $u \notin \mathcal{C}_j$, we make $\mathcal{C}_j \in \mathcal{X}_i$ where i is such that $\mathsf{Border}(\mathcal{C}_j) \subseteq A_i$.
2. If there exists a connected component $\mathcal{C}_j \in \mathcal{X}$ with $u \in \mathcal{C}_j$, we split $\mathcal{C}_j$ into three connected components $\{\mathcal{C}^i_j\}_i$ by removing u from $\mathcal{C}_j$, such that $\mathsf{Border}(\mathcal{C}^i_j) \setminus \{u\} \subseteq A_i$. In addition, we take $\mathcal{C}^i_j \in \mathcal{X}_i$.

Finally, we return the set $\{\mathcal{X}_i\}_i$. The following lemma states the properties of $\mathsf{BorderSplit}$.

▶ **Lemma 5.** *Consider the operation* $\mathsf{BorderSplit}$ *on the set-component* $\mathcal{X} = \{\mathcal{C}_i\}_{1 \leq i \leq k}$*. Let* $z = |\mathsf{size}(\mathcal{X})|$*,* $m = |\mathsf{Border}(\mathcal{X})|$*, and* $\{\mathcal{X}_i\}_{1 \leq i \leq 3}$ *be the returned component set. The following assertions hold.*

1. *We have* $|\bigcup_i \mathsf{Border}(\mathcal{X}_i)| \leq m + 1$*.*
2. *For each* $1 \leq i \leq 3$*, we have* $|\mathsf{Border}(\mathcal{X}_i)| \leq \lfloor m/2 \rfloor + 1$*.*
3. *After one* $O(n)$*-time preprocessing of* T*, every call to* $\mathsf{BorderSplit}$ *requires* $O(z + m^2)$ *time.*

3.1.2 The operation Split

Consider a set-component $\mathcal{X} = \{\mathcal{C}_i\}_{1 \le i \le k}$ of T and some $\lambda \ge 2$. Let $z = \mathsf{size}(\mathcal{X})$ and $m = |\mathsf{Border}(\mathcal{X})|$. We define the operation Split on $\mathcal{X}$ which returns two set-components $\{\mathcal{X}_i\}_{1 \le i \le 2}$ such that the following properties hold.

1. $\mathcal{X}_i \sqsubseteq \mathcal{X}$ and $\mathcal{X}_1 \sqcap \mathcal{X}_2 = \emptyset$.
2. $\mathsf{size}(\mathcal{X}_i) \le z \cdot (1/2 + 2^{-\lambda})$, i.e., the size of each set-component is approximately half the size of $\mathcal{X}$, and this approximation is controlled by λ, and
3. $|\mathsf{Border}(\mathcal{X}_i)| \le 9/10 \cdot m + \lambda + 3$, i.e., the size of the border of each set-component is about a fraction $9/10$ of the size of the border of $\mathcal{X}$.

Split initializes the two set-components as empty, i.e., $\mathcal{X}_1 = \mathcal{X}_2 = \emptyset$, and will be inserting connected components in each $\mathcal{X}_i$ such that, in the end, the stated properties hold. The algorithm operates in two steps, where the input to the second step is a set-component $\mathcal{X}^*$ constructed in the first step. In the following, we describe the two steps. We say that a set-component $\mathcal{X}' \sqsubseteq \mathcal{X}$ is *border-balancing* if either $m < 10$, or $\mathsf{size}(\mathcal{X}') \le z/2$ and $|\mathsf{Border}(\mathcal{X}')| \ge m/10 - 1$. Hence if $m < 10$ we take every sub-component of $\mathcal{X}$ to be border-balancing, otherwise a border-balancing sub-component must be at most of half the size of $\mathcal{X}$ and have about at least 1/10-th of the size of the border of $\mathcal{X}$. Given two set-components $\mathcal{X}'$ and $\mathcal{X}''$, we say that we *add* $\mathcal{X}'$ to $\mathcal{X}''$ meaning that we update $\mathcal{X}''$ to $\mathcal{X}' \cup \mathcal{X}''$.

Step 1: making $\mathcal{X}_1$, $\mathcal{X}_2$ border-balancing. If $m < 10$, then we proceed with the second step with $\mathcal{X}^* = \mathcal{X}$. Otherwise, we apply the operation BorderSplit on $\mathcal{X}$ and obtain the set-components $\{\mathcal{X}'_i\}_{1 \le i \le 3}$. We assume w.l.o.g. that $\mathsf{size}(\mathcal{X}'_1) \ge \mathsf{size}(\mathcal{X}'_2) \ge \mathsf{size}(\mathcal{X}'_3)$. If there are two border-balancing set-components, $\mathcal{X}_a$ and $\mathcal{X}_b$, among $\mathcal{X}'_1, \mathcal{X}'_2, \mathcal{X}'_3$, we add $\mathcal{X}_a$ to $\mathcal{X}_1$ and $\mathcal{X}_b$ to $\mathcal{X}_2$, and proceed to the second step with $\mathcal{X}^*$ being the unique set-component in $\{\mathcal{X}'_1, \mathcal{X}'_2, \mathcal{X}'_3\} \setminus \{\mathcal{X}_a, \mathcal{X}_b\}$. Otherwise, we apply the operation BorderSplit on $\mathcal{X}'_1$ and obtain the set-components $\{\mathcal{X}''_i\}_{1 \le i \le 3}$. We assume w.l.o.g. that $\mathsf{size}(\mathcal{X}''_1) \ge \mathsf{size}(\mathcal{X}''_2) \ge \mathsf{size}(\mathcal{X}''_3)$. In the set $A = \{\mathcal{X}''_1, \mathcal{X}''_2, \mathcal{X}''_3, \mathcal{X}'_2, \mathcal{X}'_3\}$ there are at least two border-balancing set-components, $\mathcal{X}_a$ and $\mathcal{X}_b$. We add $\mathcal{X}_a$ to $\mathcal{X}_1$ and $\mathcal{X}_b$ to $\mathcal{X}_2$. Finally, we let $\mathcal{X}^*$ be the set-component in $A \setminus \{\mathcal{X}_a, \mathcal{X}_b\}$ with the largest size. We add one of the set-components in $A \setminus \{\mathcal{X}^*, \mathcal{X}_a, \mathcal{X}_b\}$ to the smaller (in size) of $\mathcal{X}_1$ and $\mathcal{X}_2$, and repeat with the other set-component of $A \setminus \{\mathcal{X}^*, \mathcal{X}_a, \mathcal{X}_b\}$.

Step 2: balancing $\mathcal{X}_1$, $\mathcal{X}_2$ based on size. Let $\mathcal{C}$ be a largest connected component of $\mathcal{X}^*$. First, we add every connected component of $\mathcal{X}^*$ except $\mathcal{C}$ to the smaller (in size) of $\mathcal{X}_1$ and $\mathcal{X}_2$, in order (i.e., once we have added one connected component to $\mathcal{X}_1$ or $\mathcal{X}_2$, we take into account the new size of $\mathcal{X}_1$ and $\mathcal{X}_2$ for choosing where to add the next connected component). The remaining of the second step is recursive for λ levels. The j-th recursive call operates on a connected component $\mathcal{C}^j$, where $\mathcal{C}^0 = \mathcal{C}$. Given $\mathcal{C}^j$, we use Lemma 1 to identify a balancing separator $u \in \mathcal{C}^j$, such that the removal of u splits $\mathcal{C}^j$ into three connected components $\{\mathcal{C}^j_i\}_i$. We assume w.l.o.g. that $|\mathcal{C}^j_1| \ge |\mathcal{C}^j_2| \ge |\mathcal{C}^j_3|$. We add each of $\mathcal{C}^j_2$ and $\mathcal{C}^j_3$ to the smallest (in size) of $\mathcal{X}_1$ and $\mathcal{X}_2$, in order, and proceed to the next recursive call with $\mathcal{C}^{j+1}$ be $\mathcal{C}^j_1$. Finally, at the end of the recursion, we add $\mathcal{C}^\lambda$ (from the last recursive call) to the smallest of $\mathcal{X}_1$ or $\mathcal{X}_2$.

The following lemma states the properties of Split, and relies on Lemma 5.

▶ **Lemma 6.** *Consider the operation* Split *on a set-component $\mathcal{X} = \{\mathcal{C}_i\}_{1 \le i \le k}$ Let $z = \mathsf{size}(\mathcal{X})$ and $m = |\mathsf{Border}(\mathcal{X})|$, and $\{\mathcal{X}_i\}_{1 \le i \le 2}$ be the returned component set. The following assertions hold.*

1. *For each $1 \le i \le 2$, we have $\mathsf{size}(\mathcal{X}_i) \le z \cdot (1/2 + 2^{-\lambda})$.*
2. *For each $1 \le i \le 2$, we have $|\mathsf{Border}(\mathcal{X}_i)| \le 9/10 \cdot m + \lambda + 3$.*
3. *We have $|\mathsf{Border}(\mathcal{X}_1) \cup \mathsf{Border}(\mathcal{X}_2)| \le |\mathsf{Border}(\mathcal{X})| + \lambda + 2$*
4. *After $O(n)$-time preprocessing of T,* Split *requires $O(z + m^2)$ time.*

3.2 Construction of (α, β) Tree Decomposition

Here we prove the main result of this section, which concerns the construction of an (α, β) tree-decomposition of a graph G, where $\alpha = 11 \cdot \lambda + 32$ and $\beta = 1/2 + 2^{-\lambda}$, for any integer $\lambda \geq 2$. Our construction proceeds in two steps. In Section 3.2.1 we present an intermediate step that takes as input a tree T and constructs a tree decomposition of T that has width α and is β-approximate. In Section 3.2.2 we use this construction towards Theorem 3.

3.2.1 Construction of tree decompositions of trees

Here we present algorithm Balance, which takes as input a tree T and some integer $\lambda \geq 2$, and constructs a tree decomposition of T that has width α and is β-approximate.

Step 1: constructing a component tree. Consider a tree $T = (I, F)$. A *component tree* of T is a pair $(\mathcal{V}, \mathcal{R})$ where $\mathcal{R} = (\mathcal{J}, \mathcal{D})$ is a rooted tree and $\mathcal{V} = \{\mathcal{X}_i : i \in \mathcal{J}\}$ is a set of components of T. In the first step, Balance constructs a component tree $(\mathcal{V}, \mathcal{R})$ recursively, as follows.

1. The root of $\mathcal{R}$ is the component $\{I\}$.
2. Given a node i of $\mathcal{R}$, if $\mathsf{size}(\mathcal{X}_i) > 0$, we use the operation Split on the component $\mathcal{X}_i$ to obtain two components $\mathcal{X}_{j_1}, \mathcal{X}_{j_2}$. We insert these components in the set $\mathcal{V}$, make j_1, j_2 children of i in $\mathcal{R}$, and proceed recursively with each of j_1, j_2.

Step 2: turning the component tree to a tree decomposition. At the end of the first step, we have constructed a component tree $(\mathcal{V}, \mathcal{R})$ where every leaf of $\mathcal{R}$ is an empty set-component. In the second step, we construct a tree decomposition $(\mathcal{B}, \mathcal{T} = (\mathcal{I}, \mathcal{F}))$, such that $\mathcal{I} = \mathcal{J} \setminus L$, where L is the set of leaves of $\mathcal{R}$, and $\mathcal{T} = \mathcal{R} \restriction \mathcal{I}$. In words, $\mathcal{T}$ is identical to $\mathcal{R}$ without the leaves of the latter. For each $i \in \mathcal{I}$, we have $B_i = \bigcup_j \mathsf{Border}(\mathcal{X}_j)$, where j ranges over the children of i in $\mathcal{R}$. Note that i ranges over non-leaves of $\mathcal{R}$, and thus each B_i is well-defined. The following remark states that given a node i of the induced tree decomposition, the corresponding set-component $\mathcal{X}_i$ of the component tree contains exactly the nodes that appear in the bags of the subtree $\mathcal{T}(i)$.

▶ Remark 7. Let $(\mathcal{B}, \mathcal{T} = (\mathcal{I}, \mathcal{F}))$ be the tree decomposition of the component tree $(\mathcal{V}, \mathcal{R} = (\mathcal{J}, \mathcal{D}))$ constructed in the second step. For every $i \in \mathcal{I}$, we have $\mathcal{Y}_{\mathcal{B}}^{\mathcal{T}}(i) = \mathcal{X}_i$.

The following lemmas establish the correctness and complexity of Balance.

▶ **Lemma 8.** *$(\mathcal{B}, \mathcal{T} = (\mathcal{I}, \mathcal{F}))$ is a tree decomposition of T that has width $\leq \alpha$ and is β-balanced, for $\alpha = 11 \cdot \lambda + 32$ and $\beta = 1/2 + 2^{-\lambda}$.*

▶ **Lemma 9.** Balance *runs in $O(\lambda^2 \cdot |I| \cdot \log |I|)$ time.*

3.2.2 Construction of (α, β) tree decompositions

Finally, we present an algorithm for constructing (α, β) tree decompositions of arbitrary graphs. The input is a graph and a tree decomposition of the graph, and our algorithm constructs a new tree decomposition with the desired properties.

Construction of an (α, β) tree decomposition. Consider a graph $G = (V, E)$ with treewidth t, and some integer $\lambda \geq 2$. We construct a (α, β) tree decomposition of G, where $\alpha = 11 \cdot \lambda + 32$ and $\beta = 1/2 + 2^{-\lambda}$ in three steps.

1. Let $(\mathcal{B} = \{B\}_i, T = (I, F))$ be a tree decomposition of G with $|I| \leq n$ and width $\leq t$. We assume w.l.o.g. that, in $(\mathcal{B}, T)$, every node of G has a unique root bag, as we can always replace a bag which is the root of $k > 1$ nodes with a sequence of k bags, each being the root of a single node. In addition, we can remove any bag that is not the root bag of any node, and thus the size of the tree decomposition is exactly n.
2. We use Balance to construct a tree decomposition $(\mathcal{B}' = \{B'_i\}_i, T' = (I', F'))$ of T.
3. We construct the tree decomposition $(\overline{\mathcal{B}} = \{\overline{B}_i\}_i, T')$ with $\overline{B}_i = \bigcup_{j \in B'_i} B_j$ for each $i \in I'$.

We conclude Theorem 3 by arguing that the construction produces a (α, β) tree decomposition of G.

Proof of Theorem 3. Consider the tuple $(\mathcal{B}' = \{B'_i\}_i, T' = (I', F'))$ constructed in Step 2. By Lemma 8, $(\mathcal{B}', T')$ is a tree decomposition of T, and has width $\leq \alpha$ and is β-balanced. It follows that $(\overline{\mathcal{B}}, T')$ is a α-approximate tree decomposition of G, and it remains to argue that it is also β-balanced. For a node $u \in V$, let $A_u = \{i \in I : u \in B_i\}$ be the connected component of T in which u appears and we have $|A_u| \geq 1$. In particular, this holds for the (unique) root node i_u of u in T. Observe that for every node $j \in I'$, if $u \in \mathcal{Y}_{\overline{\mathcal{B}}}^{T'}(j)$, then $A_u \subseteq \mathcal{Y}_{\mathcal{B}'}^{T'}(j)$. Hence $|\mathcal{Y}_{\overline{\mathcal{B}}}^{T'}(j)| \leq |\mathcal{Y}_{\mathcal{B}'}^{T'}(j)|$. By Lemma 8, we have that $(\mathcal{B}', T')$ is β-balanced, and thus $|\mathcal{Y}_{\mathcal{B}'}^{T'}(j)| \leq |I| \cdot \beta^{\mathsf{Lv}(j)}$. Since $|I| = n$, we conclude that $|\mathcal{Y}_{\overline{\mathcal{B}}}^{T'}(j)| \leq n \cdot \beta^{\mathsf{Lv}(j)}$, as required.

We now turn our attention to the running time. The algorithm requires $\mathcal{T}(G)$ time in Step 1 for obtaining the initial tree decomposition $(\mathcal{B}, T)$ plus $O(n \cdot t)$ time for ensuring the properties of $(\mathcal{B}, T)$. By Lemma 9, the algorithm requires $O(\lambda^2 \cdot n \cdot \log n)$ in Step 2. Finally, the algorithm requires $O(\alpha \cdot t \cdot n) = O(\lambda \cdot t \cdot n)$ time in Step 3. ◀

3.3 A lower bound on the width of balanced tree decompositions

Here we present a family $\{G_t^n \mid n \geq 3 \cdot t \text{ and } n \equiv 0 \pmod{t}\}$ of graphs, where G_t^n has n nodes and treewidth $2 \cdot t - 1$, and any tree decomposition of G_t^n with width t' and depth h is such that either $h \geq n/(2 \cdot t')$ or $t' \geq 3 \cdot t - 1$. For $t = o(n/\log n)$, only in the latter case can the tree decomposition have depth $O(\log n)$ (i.e., be balanced), and hence the width must increase by a constant factor.

The graph G_t^n. The graph G_t^n is defined as follows. Let $n' = n/t$. For each $i \in \{1, \dots, n'\}$, let V_i be a set of t nodes, such that $V_i \cap V_j = \emptyset$ for $i \neq j$ and $V = \bigcup_i V_i$. Also, let $V_0 = V_{n'+1} = \emptyset$. For each $i \in \{1, \dots, n'\}$, each node in V_i has an edge to each other node in $V_{i-1} \cup V_i \cup V_{i+1}$ (see Figure 2). We start with a technical lemma that will help us later. Recall that $N_{\mathcal{B}}^T(i)$ denotes the set of nodes contained in bags of the subtree $T(i)$.

▶ **Lemma 10.** *For any t and n consider the graph G_t^n and a tree decomposition $(\mathcal{B}, T = (I, F))$ of G_t^n with width t' and depth h. Either $h \geq n/(2 \cdot t')$ or there exists integers i, i_1, i_2, such that $i_1 - i_2 \geq 2$ and $V_{i_1} \cup V_{i_2} \subseteq B_i$.*

Proof. Without loss of generality, we assume that for every $i \in I$, we have $N_{\mathcal{B}}^T(j) \setminus B_i \neq \emptyset$, where j ranges over the children of i in T. This is valid since, otherwise, we can simply remove the subtree $T(j)$ and still have a tree decomposition of G_t^n with the same or lower depth and width. Similarly, for every $i \in I$ that is not the root of T, we assume that $V \setminus N_{\mathcal{B}}^T(i) \neq \emptyset$, otherwise $T(i)$ is a tree decomposition of G_t^n with the same or lower depth and width.

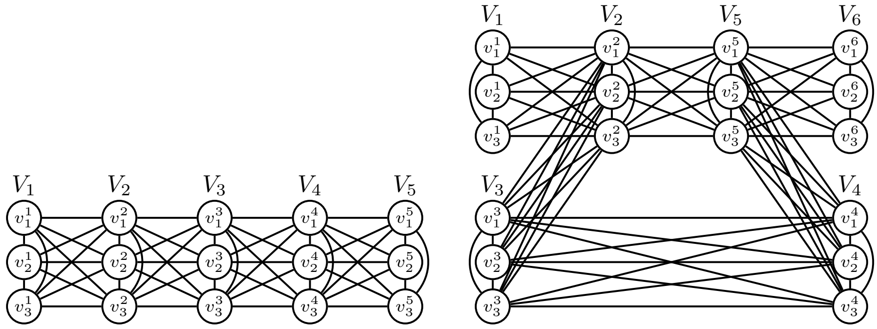


Figure 2 The graphs G_3^{15} (left) and $G(2,5,3,18)$ (right).

We distinguish the following cases. Either there exists an $i \in I$ with children $j_1, \dots, j_k$ where $k \geq 2$ (or $k \geq 3$, if i is the root of T) or not. If not, T forms a line with length at least n/t', hence the depth of T is at least $n/(2 \cdot t')$ (regardless of how T is rooted).

Otherwise, pick three nodes v_1, v_2, v_3 such that $v_1 \in N_{\mathcal{B}}^T(j_1) \setminus B_i$, $v_2 \in N_{\mathcal{B}}^T(j_2) \setminus B_i$ and $v_3 \in V \setminus N_{\mathcal{B}}^T(i)$ (or $v_3 \in N_{\mathcal{B}}^T(j_3) \setminus B_i$ in case i is the root of T). Let i', j', k' be such that $v_1 \in V_{i'}$, $v_2 \in V_{j'}$ and $v_3 \in V_{k'}$. We assume w.l.o.g. that $i' \leq j' \leq k'$. We have that $j' - i' \geq 2$ (resp. $k' - j' \geq 2$), since otherwise there is an edge between v_1 and v_2 (resp. v_2 and v_3) and hence a path between them that does not intersect with nodes in B_i, contradicting Lemma 2. Also, there exist integers i_1 and i_2 such that $i' < i_2 < j' < i_1 < k'$ (hence, $i_1 - i_2 \geq 2$) and such that $V_{i_2} \cup V_{i_1} \subseteq B_i$, since otherwise, there is at least one node in V_{i_2}, for $i' < i_2 < j'$ (resp. in $j' < i_1 < k'$) which is not in B_i and thus there is a path $P : v_1 \rightsquigarrow v_2$ (resp. $P : v_2 \rightsquigarrow v_3$) that does not contain nodes in B_i, again contradicting Lemma 2.

The desired result follows. ◀

We now show that the treewidth of G_t^n is at most $2 \cdot t - 1$.

▶ **Lemma 11.** *For any t and n, the treewidth of G_t^n is $\leq 2 \cdot t - 1$.*

Proof. Construct a tree $T = (I, F)$, where $I = \{1, \dots, n' - 1\}$ and $F = \{(i, i+1)\}_{i \in I \setminus \{n'-1\}}$. Construct the set of bags $\mathcal{B} = \{B_i = V_i \cup V_{i+1}\}_{i \in I}$. It is easy to see that $(\mathcal{B}, T)$ is a tree decomposition of G_t^n. ◀

The graph $G(i, j, t, n)$. Given numbers i and j and the graph G_t^n, for some t and n, let $G(i, j, t, n)$ be the graph similar to G_t^n, except that it also has an edge between each pair of nodes in $V_i \times V_j$ (see Figure 2). Next, we show that the treewidth of $G(i, j, t, n)$ is a factor larger than the one of G_t^n.

▶ **Lemma 12.** *For any i, j, t, n, where $j - i \geq 2$, the treewidth of $G(i, j, t, n)$ is $\geq 3 \cdot t - 1$.*

Proof. We construct a minor $G'(i, j, t, n)$ of $G(i, j, t, n)$ by contracting every edge (v_k^i, v_k^j) for $1 \leq k \leq t$. Observe that the clique $K_{3 \cdot t}$ is a minor of $G'(i, j, t, n)$. Since $K_{3 \cdot t}$ has treewidth $3 \cdot t - 1$ and treewidth is monotonic under graph minors [6, Lemma 16], it follows that the treewidth of $G(i, j, t, n)$ is a least $3 \cdot t - 1$. ◀

We next show that any tree decomposition of G_t^n has either large depth or large width.

▶ **Lemma 13.** *For any n and t, consider a tree decomposition $(\mathcal{B}, T = (I, F))$ of G_t^n of width t' and depth h. Either $h \geq n/(2 \cdot t')$ or $t' \geq 3 \cdot t - 1$.*

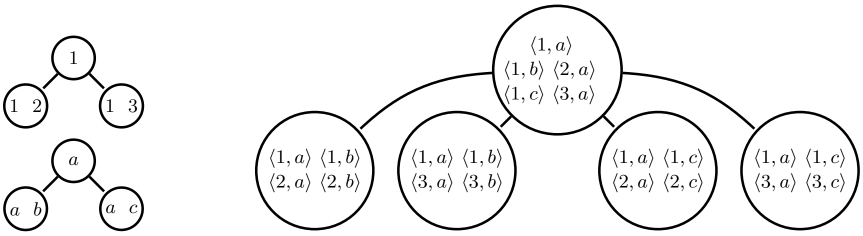

1
1 2
1 3
a
a b
a c
⟨1, a⟩ ⟨1, b⟩ ⟨2, a⟩ ⟨1, c⟩ ⟨3, a⟩
⟨1, a⟩ ⟨1, b⟩ ⟨2, a⟩ ⟨2, b⟩
⟨1, a⟩ ⟨1, b⟩ ⟨3, a⟩ ⟨3, b⟩
⟨1, a⟩ ⟨1, c⟩ ⟨2, a⟩ ⟨2, c⟩
⟨1, a⟩ ⟨1, c⟩ ⟨3, a⟩ ⟨3, c⟩

Figure 3 The product tree-decomposition given tree decompositions of the two constituent graphs.

Proof. If $h \geq n/(2 \cdot t')$, we are done. Otherwise, by Lemma 10, there exist integers i, i_1, i_2 such that $i_1 - i_2 \geq 2$ and $V_{i_1} \cup V_{i_2} \subseteq B_i$. Then, $(\mathcal{B}, T)$ is also a tree decomposition of $G(i_2, i_1, t, n)$, since each edge between V_{i_1} and V_{i_2} has both endpoints in B_i. By Lemma 12, the width of $(\mathcal{B}, T)$ is at least $3 \cdot t - 1$. ◀

We conclude with the proof of Theorem 4.

Proof of Theorem 4. The bound on t implies that any tree decomposition of width $t' < 3{\cdot}t{-}1$ and depth h, such that $h \geq n/(2 \cdot t')$ cannot be balanced. The statement then follows directly from Lemma 11 and Lemma 13. ◀

4 Applications to verification on product graphs

Here we discuss three applications of (α, β) tree decompositions in the analysis of product graphs of two constant-treewidth graphs, when the specification language is either w.r.t. semiring, mean-payoff or initial credit for energy properties. Product graphs arise frequently in verification, for example, when analyzing the behavior of two concurrent threads, or when the verification task is expressed as language inclusion wrt two automata. In such cases, the control-flow graphs or the automata are given explicitly, and the analysis proceeds by constructing and reasoning on the product graph.

We first establish a lemma which concerns the construction of a tree decomposition of the product graph with some desired properties. Afterwards we present algorithmic improvements for the analysis of such product graphs, w.r.t. mean-payoff, semiring and energy properties.

Product graphs. Given two graphs $G_i = (V_i, E_i)$, for $1 \leq i \leq 2$, the product graph of G_1 and G_2 is defined as the graph $G = (V, E)$ where $V = V_1 \times V_2$ and E is such that $(\langle u_1, u_2 \rangle, \langle v_1, v_2 \rangle) \in E$ iff $(u_i, v_i) \in E_i$ for each $1 \leq i \leq 2$. Let $(\mathcal{B}_i = \{B_i^j\}_j, T_i)$ be a tree decomposition of the graph G_i, for each $1 \leq i \leq 2$. We construct a tree decomposition $(\mathcal{B}, T)$ of G as follows (see Figure 3).

1. Let j_i be the root of T_i. Recall that $N^{T_i}_{\mathcal{B}_i}(j_i)$ is the set of nodes of G_i appearing in the bags of $T_i(j_i)$. We create a node j in T, and construct the bag $B_j = \left(B_1^{j_1} \times N^{T_2}_{\mathcal{B}_2}(j_2)\right) \cup \left(N^{T_1}_{\mathcal{B}_1}(j_1) \times B_2^{j_2}\right)$.
2. If for some $1 \leq i \leq 2$, the node j_i does not have a child (i.e., j_i is also a leaf in T_i), the process terminates. Otherwise, for every $1 \leq i \leq 2$ and child j_i' of j_i, we repeat recursively for the trees $T_i(j_i')$, and make every node j' constructed in the next recursive step a child of j in T.

It is not hard to verify that $(\mathcal{B}, T)$ is a tree decomposition of G, and T is a quaternary tree. By letting each $(\mathcal{B}_i, T_i)$ be a (α, β) tree decomposition for the graph G_i, we obtain the following lemma.

▶ **Lemma 14.** *Let $G = (V, E)$ be a product graph of two constant-treewidth graphs $G_i = (V_i, E_i)$, for $1 \leq i \leq 2$, with n nodes each, and consider any integer $\lambda \geq 2$. A quaternary tree decomposition $(\mathcal{B} = \{B_j\}_j, T = (I, F))$ of G can be constructed in $O((\lambda \cdot n)^2)$ time, such that for every $j \in I$ we have $|B_j| = O(\lambda \cdot n \cdot \beta^{\mathsf{Lv}(j)})$, where $\mathsf{Lv}(j)$ is the level of node j in T.*

Proof. For each $1 \leq i \leq 2$, we use Theorem 3 to obtain a (α, β) tree decomposition $(\mathcal{B}_i = \{B_i^j\}_j, T_i)$ for the graph G_i, where $\alpha = 11 \cdot \lambda + 32$ and $\beta = 1/2 \cdot (1 + 2^{-\lambda})$. Given each $(\mathcal{B}_i, T_i)$, the algorithm constructs the quaternary tree decomposition $(\mathcal{B}, T)$ of G. The bound $|B_j| = O(\lambda \cdot n \cdot \beta^{\mathsf{Lv}(j)})$ follows easily from Theorem 3 and the fact that each graph has treewidth t.

We now turn our attention to the time complexity, and show that the construction of $(\mathcal{B}, T)$ requires $O((\lambda \cdot n)^2)$ time. It is easy to see that the construction requires time proportional to the total sizes of all bags in $\mathcal{B}$, hence it suffices to argue that $\sum_j |B_j| = O((\lambda \cdot n)^2)$. We have

$$\sum_j |B_j| \leq 2 \cdot \sum_{j_1} \sum_{j_2} \left| B_1^{j_1} \right| \cdot \left| B_2^{j_2} \right| \leq \sum_{j_1} \Big(|B_1^{j_1}| \cdot \sum_{j_2} |B_2^{j_2}| \Big) \tag{1}$$

$$\leq \sum_{j_1} \Big(\alpha \cdot (t+1) \cdot \sum_{j_2} \alpha \cdot (t+1) \Big) = O((\lambda \cdot n)^2) \tag{2}$$

since $t = O(1)$. ◀

Note that the root bag of the product tree decomposition has $\Theta(n)$ nodes, and thus the width of the constructed tree decomposition is $\Theta(n)$ (as bags below the root decrease in size). We remark that this is unavoidable in general – e.g., two straight-line graphs G_1 and G_2 of n nodes each yield a product graph G that is a grid and thus has treewidth $n - 1$. The crucial property of our product tree decomposition is that, although it has width $\Theta(n)$, most bags have smaller size (as stated in Lemma 14).

4.1 Mean-payoff properties

In the minimum mean payoff problem, we are given a weighted graph $G = (V, E)$, a weight function $\mathsf{wt} \colon E \to \mathbb{Z}$, and a starting node $u \in V$. The decision problem is to compute whether

$$\inf_{\substack{u = x_1, x_2, \dots \\ (x_i, x_{i+1}) \in E}} \lim_{k \to \infty} \inf \frac{\sum_{i=1}^{k-1} \mathsf{wt}(x_i, x_{i+1})}{k} \geq 0$$

In words, we are interested in deciding whether the smallest weight-average among all infinite paths that originate in u is non-negative. Here we focus on the case where G is the product of two constant treewidth graphs $G_i = (V_i, E_i)$, for $1 \leq i \leq 2$, with n nodes each.

Solution on tree decompositions. The problem reduces to determining whether G contains a negative cycle w.r.t. wt that is reachable from u. We outline an existing algorithm for detecting negative cycles on G given a tree decomposition $(\mathcal{B} = \{B_i\}_i, T = (I, F))$ of G. We refer to [23] for details. We use a single data structure, which is a map $\mathsf{D} : \bigcup_i (B_i \times B_i) \to \mathbb{Q} \cup \{\infty\}$. Initially, $\mathsf{D}(u, v) = \mathsf{wt}(u, v)$ for each $(u, v) \in E$, and $\mathsf{D}(u, v) = \infty$ if $(u, v) \notin E$. Given a set of nodes $X \subseteq V$, we denote by D_X the restriction of D to the set X. We traverse T bottom-up and for each encountered node i, we compute all-pairs distances in the weighted graph $G^i = (B_i, B_i \times B_i)$ w.r.t. the weight function D_{B_i}, using the Floyd-Warshall algorithm. If a negative cycle is detected, we report that G has a negative cycle w.r.t. wt and stop. Otherwise, for every pair of nodes $u, v \in B_i$, we update the entry $\mathsf{D}(u, v)$ with the distance $d(u, v)$ in G_i, and proceed with the next node of T.

Algorithm for product graphs. Now consider that $G = (V, E)$ is the product of two constant-treewidth graphs $G_i = (V_i, E_i)$ with n nodes each. We choose $\lambda = 3$ and use Lemma 14 to construct a tree decomposition $(\mathcal{B}, T)$ of G for $\alpha = 11 \cdot \lambda + 32 = O(1)$ and $\beta = 1/2 + 2^{-\lambda} = 5/8$. Afterwards, we use the solution on tree decompositions for $(\mathcal{B}, T)$. We have the following theorem.

▶ **Theorem 15.** *Let $G = (V, E)$ be the product graph of two constant treewidth graphs $G_i = (V_i, E_i)$, for $1 \leq i \leq 2$, with n nodes each. Let $\mathsf{wt}\colon E \to \mathbb{Z}$ be a weight function on G. The decision problem of minimum mean payoff on (G, wt) can be solved in $O(n^3)$ time.*

Proof. The correctness follows directly from [23], and here we focus on the complexity. Since the tree decomposition of the product graph is quaternary, the i-th level has 4^i nodes. Since Floyd-Warshall has cubic complexity, the complexity of the algorithm for the whole level i of the tree decomposition is bounded, up to constant factors, by the following expression.

$$4^i \cdot \left(\beta^i \cdot n\right)^3 = n^3 \cdot \left(4 \cdot \left(\frac{5}{8}\right)^3\right)^i = n^3 \cdot A^i \tag{3}$$

where $A = 5^3/2^7 < 1$. It follows that the cost decreases geometrically along the levels of T and thus the running time is dominated by the first level, which runs in time $O(n^3)$. The desired result follows. ◀

The best current complexity bound for the problem is $O(n^4)$, achieved by Karp's classic algorithm [43] (recall that G has n^2 nodes). Hence Theorem 15 yields an improvement by a factor n, asymptotically.

4.2 Semiring properties

The problem of all-pairs semiring distances is defined w.r.t. a graph $G = (V, E)$ and a weight function $\mathsf{wt} : E \to \Sigma$, where Σ is the domain of an algebraic structure $(\Sigma, \oplus, \otimes, \overline{0}, \overline{1})$ with two associative operators $\oplus$ and $\otimes$ with neutral elements $\overline{0}$ and $\overline{1}$, respectively, such that (i) $\otimes$ distributes over $\oplus$, (ii) $\oplus$ is idempotent and (iii) $\overline{0}$ absorbs in $\otimes$. The task is to determine for very pair of nodes $u, v \in V$ the semiring distance from u to v, defined as

$$d(u, v) = \bigoplus_{P=(u_1,\dots,u_k):u\rightsquigarrow v} \bigotimes_{1\leq i<k} \mathsf{wt}(u_i, u_{i+1})$$

Here we focus on the case where G is the product of two constant treewidth graphs $G_i = (V_i, E_i)$, for $1 \leq i \leq 2$, with n nodes each. There are various classic algorithms for the problem, e.g. Lehmann's [47], Floyd's [34], Warshall's [54] and Kleene's [44], all of which have cubic complexity, assuming constant-time semiring operations. Since G has n^2 nodes, these algorithms take $O(n^6)$ time on G. Here we use strongly-balanced tree decompositions to obtain a solution in $\tilde{O}(n^4)$ time.

Solution on tree decompositions. We now outline an existing algorithm for computing the semiring transitive closure of G, given a tree decomposition $(\mathcal{B}=\{B_i\}_i, T=(I,F))$ of G. The algorithm is similar in spirit to the one in [23] for solving all-pairs distances. We use a single data structure, which is a map $\mathsf{D}: V\times V\to\Sigma$. Given a set of nodes $X\subseteq V$, we denote by D_X the projection of D to the set X. Initially, $\mathsf{D}(u,v)=\mathsf{wt}(u,v)$ for each $(u,v)\in E$, and $\mathsf{D}(u,v)=\overline{0}$ if $(u,v)\not\in E$. The algorithm consists of two parts.

1. We traverse T twice, first bottom-up and then top-down. For each encountered node i, we use the Floyd-Warshall algorithm to solve the algebraic path problem on the graph $G^i=(B_i, B_i\times B_i)$ w.r.t. the weight function D_{B_i}. For every pair of nodes $u,v\in B_i$, we update the entry $\mathsf{D}(u,v)$ with the semiring distance $d(u,v)$ in G^i.
2. For every node u, we perform a DFS in T starting from i_u. Given a current node i, for every node $v\in B_i$, we set $\mathsf{D}(u,v)=\bigoplus_{x\in B_i}(\mathsf{D}(u,x)\otimes\mathsf{D}(x,v))$. Finally, we return the map D, which contains the all-pairs semiring distances in G, and thus the solution to the algebraic path problem.

Algorithm for product graphs. Now consider that $G=(V,E)$ is the product of two constant-treewidth graphs $G_i=(V_i,E_i)$ with n nodes each. We choose $\lambda=c+\log\log n+1$, for some suitable constant c, and use Lemma 14 to construct a tree decomposition $(\mathcal{B},T)$ of G for $\alpha=11\cdot\lambda+32=O(\log\log n)$ and $\beta=1/2+2^{-\lambda}$. Afterwards, we use the solution on tree decompositions for $(\mathcal{B},T)$. We have the following theorem.

▶ **Theorem 16.** *Let $G=(V,E)$ be the product graph of two constant treewidth graphs $G_i=(V_i,E_i)$, for $1\leq i\leq 2$, with n nodes each. Let $\mathsf{wt}: E\to\Sigma$ be a weight function, where Σ is the domain of an idempotent semiring $(\Sigma,\oplus,\otimes,\overline{0},\overline{1})$. The semiring transitive closure of (G,wt) can be solved in $\tilde{O}(n^4)$ time.*

Proof. The correctness follows directly from [23], and here we focus on the complexity. Let $m=n\cdot\log\log n$. In Step 1, the algorithm spends cubic time in each bag, and by an argument similar to the proof of Theorem 15, we have that the running time of the first step is $O(m^3)=\tilde{O}(n^3)$.

We now proceed with the analysis of Step 2, which dominates the complexity. Observe that for each node u, the algorithm spends quadratic time in each bag of the product tree decomposition. Since the tree decomposition of the product graph is quaternary, the i-th level has 4^i nodes. Hence, the complexity of this step for the whole level i is bounded, up to constant factors, by the following expression

$$4^i\cdot(\beta^i\cdot m)^2=m^2\cdot\left(4\cdot\left(\frac{1+2^{-\lambda+1}}{2}\right)^2\right)^i=m^2\cdot\left(1+\frac{1}{c'\cdot\log n}\right)^i \tag{4}$$

where $c'=2^c$. Observe that the complexity increases with the level i, and there are at most $c''\cdot\log n$ levels, for some constant c''. We let $c'=c''$ and thus $c=\log(c'')$, and hence the complexity in the last level is bounded by

$$m^2\cdot\left(1+\frac{1}{c''\cdot\log n}\right)^{c''\cdot\log n}=O(m^2) \tag{5}$$

since $(1+1/x)^x\leq e$ for $x>0$. Summing up over all $O(\log n)$ levels, Step 2 of the algorithm spends $O(m^2\cdot\log n)$ time per node u and thus $O(n^2\cdot m^2\cdot\log n)=\tilde{O}(n^4)$ time for all nodes. ◀

Note that since the output has size $\Theta(n^4)$, the algorithm is optimal (up to polylog factors).

4.3 Initial credit for energy properties

In the minimum initial credit for energy problem, we are given a weighted graph $G = (V, E)$ and a weight function $\mathsf{wt}\colon E \to \mathbb{Z}$. The task is to compute for every node $u \in V$ the smallest energy value $\mathsf{E}(u) \in \mathbb{N} \cup \{\infty\}$ with the following property: there exists an infinite path $\mathcal{P} = (u_1, u_2, \dots)$ with $u_1 = u$ such that for every i we have $\mathsf{E}(u) + \sum_{j<i} \mathsf{wt}(u_j, u_{j+1}) \geq 0$. Conceptually, $\mathsf{E}(u)$ is the smallest "charge" we need to supply the system, so that starting from u it can exhibit infinite behavior without running out of energy. Conventionally, we let $\mathsf{E}(u) = \infty$ if no finite value exists.

Solution on tree decompositions. We outline an existing algorithm for the problem on G, given a tree decomposition $(\mathcal{B} = \{B_i\}_i, T = (I, F))$ of G. We first sketch the solution for arbitrary graphs G, and then explain how the solution is adapted to constant-treewidth graphs. We refer to [20] for details. The problem reduces to detecting non-positive cycles on weighted graphs of the form $(G^i = (V^i, E^i), \mathsf{wt}^i)_i$, where initially

1. $V^1 = V \cup \{s\}$, for some fresh node $s \notin V$ (intuitively, s acts as a sink in which all nodes x with $\mathsf{E}(x) = 0$ are collapsed),
2. $E^1 = E \cup (\{s\} \times V)$, and
3. $\mathsf{wt}^1(u, v) = -\mathsf{wt}(u, v)$ if $u \neq s$ and $\mathsf{wt}^1(u, v) = 0$ otherwise.

Given some $i \geq 1$, we detect a non-positive cycle on (G^i, wt^i), which determines a node x in that cycle for which $\mathsf{E}(x) = 0$. Then, we construct the weighted graph $(G^{i+1}, \mathsf{wt}^{i+1})$ where

1. $V^{i+1} = V^i \setminus \{x\}$,
2. $E^{i+1} = (E^i \setminus (V^i \times \{x\})) \cup \{(u, s) : (u, x) \in E^i\}$, and
3. $\mathsf{wt}^{i+1}(u, v) = \mathsf{wt}^i(u, v)$ if $v \neq s$ else $\mathsf{wt}^{i+1}(u, v) = \min(z, \mathsf{wt}^i(u, x))$, where $z = \mathsf{wt}^i(u, s)$ if $(u, s) \in E^i$ and $z = \infty$ otherwise.

Finally, if (G^i, wt^i) has no non-positive cycle, we compute the distance $d(u, s)$ from every $u \in V^i \setminus \{s\}$ to s in (G^i, wt^i) and assign $\mathsf{E}(u) = d(u, s)$ (at this point $d(u, s) > 0$ for each u).

We now consider the tree decomposition $(\mathcal{B} = \{B_i\}_i, T = (I, F))$ of G. The above solution is adapted as follows. First we construct the family of bags $\mathcal{B}' = \{\{s\} \cup B_i\}_i$, i.e., we insert the fresh node s in all bags. Observe that $(\mathcal{B}', T)$ is a valid tree decomposition of all G^i. Then, every step of non-positive-cycle detection, as well as the last step of computing the distances $d(u, s)$ is performed by a single bottom-up pass of T. In every encountered node i, an all-pairs distance computation is performed in the graph induced by B'_i, using the Floyd-Warshall algorithm.

Algorithm for product graphs. Now consider that $G = (V, E)$ is the product of two constant-treewidth graphs $G_i = (V_i, E_i)$ with n nodes each. We choose $\lambda = 3$ and use Lemma 14 to construct a tree decomposition $(\mathcal{B}, T)$ of G for $\alpha = 11 \cdot \lambda + 32 = O(1)$ and $\beta = 1/2 + 2^{-\lambda} = 5/8$. Afterwards, we use the solution on tree decompositions for $(\mathcal{B}, T)$. We have the following theorem.

▶ **Theorem 17.** *Let $G = (V, E)$ be the product graph of two constant treewidth graphs $G_i = (V_i, E_i)$, for $1 \leq i \leq 2$, with n nodes each. Let $\mathsf{wt}\colon E \to \mathbb{Z}$ be a weight function on G. The minimum initial credit for energy problem on (G, wt) can be solved in $O(n^5)$ time.*

Proof. The correctness follows directly from [20], and here we focus on the complexity. Similarly to Theorem 15, every bottom-up traversal of the tree decomposition $(\mathcal{B}', T)$ requires $O(n^3)$ time. Since every time we construct G^{i+1} from G^i we remove one node, we have $i \leq n^2$, i.e., there will be at most n^2 iterations of non-positive-cycle detection. Hence we make at most $n^2 + 1$ bottom-up traversals of $(\mathcal{B}', T)$, for a total running time of $O(n^5)$. The desired result follows. ◀

The best existing solution for the problem is due to [20], which has running time $O((n^2)^4) = O(n^8)$. Hence, Theorem 17 yields an improvement by a factor n^3, asymptotically.

5 Conclusion

Product graphs have numerous applications in verification, such as in the analysis of concurrent systems, as well as in language inclusion problems, which arise frequently in model checking. In this work, we have studied product graphs of two components w.r.t. three classic specification languages that arise in verification, namely semiring, mean-payoff, and initial credit for energy properties. We have studied these problems under the consideration that the components are specified as low-treewidth graphs, a property that is met by control-flow graphs of programs and has also found applications in logic, most notably due to the celebrated theorem of Courcelle for MSO. Our results show that these problems admit faster solutions than existing approaches, and in the case of semiring properties, our algorithm is optimal. At the heart of our new algorithms lies the newly introduced concept of (α, β) tree decompositions, which have a strong balancing property while suffering a small factor increase in their width. Moreover, we have shown that for balanced tree decompositions, such a factor increase in the width is generally unavoidable. Finally, we have developed an algorithm for constructing (α, β) tree decompositions efficiently for low-treewidth graphs.

References

1 Luca Aceto, A. Ingólfsdóttir, Mohammad Reza Mousavi, and M. A. Reniers. Algebraic properties for free! *Bulletin of the European Association for Theoretical Computer Science*, 99:81–103, 2009.

2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, April 1991.

3 C. Baier and J-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

4 Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. *Journal of Computer and System Sciences*, 48(2), 1994.

5 Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *Computer Aided Verification*, pages 140–156, 2009.

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

7 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. In Zoltán Fülöp and Ferenc Gécseg, editors, *Automata, Languages and Programming*, pages 268–279, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

8 Mikołaj Bojańczyk and Michał Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 407–416, New York, NY, USA, 2016. ACM.

9 Udi Boker, Krishnendu Chatterjee, Thomas A. Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. *ACM TOCL*, 15(4):27:1–27:25, 2014.

10 Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. Timed automata with observers under energy constraints. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '10, pages 61–70, New York, NY, USA, 2010. ACM.

11 Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In *Formal Modeling and Analysis of Timed Systems*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer Berlin Heidelberg, 2008.

12 Patricia Bouyer, Nicolas Markey, and Raj Mohan Matteplackel. Averaging in LTL. In *CONCUR 2014*, pages 266–280, 2014.

13 Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim G. Larsen, and Simon Laursen. Average-energy games. *Acta Informatica*, 55(2):91–127, March 2018.

14 Pavol Cerný, Krishnendu Chatterjee, Thomas A. Henzinger, Arjun Radhakrishna, and Rohit Singh. Quantitative synthesis for concurrent programs. In *Computer Aided Verification*, pages 243–259, 2011.

15 Krishnendu Chatterjee, Bhavya Choudhary, and Andreas Pavlogiannis. Optimal dyck reachability for data-dependence and alias analysis. *PACMPL*, 2(POPL):30:1–30:30, 2018.

16 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Expressiveness and closure properties for quantitative languages. *LMCS*, 6(3), 2010.

17 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM TOCL*, 11(4):23, 2010.

18 Krishnendu Chatterjee, Amir Kafshdar Goharshady, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. Optimal and perfectly parallel algorithms for on-demand data-flow analysis. In Peter Müller, editor, *ETAPS (ESOP)*, volume 12075 of *Lecture Notes in Computer Science*, pages 112–140. Springer, 2020.

19 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Amir Kafshdar Goharshady, and Andreas Pavlogiannis. Algorithms for algebraic path properties in concurrent systems of constant treewidth components. *ACM Trans. Program. Lang. Syst.*, 40(3):9:1–9:43, 2018. doi:10.1145/3210257.

20 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. Faster algorithms for quantitative verification in constant treewidth graphs. In Daniel Kroening and Corina S. Păsăreanu, editors, *Computer Aided Verification*, pages 140–157, Cham, 2015. Springer International Publishing.

21 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Andreas Pavlogiannis, and Prateesh Goyal. Faster algorithms for algebraic path properties in recursive state machines with constant treewidth. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 97–109, New York, NY, USA, 2015. ACM.

22 Krishnendu Chatterjee, Andreas Pavlogiannis, and Yaron Velner. Quantitative interprocedural analysis. In *Principles of Programming Languages, POPL 2015*, pages 539–551, 2015.

23 Shiva Chaudhuri and Christos D. Zaroliagis. Shortest Paths in Digraphs of Small Treewidth. Part I: Sequential Algorithms. *Algorithmica*, 27:212–226, 1995.

24 Ravi Chugh, Jan W. Voung, Ranjit Jhala, and Sorin Lerner. Dataflow analysis for concurrent programs using datarace detection. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI, 2008.

25 Fan RK Chung. *Separator theorems and their applications.* Universität Bonn. Institut für Ökonometrie und Operations Research, 1988.

26 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85, 1990.

27 Arnab De, Deepak D'Souza, and Rupesh Nasre. Dataflow analysis for datarace-free programs. In *Proceedings of the 20th European Conference on Programming Languages and Systems: Part of the Joint European Conferences on Theory and Practice of Software*, ESOP'11/ETAPS'11, pages 196–215. Springer-Verlag, 2011.

28 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata.* Springer, 1st edition, 2009.

29 Manfred Droste and Ingmar Meinecke. Weighted automata and weighted MSO logics for average and long-time behaviors. *Inf. Comput.*, 220:44–59, 2012.

30 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 143–152, USA, 2010. IEEE Computer Society.

31 Uli Fahrenberg, Line Juhl, Kim G. Larsen, and Jiří Srba. Energy games in multiweighted automata. In *Proceedings of the 8th International Conference on Theoretical Aspects of Computing*, ICTAC'11, pages 95–115, Berlin, Heidelberg, 2011. Springer-Verlag.

32 Azadeh Farzan and P. Madhusudan. Causal dataflow analysis for concurrent programs. In *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS, 2007.

33 J. Filar and K. Vrieze. *Competitive Markov Decision Processes.* Springer-Verlag, 1997.

34 Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

35 Fedor V. Fomin, Daniel Lokshtanov, MichałPilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1419–1432, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics.

36 Dirk Grunwald and Harini Srinivasan. Data flow equations for explicitly parallel programs. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP, 1993.

37 Jens Gustedt, OleA. Maehle, and JanArne Telle. The treewidth of java programs. In *Algorithm Engineering and Experiments*, volume 2409 of *Lecture Notes in Computer Science*, pages 86–97. Springer Berlin Heidelberg, 2002.

38 C. A. Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent kleene algebra. In *Proceedings of the 20th International Conference on Concurrency Theory*, CONCUR 2009, pages 399–414, Berlin, Heidelberg, 2009. Springer-Verlag.

39 Peter Jipsen. Concurrent kleene algebra with tests. In Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller, editors, *Relational and Algebraic Methods in Computer Science*, pages 37–48, Cham, 2014. Springer International Publishing.

40 Line Juhl, Kim Guldstrand Larsen, and Jean-François Raskin. Optimal bounds for multiweighted and parametrised energy games. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, *Theories of Programming and Formal Methods*, pages 244–255, Berlin, Heidelberg, 2013. Springer-Verlag.

41 Vineet Kahlon, Nishant Sinha, Erik Kruus, and Yun Zhang. Static data race detection for concurrent programs with asynchronous calls. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 13–22, 2009.

42 Tobias Kappé, Paul Brunet, Alexandra Silva, and Fabio Zanasi. Concurrent kleene algebra: Free model and completeness. In Amal Ahmed, editor, *Programming Languages and Systems*, pages 856–882, Cham, 2018. Springer International Publishing.

43 Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 1978.

44 S. C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, 1956.

45 Jens Knoop, Bernhard Steffen, and Jürgen Vollmer. Parallelism for free: Efficient and optimal bitvector analyses for parallel programs. *ACM Trans. Program. Lang. Syst.*, 1996.

46 J. Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 173–182 vol.1, 1990.

47 Daniel J. Lehmann. Algebraic structures for transitive closure. *Theoretical Computer Science*, 1977.

48 Jan Obdrzálek. Fast mu-calculus model checking when tree-width is bounded. In *CAV*, 2003.

49 M.L. Puterman. *Markov Decision Processes.* John Wiley and Sons, 1994.

50 Bruce A. Reed. Finding approximate separators and computing tree width quickly. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, 1992.

51 Thomas Reps, Susan Horwitz, and Mooly Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL*, New York, NY, USA, 1995. ACM.

52 Neil Robertson and P.D Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

53 Mikkel Thorup. All Structured Programs Have Small Tree Width and Good Register Allocation. *Information and Computation*, 142(2):159–181, 1998.

54 Stephen Warshall. A Theorem on Boolean Matrices. *J. ACM*, 9(1):11–12, January 1962.

A Proofs of Section 3

Here we present the proofs of Section 3. The structure of this section follows the structure of Section 3.

A.1 Proofs of Section 3.1

Here we prove Lemma 5 and Lemma 6 which state the correctness and complexity of the two operations on components $\mathsf{BorderSplit}$ and Split, respectively.

Proof of BorderSplit. Here we prove Lemma 5 which concerns the correctness and complexity of the operation $\mathsf{BorderSplit}$.

▶ **Lemma 5.** *Consider the operation* $\mathsf{BorderSplit}$ *on the set-component* $\mathcal{X} = \{\mathcal{C}_i\}_{1\leq i\leq k}$. *Let* $z = |\mathsf{size}(\mathcal{X})|$, $m = |\mathsf{Border}(\mathcal{X})|$, *and* $\{\mathcal{X}_i\}_{1\leq i\leq 3}$ *be the returned component set. The following assertions hold.*

1. *We have* $|\bigcup_i \mathsf{Border}(\mathcal{X}_i)| \leq m+1$.
2. *For each* $1 \leq i \leq 3$, *we have* $|\mathsf{Border}(\mathcal{X}_i)| \leq \lfloor m/2 \rfloor + 1$.
3. *After one* $O(n)$*-time preprocessing of* T, *every call to* $\mathsf{BorderSplit}$ *requires* $O(z+m^2)$ *time.*

Proof. We prove each assertion separately.

1. This item holds trivially, since we remove at most one node u from $\mathcal{X}$.
2. If $m \geq 3$, by construction, we have $|A_i \cap \mathsf{Border}(\mathcal{X})| \leq \lfloor m/2 \rfloor$ for each i. In addition, $\mathsf{Border}(\mathcal{X}_i) \subseteq (A_i \cap \mathsf{Border}(\mathcal{X})) \cup \{u\}$ and thus $|\mathsf{Border}(\mathcal{X}_i)| \leq \lfloor m/2 \rfloor + 1$. Finally, if $|\mathsf{Border}(\mathcal{X})| < 3$ then the algorithm simply returns $\{\mathcal{X}\}$ and thus $|\mathsf{Border}(\mathcal{X}_i)| = m \leq \lfloor m/2 \rfloor + 1$.
3. Since T is binary, we have $m = O(z)$, thus obtaining the set $\mathsf{Border}(\mathcal{X})$ requires $O(z)$ time. Note that the LCA tree $\mathcal{T}_{\mathsf{Border}(\mathcal{C}_j)}$ contains $O(m)$ nodes, as it is binary and it has $O(m)$ leaves. It is known that T can be preprocessed in $O(n)$ time, after which LCA queries can be answered in $O(1)$ time [4]. It follows easily that $\mathcal{T}_{\mathsf{Border}(\mathcal{C})}$ can be constructed in $O(m^2)$ time, by performing $O(m^2)$ LCA queries. Determining the desired node u can easily be done in $O(m)$ time. Finally, constructing each connected component $\mathcal{C}_i$ can be easily done in $O(m)$ time.

The desired result follows. ◀

Proof of Split. Here we prove Lemma 6 which concerns the correctness and complexity of the operation Split.

Towards the proof of Lemma 6, we will present some simple lemmas. In the end, we will combine these lemmas in the proof of Lemma 6. Recall that Split operates on a component $\mathcal{X}$, and given a positive integer λ. We also let $z = \mathsf{size}(\mathcal{X})$ and $m = |\mathsf{Border}(\mathcal{X})|$. We start with a simple lemma that will be used frequently.

▶ **Lemma 18.** *Consider integers $x_1, x_2, x_{3,x}4$. If $x_1 \geq x_2$ and $x_3 \geq x_4$ and $x_1+x_2 \geq x_3+x_4$, then $x_4 \leq x_1$.*

Proof. The proof is trivial:

$$x_4 \leq \frac{x_3+x_4}{2} \leq \frac{x_1+x_2}{2} \leq x_1$$ ◀

The following lemma states the properties of the first step of Split.

▶ **Lemma 19.** *At the end of first step, either $m < 10$ and $\mathcal{C}^* = \mathcal{X}$ and $\mathcal{X}_1 = \mathcal{X}_2 = \emptyset$, or each $\mathcal{X}_i$ is border-balancing.*

Proof. The claim clearly holds if $m < 10$. Now consider that $m \geq 10$, and the algorithm performs an operation BorderSplit on $\mathcal{X}$ to obtain the components $\{\mathcal{X}'_i\}_{1\leq i\leq 3}$. If there are two border-balancing components among all $\mathcal{X}'_i$ then the claim holds by construction.

Now assume that there are no two such components, and hence the algorithm proceeds with performing the operation BorderSplit on the component $\mathcal{X}'_1$. We first argue that in the set $A = \{\mathcal{X}'_2, \mathcal{X}'_3, \mathcal{X}''_2, \mathcal{X}''_3\}$ there are at least two border-balancing components, $\mathcal{X}_a$ and $\mathcal{X}_b$. Since $\mathcal{X}'_1$ is the largest component among $\mathcal{X}'_i$, we have that $\mathsf{size}(\mathcal{X}'_3) \leq \mathsf{size}(\mathcal{X}'_2) \leq z/2$. Also, by Lemma 5, we have $|\mathsf{Border}(\mathcal{X}'_1)| \leq m/2+1$, thus at least one of $\mathcal{X}'_2, \mathcal{X}'_3$ has border with size at least

$$\frac{m-(m/2+1)}{2} = \frac{m}{4} - \frac{1}{2} \geq \frac{m}{10} - 1 \tag{6}$$

Hence at least one of $\mathcal{X}'_2, \mathcal{X}'_3$ is border-balancing. We take that component to be $\mathcal{X}_a$.

Second, we argue that at least one of $\mathcal{X}''_2, \mathcal{X}''_3$ is border-balancing. A similar argument to the previous case shows that $\mathsf{size}(\mathcal{X}''_3) \leq \mathsf{size}(\mathcal{X}''_2) \leq z/2$. Let $m' = |\mathsf{Border}(\mathcal{X}'_1)|$. Since one of $\mathcal{X}'_1, \mathcal{X}'_2$ is not border-balancing and by Lemma 5 the other has border of size at most $m/2+1$, we have that

$$m' \geq m - \left(\frac{m}{2}+1\right) - \left(\frac{m}{10}-1\right) = \frac{4}{10}\cdot m \tag{7}$$

A similar argument as before shows that at least one of $\mathcal{C}''_2, \mathcal{C}''_3$ has border with size at least

$$\frac{m'-(m'/2+1)}{2} = \frac{m'}{4} - \frac{1}{2} \geq \frac{m}{10} - 1/2 \tag{8}$$

and thus it is border-balancing. We take that component to be $\mathcal{X}_b$.

It remains to argue that $\mathsf{size}(\mathcal{X}_i) \leq z/2$ for each $1 \leq i \leq 2$. Since each $\mathcal{X}_a$ and $\mathcal{X}_b$ are border-balancing, the claim clearly holds after we have added $\mathcal{X}_a$ to $\mathcal{X}_1$ and $\mathcal{X}_b$ to $\mathcal{X}_2$. Assume w.l.o.g. that $\mathsf{size}(\mathcal{X}_1) \leq \mathsf{size}(\mathcal{X}_2)$, and let $x_1 = m/2 - \mathsf{size}(\mathcal{X}_1)$ and $x_2 = m/2 - \mathsf{size}(\mathcal{X}_2)$, thus $x_1 \geq x_2$. Let $x_3 = \mathsf{size}(\mathcal{X}^*)$, and x_4 be the size of any other component in $A \setminus \{\mathcal{X}^*, \mathcal{X}_a, \mathcal{X}_b\}$, and by definition $x_3 \geq x_4$. Observe that $x_1 + x_2 \geq x_3 + x_4$ since $x_1 + x_2$ is at least as large as $|A \setminus \{\mathcal{X}_a, \mathcal{X}_b\}|$. It follows by Lemma 18 that we can add the component with size x_4 to $\mathcal{X}_1$ while ensuring that the size of $\mathcal{X}_1$ stays at most $z/2$ after this operation. Similarly for adding the second component in the smaller of $\mathcal{X}_1, \mathcal{X}_2$. It follows that at the end of the second step, each $\mathcal{X}_1, \mathcal{X}_2$ is border-balancing, as desired. ◀

We now turn our attention to the second step of Split. The following lemma is straightforward.

▶ **Lemma 20.** *For all j we have that $\mathsf{size}(\mathcal{C}^{j+1}) \leq \mathsf{size}(\mathcal{C}^j)/2$ and thus $\mathsf{size}(\mathcal{C}^j) \leq z \cdot 2^{-j}$.*

Proof. The lemma follows since $\mathcal{C}^{j+1} = \mathcal{C}_1^j$, and due to Lemma 1 we have $|\mathcal{C}_1^j| \leq |\mathcal{C}^j|/2$. ◀

We now show that throughout the recursion of the second step, each of $\mathcal{X}_1, \mathcal{X}_2$ has size at most $z/2$.

▶ **Lemma 21.** *Until (but not including) the very end of the second step, we have* $\mathsf{size}(\mathcal{X}_1), \mathsf{size}(\mathcal{X}_2) \leq z/2$.

Proof. The statement holds at the beginning of the second step since by Lemma 19 each of $\mathcal{X}_1, \mathcal{X}_2$ is border-balancing. The statement also holds at the beginning of the recursion, since $\mathcal{C}$ is the largest connected component of $\mathcal{X}^*$, by Lemma 18 similarly to the proof of Lemma 19. Now consider the recursive step j. The statement follows by the induction hypothesis and the fact that $\mathcal{C}_1^j$ is the largest connected component among $\{\mathcal{C}_i^j\}_{1\leq i\leq 3}$, as above. ◀

Finally, we conclude with the proof of Lemma 6.

Proof of Lemma 6. We prove each item separately.

1. Lemma 20 and Lemma 21 together ensure that $\mathsf{size}(\mathcal{X}_i) \leq z\cdot(1/2+2^{-\lambda})$ for each $1 \leq i \leq 2$ (because we add $\mathcal{C}^\lambda$ to either $\mathcal{X}_1$ or $\mathcal{X}_2$ at the very end of the second step).
2. First assume that $m \geq 10$. By Lemma 19, we have that each of $\mathcal{X}_1, \mathcal{X}_2$ is border-balancing, thus $\mathsf{Border}(\mathcal{X}_i) \geq m/10 - 1$ for each $1 \leq i \leq 2$. Also, for each $1 \leq i \leq 2$, we have $|\mathsf{Border}(\mathcal{X}_i) \cap (\mathsf{Border}(\mathcal{X}_{3-i}) \cup \mathsf{Border}(\mathcal{X}^*)| \leq 2$. This follows from the fact that every time we apply the operation $\mathsf{BorderSplit}$, we create three components and the intersection between the borders of any pair of components is either empty, or it is the singleton $\{u\}$, where the node u is defined in $\mathsf{BorderSplit}$. The inequality then holds as the components $\mathcal{X}_1$, $\mathcal{X}_2$ and $\mathcal{X}^*$ are created by applying $\mathsf{BorderSplit}$ at most twice. Let Q_i and Q_i' denote the border of component $\mathcal{X}_i$ at the beginning of the second step, and before the recursive process of the second step, hence the previous inequality can be written as $|Q_i \cap (Q_{3-i} \cup \mathsf{Border}(\mathcal{X}^*))| \leq 2$. Thus, we have $|Q_i \setminus (Q_{3-i} \cup \mathsf{Border}(\mathcal{X}^*))| \geq m/10 - 3$. In addition, we have $Q_i' \subseteq Q_i \cup \mathsf{Border}(\mathcal{X}^*)$, and thus $Q_i' \leq 9/10 \cdot m + 3$. Finally, in each recursive call of the second step we create at most one new border node, which is the balancing separator of the connected component $\mathcal{C}^j$. Hence after λ recursive calls, we have created at most λ new border nodes. It follows that at the end of the second step, we have $|\mathsf{Border}(\mathcal{X}_i)| \leq 9/10 \cdot m + \lambda + 3$.
   Now assume that $m < 10$. Then clearly $\mathsf{Border}(\mathcal{X}_i) \leq m + \lambda \leq 9/10 \cdot m + \lambda + 3$.
3. This part is trivial, since Split removes at most $\lambda + 2$ nodes when creating the components $\mathcal{X}_1$ and $\mathcal{X}_2$.
4. The first step takes $O(z + m^2)$ time, by Lemma 5. Each recursive call of the second step takes $O(|\mathcal{X}^j|)$ time, by Lemma 1. By Lemma 20 the size of $\mathcal{X}^j$ halves in each call, hence the time for the second step is $O(|\mathcal{X}^*|) = O(z)$. ◀

A.2 Proofs of Section 3.2

Here we prove Lemma 8 and Lemma 9 which concern the correctness and complexity of $\mathsf{Balance}$. We start with an auxiliary lemma, which states a bound on the size of the border of each component of the component tree constructed in the first step of $\mathsf{Balance}$.

▶ **Lemma 22.** *Consider the component tree $(\mathcal{V}, \mathcal{R} = (\mathcal{J}, \mathcal{D}))$ constructed by* $\mathsf{Balance}$. *For every $i \in \mathcal{J}$, we have* $|\mathsf{Border}(\mathcal{X}_i)| \leq 10 \cdot (\lambda + 3)$.

Proof. The claim clearly holds for i being the root of $\mathcal{R}$, since in that case $|\mathsf{Border}(\mathcal{X}_i)| = 0$. Now we will argue that the claim holds for any $i \in \mathcal{J}$, assuming that it holds for the parent j of i in $\mathcal{R}$.

Indeed, let $m = |\mathsf{Border}(\mathcal{X}_j)|$ and $m' = |\mathsf{Border}(\mathcal{X}_i)|$. By Lemma 6 we have that

$$m' \leq \frac{9}{10} \cdot m + \lambda + 3 \leq \frac{9}{10} \cdot 10 \cdot (\lambda + 3) + \lambda + 3 = 10 \cdot (\lambda + 3) \tag{9}$$

The desired result follows. ◀

We now turn our attention to Lemma 8 which concerns the correctness of Balance.

▶ **Lemma 8.** *$(\mathcal{B}, \mathcal{T} = (\mathcal{I}, \mathcal{F}))$ is a tree decomposition of T that has width $\leq \alpha$ and is β-balanced, for $\alpha = 11 \cdot \lambda + 32$ and $\beta = 1/2 + 2^{-\lambda}$.*

Proof. We first argue that $(\mathcal{B}, \mathcal{T})$ is a tree decomposition of T, and then that it has width $\leq \alpha$ and is β-balanced.

First, consider any edge $(u, v) \in F$, and assume w.l.o.g. that u is the parent of v. Observe that T has the (empty) connected component $\mathcal{C}$ with $\mathsf{Border}(\mathcal{C}) = \{u, v\}$. It follows easily that there exists a leaf i of $\mathcal{R}$ such that $\mathcal{C} \in \mathcal{X}_i$ and thus $\{u, v\} \subseteq \mathsf{Border}(\mathcal{X}_i)$. By construction, $u, v \in B_j$, where j is the parent of i in $\mathcal{R}$. Hence, $(\mathcal{B}, \mathcal{R})$ satisfies condition 2 of tree decompositions. Since T is connected, it follows that condition 1 is also satisfied.

We now turn our attention into showing that every node u appears in a contiguous subtree of $(\mathcal{B}, \mathcal{R})$, which will prove that $(\mathcal{B}, \mathcal{T})$ satisfies condition 3 of tree decompositions. First, observe that there exists a lowest-level node $i \in \mathcal{I}$ such that $u \in \mathcal{X}_i$, and in fact $u \in B_i$, as $u \in \mathsf{Border}(\mathcal{X}_j)$ for some child j of i. In addition, note that for every bag B_j with $u \in B_j$, we have that j is a descendant of i. Finally, consider any strict descendant j of i in $\mathcal{R}$ such that $u \notin B_j$. It follows that (i) $u \notin \mathcal{X}_j$ (by our choice of i) and (ii) $u \notin \mathsf{Border}(\mathcal{X}_j)$ (since $u \notin B_j$). It is straightforward to see that $u \notin B_{j'}$ for any descendant j' of j in $\mathcal{R}$, which concludes the condition 3.

We now turn our attention in showing that $(\mathcal{B}, \mathcal{T})$ has width $\leq \alpha$ and is β-balanced. We first argue about the width. Consider the component tree $(\mathcal{V}, \mathcal{R} = (\mathcal{J}, \mathcal{D}))$ constructed by Balance in the first step. Consider any node $i \in \mathcal{J}$ that is not a leaf, and let j_1, j_2 be the children of i in $\mathcal{R}$. By Lemma 22, we have $|\mathsf{Border}(\mathcal{X}_i)| \leq 10 \cdot (\lambda + 3)$. By Lemma 6, we have

$$|\mathsf{Border}(\mathcal{X}_{j_1}) \cup \mathsf{Border}(\mathcal{X}_{j_2})| \leq |\mathsf{Border}(\mathcal{X}_i)| + \lambda + 2 \leq 11 \cdot \lambda + 32$$

By construction, we have $B_i = \mathsf{Border}(\mathcal{X}_{j_1}) \cup \mathsf{Border}(\mathcal{X}_{j_2})$, and thus $|B_i| \leq 11 \cdot \lambda + 32$.

Finally, we show that $(\mathcal{B}, \mathcal{T})$ is β-balanced. By Lemma 6, for every $i \in \mathcal{I}$ and j child of i in $\mathcal{R}$, we have $\mathsf{size}(\mathcal{X}_j) \leq \mathsf{size}(\mathcal{X}_i) \cdot (1/2 + 2^{-\lambda})$, and since the size of the root component of $\mathcal{R}$ is $|I|$, we have $\mathsf{size}(\mathcal{X}_j) \leq |I| \cdot (1/2 + 2^{-\lambda})^{\mathsf{Lv}(j)}$. By Remark 7, we have $\mathcal{Y}_{\mathcal{B}}^{\mathcal{T}}(j) = \mathsf{size}(\mathcal{X}_j) \leq |I| \cdot (1/2 + 2^{-\lambda})^{\mathsf{Lv}(j)}$, as required.

The desired result follows. ◀

Finally, we prove Lemma 9 which captures the running time of Balance.

Proof of Lemma 9. We start with the first step of Balance. Since $\lambda \geq 2$, by Lemma 6 the size of each component decreases by at least a constant factor with each recursive call, hence the first step is executed for $O(\log |I|)$ levels. By the same lemma, and since the border of each component has size $O(\lambda)$ (by Lemma 22), each level in this recursion runs in $O(\lambda^2 \cdot |I|)$. Hence the first step runs in $O(\lambda^2 \cdot |I| \cdot \log |I|)$ time. In the second step, the tree decomposition is easily constructed in $O(\alpha \cdot |I|) = O(\lambda \cdot |I|)$ time.

The desired result follows. ◀

Synthesizing Computable Functions from Rational Specifications over Infinite Words

Emmanuel Filiot
Université libre de Bruxelles, Brussels, Belgium

Sarah Winter
Université libre de Bruxelles, Brussels, Belgium

Abstract

The synthesis problem asks to automatically generate, if it exists, an algorithm from a specification of correct input-output pairs. In this paper, we consider the synthesis of computable functions of infinite words, for a classical Turing computability notion over infinite inputs. We consider specifications which are rational relations of infinite words, i.e., specifications defined by non-deterministic parity transducers. We prove that the synthesis problem of computable functions from rational specifications is undecidable. We provide an incomplete but sound reduction to some parity game, such that if Eve wins the game, then the rational specification is realizable by a computable function. We prove that this function is even computable by a deterministic two-way transducer.

We provide a sufficient condition under which the latter game reduction is complete. This entails the decidability of the synthesis problem of computable functions, which we proved to be ExpTime-complete, for a large subclass of rational specifications, namely deterministic rational specifications. This subclass contains the class of automatic relations over infinite words, a yardstick in reactive synthesis.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Automata over infinite objects; Theory of computation → Transducers

Keywords and phrases uniformization, synthesis, transducers, continuity, computability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.43

Related Version A preliminary version of this work is hosted on arXiv under a different title.
Preliminary Version: `https://arxiv.org/abs/2103.05674v1`
Full Version: `https://arxiv.org/abs/2103.05674`

Funding *Emmanuel Filiot*: This work is partially supported by the MIS project F451019F (F.R.S.-FNRS). Emmanuel Filiot is a research associate at F.R.S.-FNRS.
Sarah Winter: Sarah Winter is a postdoctoral researcher at F.R.S.-FNRS.

Acknowledgements The authors want to thank Nathan Lhote for a discussion on two-way transducers and Martin Zimmermann for pointing out a reference.

1 Introduction

Program synthesis aims at automatically generating programs from specifications. This problem can be formalized as follows. There are four parameters: two sets of input and output domains I, O, a set $\mathcal{S}$ of relations (called specifications) from I to O, and a set $\mathcal{I}$ of (partial) functions (called implementations) from I to O. Then, given a specification $S \in \mathcal{S}$ defining the correct input/output relationships, the synthesis problem asks to check whether there exists a function $f \in \mathcal{I}$ satisfying S in the following sense: its graph is included in S and it has the same domain as S (i.e., f is defined on $x \in I$ iff $(x, y) \in S$ for some $y \in O$). Using a set-theoretic terminology, f is said to *uniformize* S. Moreover in synthesis, if such an f exists, then the synthesis algorithm should return (a finite presentation of) it.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 43; pp. 43:1–43:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program synthesis quickly turns to be undecidable depending on the four parameters mentioned before. Therefore, research on synthesis either turn to developing efficient sound but incomplete methods, see for example the syntax-guided synthesis approach [2] or bounded synthesis [14, 15], or restrict the class of specifications $\mathcal{S}$ and/or the class of implementations $\mathcal{I}$. A well-known example of the latter approach is reactive synthesis, where $\mathcal{S}$ are automatic relations[1] over infinite words, and $\mathcal{I}$ are Mealy machines [6, 23, 10]. Infinite words (over a finite alphabet) are used to model infinite executions of reactive systems, and Mealy machines are used as a model of reactive systems processing bit streams.

In this paper, our goal is to synthesize, from specifications which are semantically binary relations of infinite words, stream-processing programs, which are semantically *streaming computable* functions of infinite words (just called *computable* functions in the sequel). Let us now make the computability notion we use more precise. Let Σ and Γ be to finite alphabets. A partial function $f\colon \Sigma^\omega \to \Gamma^\omega$, whose domain is denoted $\text{dom}(f)$, is said to be computable, if there exists a deterministic (Turing) machine M with three tapes, a read-only one-way input tape, a two-way working tape, and a write-only output tape that works as follows: if the input tape holds an input sequence $\alpha \in \text{dom}(f)$, then M outputs longer and longer prefixes of $f(\alpha)$ when reading longer and longer prefixes of α. A definition of this machine model can be found, for instance, in [25].

▶ **Example 1.** Over the alphabet $\Sigma = \Gamma = \{a, b, A, B\}$, consider the specification given by the relation $R_1 = \{(ux\alpha, xu\beta) \mid u\alpha, u\beta \in \{a, b\}^\omega, x \in \{A, B\}\}$. The relation R_1 is automatic: an automaton needs to check that the input prefix u occurs shifted by one position on the output, which is doable using only finite memory. Checking that the first output letter x also appears after u on the input can also be done by storing x in the state. Note that some acceptance condition (e.g., parity) is needed to make sure that x is met again on the input. There is no Mealy machine which can realize R_1, because Mealy machines operate in a synchronous manner: they read one input symbol and must deterministically produce one output symbol. Here, the first output symbol which has to be produced depends on the letter x which might appear arbitrarily far in the input sequence. However, R_1 can be uniformized by a computable function: there is an *algorithm* reading the input from left to right and which simply waits till the special symbol $x \in \{A, B\}$ is met on the input. Meanwhile, it stores longer and longer prefixes of u in memory (so it needs unbounded memory) and once x is met, it outputs xu. Then, whatever it reads on the input, it just copies it on the output (realizing the identity function over the remaining infinite suffix α). Note that this algorithm produces a correct output under the assumption that x is eventually read.

Contributions. We first investigate the synthesis of computable functions from *rational specifications*, which are those relations recognizable by non-deterministic finite state transducers, i.e., parity automata over a product of two free monoids. We however show this problem is undecidable (Proposition 4). We then give an incomplete but sound algorithm in Section 3, based on a reduction to ω-regular two-player games. Given a transducer $\mathcal{T}$ defining a specification $\mathcal{R}_\mathcal{T}$, we show how to effectively construct a two-player game $\mathcal{G}_\mathcal{T}$, proven to be solvable in ExpTime, such that if Eve wins $\mathcal{G}_\mathcal{T}$, then there exists a computable function which uniformizes the relation recognized by $\mathcal{T}$, which can even be computed by some input-deterministic **two-way** finite state transducer (a transducer which whenever it reads an input symbol, it deterministically produces none or several output symbols and either moves forward or backward on the input). It is easily seen that two-wayness is

[1] relations recognized by two-tape parity automata alternatively reading one input and one output symbol.

necessary: the relation R_1 cannot be uniformized by any deterministic device which moves only forward over the input and only uses finitely many states, as the whole prefix u has to be remembered before reaching x. However, a two-way finite-state device can do it: first, it scans the prefix up to x, comes back to the beginning of the input, knowing whether $x = A$ or $x = B$, and then can produce the output.

Intuitively, in the two-player game we construct, called unbounded delay game, Adam picks the input symbols while Eve picks the output symbols. Eve is allowed to delay her moves an arbitrarily number of steps, gaining some lookahead information on Adam's input. We use a finite abstraction to store the lookahead gained on Adam's moves. We show that any finite-memory winning strategy in this game can be translated into a function uniformizing the specification such that it is computable by an input-deterministic two-way transducer.

In Section 4, we provide a sufficient condition $\mathcal{P}$ on relations for which the game reduction is complete. In particular, we show that if a relation R satisfies $\mathcal{P}$, then Eve wins the game iff R can be uniformized by a computable function. A large subclass of rational relations satisfying this sufficient condition is the class of deterministic rational relations (DRAT, [24]). Deterministic rational relations are those relations recognizable by deterministic two-tape automata, one tape holding the input word while the other holds the output word. It strictly subsumes the class of automatic relations, and, unlike for automatic relations, the two heads are not required to move at the same speed. Furthermore, when the domain of the relation is topologically closed for the Cantor distance[2], we show that strategies in which Eve delays her moves at most a bounded number of steps are sufficient for Eve to win. Such a strategy can in turn be converted into an input-deterministic **one-way** transducer. This entails that for DRAT-specifications with a closed domain (such as for instance specifications with domain Σ^ω, i.e., total domains), if it is uniformizable by a computable function, then it is uniformizable by a function computable by an input-deterministic one-way transducer.

Based on the completeness result, we prove our main result, that the synthesis problem of computable functions from deterministic rational relations is ExpTime-complete. Hardness also holds in the particular case of automatic relations of total domain.

Total versus partial domains. We would like to emphasize here on a subtle difference between our formulation of synthesis problems and the classical formulation in reactive synthesis. Classically in reactive synthesis, it is required that a controller produces for *every* input sequence an output sequence correct w.r.t. the specification. Consequently, specifications with partial domain are by default unrealizable. So, in this setting, the specification R_1 of the latter example is not realizable, simply because its domain is not total (words with none or at least two occurrences of a symbol in $\{A, B\}$ are not in its domain). In our definition, specifications with partial domain can still be realizable, because the synthesized function, if it exists, can be partial and must be defined only on inputs for which there exists at least one matching output in the specification. A well-known notion corresponding to this weaker definition is that of uniformization [21, 9, 7, 13], this is why we often use the terminology "uniformizes" instead of the more widely used terminology "realizes". The problem of synthesizing functions which uniformize quantitative specifications has been recently investigated in [1]. In [1], it was called the good-enough synthesis problem, a controller being good-enough if it is able to compute outputs for all inputs for which there

[2] A set $X \subseteq \Sigma^\omega$ is *closed* if the limit, if it exists, of any sequence $(x_i)_i$ of infinite words in X is in X. The limit here is defined based on the Cantor distance, which, for any two infinite words u, v, is 0 if $u = v$ and otherwise $2^{-\ell}$ where ℓ is the length of their longest common prefix.

exists a least one matching output by the specification. The uniformization setting can be seen as an assumption that the input the program receives is not any input, but belongs to some given language. Related to that, there is a number of works on reactive synthesis under assumptions on the behavior of the environment [8, 4, 22, 11, 5].

Related work. To the best of our knowledge, this work is the first contribution which addresses the synthesis of algorithms from specifications which are relations over infinite inputs, and such that these algorithms may need unbounded memory, as illustrated by the specification R_1 for which any infinite-input Turing-machine realizing the specification needs unbounded memory. There are however two related works, in some particular or different settings.

First, in [12], the synthesis of computable functions has been shown to be decidable in the particular case of *functional* relations, i.e., graphs of functions. The main contribution of [12] is to prove that checking whether a function represented by a non-deterministic two-way transducer is computable is decidable, and that computability coincides with continuity (for the Cantor distance) for this large class of functions. The techniques of [12] are different to ours, e.g., games are not needed because output symbols functionally depends on input ones, even in the specification, so, there is not choice to be made by Eve.

Second, Hosch and Landweber [19] proved decidability of the synthesis problem of Lipschitz-continuous functions from automatic relations with *total* domain, Holtmann, Kaiser and Thomas [17] proved decidability of the synthesis of continuous functions from automatic relations with *total* domain, and Klein and Zimmermann [20] proved ExpTime-completeness for the former problem. So, we inherit the lower bound because automatic relations are particular DRAT relations, and as we show in the last section of the paper, the synthesis problem of computable functions is the same as the synthesis problem of continuous functions. We obtain the same upper bound as [20] for a more general class of specifications, namely DRAT, and in the more general setting of specifications with partial domain. As we show, total vs. partial domains make an important difference: two-way transducers may be necessary in the former case, while one-way transducers are sufficient in the latter. [17, 20] also rely on a reduction to two-player games called delay games, but for which bounded delay are sufficient. However, our game is built such that it accounts for the fact that unbounded delays can be necessary and it also monitors the domain, which is not necessary in [17, 20] because specifications have total domain. Accordingly, the main differences between [20] and our delay games are their respective winning objectives and correctness proofs. Another difference is that our game applies to the general class of rational relations, which are *asynchronous* (several symbols, or none, can correspond to a single input symbol) in contrast to automatic relations which are synchronous by definition.

Omitted and sketched proofs can be found in full in the full version.

2 Preliminaries

Words, languages, and relations. Let $\mathbb{N}$ denote the set of non-negative integers. Let Σ and Γ denote *alphabets* of elements called *letters* or *symbols*. A *word* resp. *ω-word* over Σ is an empty or non-empty finite resp. infinite sequence of letters over Σ. The empty word is denoted by ε, the length of a word by $|\cdot|$. Usually, we denote finite words by u, v, w, etc., and infinite words by α, β, γ, etc. Given an (in)finite word $\alpha = a_0a_1\cdots$ over Σ with $a_0, a_1, \dots \in \Sigma$, let $\alpha(i)$ denote the letter a_i, $\alpha(i{:}j)$ denote the infix $a_ia_{i+1}\cdots a_j$, $\alpha({:}i)$ the prefix $a_0a_1\cdots a_i$, and $\alpha(i{:})$ the suffix $a_ia_{i+1}\cdots$ for $i \leq j \in \mathbb{N}$. For two (in)finite words

α, β, let $\alpha \wedge \beta$ denote their longest common prefix. Let Σ^*, Σ^+, and Σ^ω denote the set of finite, non-empty finite, and infinite words over Σ, respectively. Let Σ^∞ denote $\Sigma^* \cup \Sigma^\omega$. A *language* resp. *$\omega$-language* L is a subset of Σ^* resp. Σ^ω, its set of prefixes is denoted by $\mathrm{Prefs}(L)$. A (binary) *relation* resp. *ω-relation* R is a subset of $\Sigma^* \times \Gamma^*$ resp. $\Sigma^\omega \times \Gamma^\omega$. An ω-relation is just called a relation when infiniteness is clear from the context. The domain $\mathrm{dom}(R)$ of a (ω)-relation R is the set $\{\alpha \mid \exists \beta \ (\alpha, \beta) \in R\}$. It is *total* if $\mathrm{dom}(R) = \Sigma^*$ resp. Σ^ω. Likewise, we define $\mathrm{img}(R)$ the image of R, as the domain of its inverse. A relation R is *functional* if for each $u \in \mathrm{dom}(R)$ there is at most one v such that $(u, v) \in R$. *By default in this paper, relations and functions are partial, i.e., are not necessarily total.*

Automata. A *parity automaton* is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta, c)$, where Q is a finite set of states, Σ a finite alphabet, $q_0 \in Q$ an initial state, $\Delta \subseteq Q \times \Sigma \times Q$ a transition relation, and $c\colon Q \to \mathbb{N}$ is a function that maps states to *priorities*, also called *colors*. A parity automaton is *deterministic* if its transition relation Δ is given as a transition function δ. We denote by δ^* the usual extension of δ from letters to finite words. A *run of $\mathcal{A}$ on a word $w \in \Sigma^\infty$* is a word $\rho \in Q^\omega$ such that $(\rho(i), w(i), \rho(i+1)) \in \Delta$ for all $0 \leq i \leq |w|$. A run on ε is a single state. We say that ρ begins in $\rho(0)$ and ends in $\rho(|w|)$ if w is finite. We define $\mathrm{Occ}(\rho)$ as the set of states that occur in ρ, $\mathrm{Inf}(\rho)$ as the set of states that occur infinitely often in ρ, and $c(\rho)$ as $c(\rho(0))c(\rho(1))\cdots$. A run ρ is *accepting* if $\rho \in Q^\omega$, $\rho(0) = q_0$ and $\max \mathrm{Inf}(c(\rho))$ is even. The language *recognized* by $\mathcal{A}$ is the set $L(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \text{there is an accepting run } \rho \text{ of } \mathcal{A} \text{ on } \alpha\}$. A language $L \subseteq \Sigma^\omega$ is called *regular* if L is recognizable by a parity automaton.

One-way transducers. A *transducer* (1NFT) is a tuple $\mathcal{T} = (Q, \Sigma, \Gamma, q_0, \Delta, c)$, where Q is finite state set, Σ and Γ are finite alphabets, $q_0 \in Q$ is an initial state, $\Delta \subseteq Q \times \Sigma^* \times \Gamma^* \times Q$ is a finite set of transitions, and $c\colon Q \to \mathbb{N}$ is a parity function. It is *input-deterministic* (1DFT) (also called *sequential* in the literature) if Δ is expressed as a function $Q \times \Sigma \to \Gamma^* \times Q$. A *finite non-empty run* ρ is a non-empty sequence of transitions of the form $(p_0, u_0, v_0, p_1)(p_1, u_1, v_1, p_2) \ldots (p_{n-1}, u_{n-1}, v_{n-1}, p_n) \in \Delta^*$. The *input (resp. output)* of ρ is $\alpha = u_0 \cdots u_{n-1}$ (resp. $\beta = v_0 \cdots v_{n-1}$). As shorthand, we write $\mathcal{T}\colon p_0 \xrightarrow{\alpha/\beta} p_n$. An *empty run* is denoted as $\mathcal{T}\colon p \xrightarrow{\varepsilon/\varepsilon} p$ for all $p \in Q$. Similarly, we define an *infinite run*. A run is *accepting* if it is infinite, begins in the initial state and satisfies the parity condition. In this paper, we also assume that for any accepting run ρ, its input and output are both infinite. This can be syntactically ensured with the parity condition. The relation *recognized* by $\mathcal{T}$ is $R(\mathcal{T}) = \{(\alpha, \beta) \mid \text{there is an accepting run of } \mathcal{T} \text{ with input } \alpha \text{ and output } \beta\}$. Note that with the former assumption, we have $R(\mathcal{T}) \subseteq \Sigma^\omega \times \Gamma^\omega$. A relation is called *rational* if it is recognizable by a transducer, we denote by RAT the class of rational relations. A *sequential* function is a function whose graph is $R(\mathcal{T})$ for an input-deterministic transducer $\mathcal{T}$.

Two-way transducers. Given Σ, let $\Sigma_\vdash$ denote $\Sigma \uplus \{\vdash\}$, $\vdash$ is a new left-delimiter symbol. An *input-deterministic two-way transducer* (2DFT) is a tuple $\mathcal{T} = (Q, \Sigma_\vdash, \Gamma, q_0, \delta, c)$, where Q is a finite state set, Σ and Γ are finite alphabets, $q_0 \in Q$ is an initial state, $\delta\colon Q \times \Sigma \to \times \Gamma^* \times \{1, -1\} \times Q$ is a transition function, and $c\colon Q \to \mathbb{N}$ is a function that maps states to colors. A two-way transducer has a two-way read-only input tape and a one-way write-only output tape. Given an input sequence $\alpha \in \Sigma^\omega$, let $\alpha(-1) = \vdash$, the input tape holds $\vdash \alpha$. We denote a transition $\delta(p, a) = (\gamma, d, q)$ as a tuple (p, a, γ, d, q), and Δ denotes the tuple representation of δ. A *run* of $\mathcal{T}$ on $\alpha \in \Sigma^\omega$ is a sequence of transitions $(q_0, \alpha(i_0), \gamma_0, d_0, q_1)(q_1, \alpha(i_1), \gamma_1, d_1, q_2) \cdots \in \Delta^\omega$ such that $i_0 = 0$, and $i_{k+1} = i_k + d_k$ for all $k \in \mathbb{N}$. The *input* of ρ is α and the *output* of ρ is $\beta = \gamma_0\gamma_1 \cdots$. We define $c(\rho)$ as the sequence of colors $c(q_0)c(q_1)\cdots$, ρ is *accepting* if $\max \mathrm{Inf}(c(\rho))$ is even. The functional ω-relation *recognized* by the deterministic-two way transducer is defined as $R(\mathcal{T}) = \{(\alpha, \beta) \mid$ there is an accepting run of $\mathcal{T}$ with input α and output $\beta\}$.

Games. A *game arena* is a tuple $G = (V_0, V_1, v_0, A, E)$, where $V = V_0 \uplus V_1$ is a set of vertices, V_0 belongs to Eve and V_1 to Adam, v_0 is an initial vertex, A is a finite set of actions, and $E \subseteq V \times A \times V$ is a set of labeled edges such that $(v, a, v') \in E$ and $(v, a, v'') \in E$ implies that $v' = v''$ for all $v \in V$ and $a \in A$. We assume that the arena is deadlock-free. We use letters on edges as it is more convenient to have them at hand for the proofs, it is however not necessary. A *play* in G is an infinite sequence $v_0a_0v_1a_1\cdots$ such that $(v_i, a_i, v_{i+1}) \in E$ for all $i \in \mathbb{N}$. Note that a play is uniquely determined by its action sequence. A *game* is of the form $\mathcal{G} = (G, \mathit{Win})$, where G is a game arena and $\mathit{Win} \subseteq V^\omega$ is a winning condition. Eve wins a play $\alpha = v_0a_0v_1a_1\cdots$ if $v_0v_1\cdots \in \mathit{Win}$, otherwise Adam wins. For ease of presentation, we also write $\alpha \in \mathit{Win}$.

A *strategy* for Eve resp. Adam is a function $(VA)^*V_0 \to A$ resp. $(VA)^*V_1 \to A$ such that $\sigma(xv) = a$ with $x \in (VA)^*$, $v \in V$, and $a \in A$ implies that there is $v' \in V$ such that $(v, a, v') \in E$. A play $v_0a_0v_1a_1\cdots$ is consistent with a strategy σ for Eve resp. Adam if $\sigma(v_0a_0\cdots v_i) = a_i$ for all $i \in \mathbb{N}$ with $v_i \in V_0$ resp. $v_i \in V_1$. A strategy σ for Eve is a *winning strategy* if $\alpha \in \mathit{Win}$ for all plays α consistent with σ. A *strategy automaton* for Eve is a tuple $\mathcal{S} = (M, V, m_0, \delta, \mu)$, where M is a finite set of (memory) states, V is the alphabet, m_0 is an initial state, $\delta\colon M \times V \to M$ is the memory update function, and $\mu\colon M \times V_0 \to A$ is the next action function such that for all $v \in V_0$ and $m \in M$, there is $v' \in V$ with $(v, \mu(m, v), v') \in E$.

Problem statement. In this section, we introduce the problem we want to solve. Let Σ, Γ be two finite alphabets. Given a relation $R \subseteq \Sigma^\omega \times \Gamma^\omega$ and a (partial) function $f : \Sigma^\omega \to \Gamma^\omega$, f is said to *uniformize* R if $\mathrm{dom}(f) = \mathrm{dom}(R)$ and $(\alpha, f(\alpha)) \in R$ for all $\alpha \in \mathrm{dom}(R)$. We also say that R is uniformizable by f or that f is a uniformizer of R. We are interested in computable uniformizers, which we now introduce.

▶ **Definition 2** ([25] computable functions)**.** *A function $f\colon \Sigma^\omega \to \Gamma^\omega$ is called* computable *if there exists a deterministic multi-tape machine M that computes f in the following sense, M has a read-only one-way input tape, a two-way working tape, and a write-only one-way output tape. All tapes are infinite to the right, finite to the left. For any finite word $w \in \Sigma^*$, let $M(w)$ denote the output*[3] *of M on w. The function f is said to be computable if for all $\alpha \in \mathrm{dom}(f)$ and $i \in \mathbb{N}$ there exists $j \in \mathbb{N}$ such that $f(\alpha)(:i)$ is prefix of $M(\alpha(:j))$.*

Note that in the above definition, checking whether the infinite input belongs to the domain is not a requirement and should not be, because in general, it is impossible to do it reading only a finite prefix of the input. That is why in this definition, we assume that the input belongs to the domain of the function. It is a reasonable assumption. For instance, the inputs may have been produced by another program (e.g., a transducer) for which one has guarantees that they belong to some well-behaved (e.g., regular) language.

▶ **Example 3.** To begin with, consider the function $f_1\colon \{a, b, c\}^\omega \to \{b, c\}^\omega$ defined by $f_1(a^nba^\omega) = b^\omega$ and $f_1(a^nca^\omega) = c^\omega$ for all $n \in \mathbb{N}_{\geq 1}$. It is computable by a TM which on inputs of the form a^nxa^ω for $x \in \{b, c\}$, outputs nothing up to reading x, and then, depending on x, either outputs c or b whenever it reads an a in the remaining suffix a^ω.

[3] The finite word written on the output tape the first time M reaches the $|w|$th cell of the input tape

Consider the function $f_2 : \{a,b\}^\omega \to \{a,b\}^\omega$ defined by $f_2(\alpha) = a^\omega$ if α contains infinitely many a and $f(\alpha) = b^\omega$ otherwise for all $\alpha \in \{a,b\}^\omega$. It is rational but not computable, because to determine even the first output letter, an infinite lookahead is needed.

Let $\mathcal{S}$ be a class of relations. The *$\mathcal{S}$-synthesis problem* asks, given a relation $S \in \mathcal{S}$ (finitely represented), whether there exists a computable function which uniformizes S. If such a function exists, then the procedure must return a TM computing it. Our first result, proved using an easy adaptation of the proof of [7, Theorem 17], showing that it is undecidable whether a given rational relation of finite words has a sequential uniformizer, is an undecidability result.

▶ **Proposition 4.** *The* RAT*-synthesis problem is undecidable, even if restricted to the subclass of rational relations with total domain.*

Proof sketch. We sketch a reduction from Post's correspondence problem. Let $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ be a PCP instance. We construct the ω-rational relation R that contains pairs (α, β) of the form $\alpha = i_1 \cdots i_k \alpha'$ with $i_1 \cdots i_k \in \{1, \ldots, n\}^*$ and $\alpha \in \{a,b\}^\omega$ and $\beta = u_{i_1} \cdots u_{i_k} a^\omega$ if α' contains infinitely many a and $\beta \neq v_{i_1} \cdots v_{i_k} \beta'$ otherwise. If the PCP has no solution, then the function $f \colon i_1 \cdots i_k \alpha' \mapsto u_{i_1} \cdots u_{i_k} a^\omega$ uniformizes R, because $u_{i_1} \cdots u_{i_k} \neq v_{i_1} \cdots v_{i_k}$. The function f is clearly computable. If the PCP has a solution, no computable function uniformizes R. If the integer sequence $i_1 \cdots i_k$ is the solution, then $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$. Intuitively, for an input sequence starting with the solution, no prefix of the input sequence allows to determine whether the output must begin with $u_{i_1} \cdots u_{i_k}$ or is not allowed to begin with $u_{i_1} \cdots u_{i_k}$.

The relation R can be made complete by also allowing all "invalid" inputs together with any output, i.e., by adding all pairs $(\alpha, \beta) \in \{1, \ldots, n, a, b\}^\omega \times \{1, \ldots, n, a, b\}^\omega$ where the input sequence α is not of the form $i_1 \cdots i_k \alpha'$ with $i_1 \cdots i_k \in \{1, \ldots, n\}^*$ and $\alpha \in \{a,b\}^\omega$, and any output sequence β. Any Turing machine that computes f can easily be adapted to verify whether the input valid. ◀

Next, we give a semi-decision procedure for solving the RAT-synthesis problem which is sound but not complete. In Section 4, we introduce a sufficient condition for completeness which yields a (sound and complete) decision procedure for large classes of rational relations.

3 Unbounded Delay Game

In this section, given a rational relation (as a transducer), we show how to construct a finite-state ω-regular two-player game called (unbounded) delay game. We prove that if Eve wins this game then there exists a computable function which uniformizes the relation. Moreover, this function is computable by an input-deterministic two-way transducer. We analyze the complexity of solving the game, which turns out to be in ExpTime. Solving these games yields an incomplete, but sound, decision procedure for the RAT-synthesis problem.

In the game, Adam provides inputs and Eve must produce outputs such that combination of inputs and outputs is in the relation. However, as seen in Example 1, Eve might need to wait arbitrarily long before she can safely produce output. Hence, as the game is finite, it can not store arbitrary long input words, and Eve's actions cannot produce arbitrarily long words neither. Instead, we finitely abstract input and output words using a notion we call profiles. Informally, a profile of an input word stores the effects of the word (together with some output word) on the states of the transducer (that specifies the relation) as well as the maximal priority seen along the induced state transformation. Such profiles contain sufficient information to express a winning condition which makes sure that given the word of input symbols provided by Adam, if Eve would output concrete output words instead of their abstraction, she would produce infinitely often non-empty output words whose concatenation, together with the input word, belongs to the relation.

State transformation profiles. Let $R \subseteq \Sigma^\omega \times \Gamma^\omega$ be a rational relation given by a transducer $\mathcal{T} = (Q_\mathcal{T}, \Sigma, \Gamma, q_0^\mathcal{T}, \Delta_\mathcal{T}, c_\mathcal{T})$, and $C_\mathcal{T} = \text{img}(c_\mathcal{T})$ its set of used priorities. Let $\mathcal{D} = (Q_\mathcal{D}, \Sigma, q_0^\mathcal{D}, \delta_\mathcal{D}, c_\mathcal{D})$ be a deterministic parity automaton that recognizes $\text{dom}(R)$, and $C_\mathcal{D} = \text{img}(c_\mathcal{D})$ its set of used priorities; $\mathcal{D}$ can always be constructed from $\mathcal{T}$ by projecting away its outputs and by determinizing the resulting automaton.

Given $u \in \Sigma^*$, its *profile* P_u are all the possible state transformations it induces for any output. Formally, $P_u \subseteq Q_\mathcal{T} \times Q_\mathcal{T} \times C_\mathcal{T}$ is defined as $\big\{(p,q,c) \mid$ there is $v \in \Gamma^*$ and there is a run ρ of the form $\mathcal{T} \colon p \xrightarrow{u/v} q$ with $\max \text{Occ}(\rho) = c\big\}$. Profiles can be multiplied as $P_1 \otimes P_2 = \{(p,r,\max\{m,n\}) \mid \exists q \colon (p,q,m) \in P_1, (q,r,n) \in P_2\}$. Given $u_1, u_2 \in \Sigma^*$, it is easy to verify that $P_{u_1u_2} = P_{u_1} \otimes P_{u_2}$, and P_ε is neutral for $\otimes$.

Finite-state unbounded delay game. We now present a two-player ω-regular game $\mathcal{G}_\mathcal{T} = (G, \mathit{Win})$ such that if Eve has a winning strategy, then R has a computable uniformizer. In this game, Adam's actions are to provide input letters, letter-by-letter. Eve's goal is to construct a sequence of state transformations $(q_0, q_1, m_1)(q_1, q_2, m_2)\ldots$ such that if the infinite input $\alpha \in \Sigma^\omega$ provided by Adam is in $\text{dom}(R)$, then (i) the maximal priority seen infinitely often in $(m_i)_i$ is even and (ii) $\alpha = u_0u_1\cdots$ for some $u_i \in \Sigma^*$ such that $(q_i, q_{i+1}, m_{i+1}) \in P_{u_i}$ for all $i \geq 0$. As a consequence, all these finite runs can be concatenated to form an accepting run on $\alpha/v_0v_1\ldots$, entailing $(\alpha, v_0v_1\ldots) \in R$. One can then show that if Eve has a strategy to pick the state transformations while ensuring the latter property, then this strategy can be turned into a computable function, and conversely. Picking a state transformation is what we call a *producing action* for Eve. Since a state transformation picked by Eve may correspond to an arbitrarily long word u_i, she also has an action *skip* which allows her to wait before making such a producing action. Now, the difficulty for Eve is to decide when she makes producing actions, in other words, how to decompose the input α, only based on prefixes of α. To that end, before picking a state transformation, she may need to gather lookahead information from Adam. Consequently, the vertices of the game manipulates two consecutive profiles P_1 and P_2, with the invariant that P_1 is the profile of u_i while P_2 is the profile of u_{i+1}, when the input played so far by Adam is $u_0\ldots u_{i+1}$. When Eve knows enough, she picks a state transformation (q_i, q_{i+1}, m_i) in P_1, then P_1 becomes P_2 and P_2 is reset to P_ε. The inputs of Adam up to the next producing action of Eve form the word u_{i+2}, and so on. The vertices of the game also store information to decide whether the input belongs to the domain of R (states of $\mathcal{D}$), the parities m_i, as well as the states $q_0, q_1, \ldots$. Formally, the game graph $G = (V, E)$ is composed of vertices of the form $\big(q, c, P_1, P_2, r\big) \times \{\forall, \exists\}$, where

- $q \in Q_\mathcal{T}$, *State reached on the combination of input and output sequence.*
- $c \in \{-1\} \cup C_\mathcal{T}$, *Priorities of the state transformations, -1 is used to indicate that no state transformation was chosen (skip action below).*
- P_1, P_2, *Profiles obtained from the given lookahead of the input word.*
- $r \in Q_\mathcal{D}$. *State reached on the given lookahead of the input word.*

From a vertex of the form $\big(q, c, P_1, P_2, r, \forall\big)$, Adam has the following actions:

- $\xrightarrow{a} \big(q, -1, P_1, P_2 \otimes P_a, \delta_\mathcal{D}(r, a), \exists\big)$, for all $a \in \Sigma$.
  Adam provides the next lookahead letter and P_2 is updated accordingly.

From a vertex of the form $\big(q, c, P_1, P_2, r, \exists\big)$, Eve has the following actions:

- $\xrightarrow{skip} \big(q, -1, P_1, P_2, r, \forall\big)$, and
  Eve makes a non-producing action, i.e., she waits for further lookahead on the input.
- $\xrightarrow{(q,q',m)} \big(q', m, P_2, P_\varepsilon, r, \forall\big)$, where $(q, q', m) \in P_1$.
  Eve makes a producing action: a state transformation from the first lookahead profile is chosen, the state transformation is applied, and the first profile is consumed.

The initial vertex of the game is $\big(q_0^{\mathcal{T}}, -1, P_\varepsilon, P_\varepsilon, q_0^{\mathcal{D}}, \forall\big)$.

Let us now define $Win \subseteq V^\omega$. The condition makes sure that if the input sequence provided by Adam is in the domain of R, then the sequence of state transformations can be used to build on accepting run of $\mathcal{T}$ on that input. $Win \subseteq V^\omega$ is the set of all plays γ satisfying the property

$$\max \mathrm{Inf}(col_{\mathcal{D}}(\gamma)) \text{ is even} \rightarrow \max \mathrm{Inf}(col_{\mathcal{T}}(\gamma)) \text{ is even},$$

where $col_{\mathcal{D}}(\gamma) = c_{\mathcal{D}}(\pi^5(\gamma))$, $col_{\mathcal{T}}(\gamma) = \pi^2(\gamma)$, and $\pi^i(\gamma)$ is the projection of γ onto the ith component of each vertex. It is not difficult to see that Win is ω-regular, e.g., one can design a parity automaton for it.

We explain the intuition behind Win. Our goal is to extract a computable function that uniformizes the relation from a winning strategy. Intuitively, there is a computable function that uniformizes R, if every input word $\alpha \in \mathrm{dom}(R)$ can be read letter-by-letter, and from time to time, a segment of output letters is produced, continuously building an infinite output word β such that $(\alpha, \beta) \in R$. We relate this to Win. Recall that R is defined by $\mathcal{T}$, and $\mathrm{dom}(R)$ by $\mathcal{D}$. Given a play γ, there is a unique input word $\alpha \in \Sigma^\omega$ that corresponds to γ. Since we are looking to build a computable function f with $\mathrm{dom}(f) = \mathrm{dom}(R)$, we care whether $\alpha \in \mathrm{dom}(R)$. The ω-word $col_{\mathcal{D}}(\gamma)$ is equal to $c(\rho_{\mathcal{D}})$, where $\rho_{\mathcal{D}}$ is the run of $\mathcal{D}$ on α. If $\max \mathrm{Inf}(col_{\mathcal{D}}(\gamma))$ is even, $\alpha \in L(\mathcal{D})$, i.e., $\alpha \in \mathrm{dom}(R)$. An output word $\beta \in \Gamma^\infty$ that corresponds to γ is only indirectly defined, instead the play defines a (possibly finite) sequence of state transformations that an output word β should induce together with α. How to extract a concrete β from γ is formally defined in the proof of Theorem 5. The ω-word $col_{\mathcal{T}}(\gamma)$ contains the relevant information to determine whether $(\alpha, \beta) \in R(\mathcal{T})$, i.e., $(\alpha, \beta) \in R$. In particular, if β is finite, $\max \mathrm{Inf}\, col_{\mathcal{T}}(\gamma)$ is -1, that means that only finitely many producing actions have been taken. If $\max \mathrm{Inf}\, col_{\mathcal{T}}(\gamma)$ is even, we have that $(\alpha, \beta) \in R$. Thus, Win expresses that if $\alpha \in L(\mathcal{D})$, then there is some $\beta \in \Gamma^\omega$, which can be built continuously while reading α such that $(\alpha, \beta) \in R$.

From winning strategies to uniformizers. We are ready to state our first positive result: If Eve has a winning strategy in the unbounded delay game $\mathcal{G}_{\mathcal{T}}$, then $R(\mathcal{T})$ is uniformizable by a computable function. In fact, we show a more precise result, namely, that if Eve has a winning strategy, then the relation is uniformizable by a function recognized by a deterministic two-way parity transducer. Additionally, if the domain of the relation is closed[4], then a deterministic one-way transducer suffices. Just as (one-way) transducers extend parity automata with outputs on their transitions, input-deterministic two-way transducers extend deterministic two-way parity automata with outputs. The reading tape is two-way, but the output tape is one-way. The class of functions recognizable by 2DFTs is smaller than the class of computable functions and enjoys many good algorithmic properties, e.g., decidability

[4] Recall Footnote 2.

Algorithm 1 Algorithm computing continuous function f that uniformizes R. The algorithm is described in the proof sketch of Theorem 5.

```
Input: $\alpha \in \Sigma^\omega$, $G$ game arena, $\mathcal{S} = (M, V, m_0, \delta, \mu)$ strategy automaton
Output: $\beta \in \Gamma^\infty$, if $\alpha \in \mathrm{dom}(R)$, then $(\alpha, \beta) \in R$
$m := m_0$ ;                                   // current state of the strategy automaton
$u_1 := \varepsilon$ ;                                                  // first input block
$u_2 := \varepsilon$ ;                                                 // second input block
$s_{prev} := s_0$, initial vertex of $G$ ;                    // previous vertex in the game
$s_{cur} := s_0$ ;                                              // current vertex in the game
$a := \alpha(0)$ ;                                                 // current letter of $\alpha$
while $true$ do
    $u_2 := u_2.a$ ;
    $s_{cur} := s$ if $s_{cur} \xrightarrow{a} s \in E$ ;     // update game vertex according to Adam's
     action
    $m := \delta(m, s_{cur})$ ;   // strategy automaton is updated with Adam's action
    $s_{prev} := s_{cur}$ ;
    $s_{cur} := s$ if $s_{cur} \xrightarrow{\mu(m,s_{cur})} s \in E$ ;        // strategy automaton yields Eve's
     action, the updated game vertex is of the form $(\cdot, \cdot, P_{u_1}, P_{u_2}, \cdot, \cdot)$
    $m := \delta(m, s_{cur})$ ;    // strategy automaton is updated with Eve's action
    if $e := (s_{prev}, (p, q, c), s_{cur})$ is a producing edge then
        choose output block $v_1 \in \Gamma^*$ such that $\mathcal{T}\colon p \xrightarrow{u_1/v_1} q$ with max prio $c$ ;
        ;  // this choice can be made canonical by computing for instance
         the smallest word in lexicographic order satisfying this
         property
        $u_1 := u_2$ ;
        $u_2 := \varepsilon$ ;
        $print(v_1)$ ;                                            // produce output block
    end
    $a := \alpha.nextLetter()$ ;                                  // read next input letter
end
```

of the equivalence problem [3]. Note that any function recognizable by a 2DFT is computable, in the sense that it suffices to "execute" the 2DFT to get the output. So, from now on, we may freely say that a function is computable by a 2DFT.

▶ **Theorem 5.** *Let R be defined by a transducer $\mathcal{T}$. If Eve has a winning strategy in $\mathcal{G}_{\mathcal{T}}$, then R is uniformizable by a function computable by a 2DFT.*

Proof sketch. If Eve has a winning strategy in $\mathcal{G}_{\mathcal{T}}$, then she also has a finite-state winning strategy because the winning condition is ω-regular. From such a strategy we can build an algorithm (a Turing machine), see Algorithm 1, that computes a function f that uniformizes R. The high-level idea of the algorithm is to simulate the strategy, which abstracts inputs and outputs by profiles, and in parallel store concrete inputs and outputs corresponding to those profiles. This is possible as Turing machines have infinite storage capacity. In the algorithm, an input sequence α is read letter-by-letter and the corresponding play in $\mathcal{G}$ is simulated where Adam plays according to α and Eve according to her winning strategy. In the play, a lookahead $u_i \in \Sigma^*$ gained on Adam's input is stored as its profile P_{u_i}. In the algorithm, the lookahead u_i is stored concretely in addition to its profile P_{u_i}. When Eve takes an action she picks a state transformation (that should occur in the transducer) from $P_{u_{i-1}}$, the profile of the previous lookahead sequence u_{i-1}, also stored by the algorithm. The algorithm picks some $v_{i-1} \in \Gamma^*$ such that u_{i-1}/v_{i-1} induces the state transformation picked by Eve. A new non-empty lookahead $u_{i+1} \in \Sigma^*$ is built, stored as its profile $P_{u_{i+1}}$ in the play and as the concrete sequence u_{i+1} in the algorithm until Eve picks a state transformation from P_{u_i}, and so on. The lookaheads that are built are non-empty (except for the first one), and since Eve plays according to her winning strategy, the sequence of state transformations she picked can be used to build an accepting run of $\mathcal{T}$ on $(u_0u_1\dots, v_0v_1\dots)$, proving that the latter pair belongs to R.

Then we show that f can be actually recognized by a 2DFT. The main idea is to use two-wayness to encode finite lookahead over the input: the reading head goes forward to gather input information, and then must return to the initial place where the lookahead was needed to transform the input letter. The difficulty is for the 2DFT to return to the correct position, even though the lookahead can be arbitrarily long. In order to find the correct positions, we make use of a finite-state strategy automaton for Eve's winning strategy in the following sense. A (left-to-right) run of the strategy automaton on the input word yields a unique segmentation of the input, such that segments i and $i+1$ contain enough information to determine the output for segment i. The idea is to construct a 2DFT that simulates the strategy automaton in order to find the borders of the segments. If the 2DFT goes right, simulating a computation step of the deterministic strategy automaton is easy. Recovering the previous step of the strategy automaton when the 2DFT goes left is non-trivial, it is possible to compute this information using the Hopcroft-Ullman construction presented in [18]. We show that having the knowledge of the profiles of segments i and $i+1$ is enough to deterministically produce a matching output for segment i on-the-fly going from left-to-right over segment i again. ◀

We make some remarks about the form of the game, in particular the use of two lookahead profiles, instead of one. Assume we would have only one profile abstracting the lookahead over Adam inputs. For simplicity, assume the specification is automatic (i.e., letter-to-letter). Suppose, so far, Adam and Eve have alternated between providing an input letter and producing an output letter (in the finite-state game, Eve producing letter(s) corresponds to the abstract action of picking state transformations), but now, she needs to wait for more inputs before she can safely output something new. Suppose that Adam has provided some more input, say the word u, and Eve now has enough information about the input to be able to produce something new. Abstractly, it means that in the game, Adam has given the word u but only its profile P is stored. Eve might not be able to produce an output of the same length as u (for example, if producing the ith output letter depends on the $i+k$th input letter). So, she cannot consume the whole profile P (i.e., pick a state transformation in P). What she has to do, is to decompose the profile P into two profiles such that $P = P_1 \otimes P_2$ and pick a state transformation in P_1, and then continue the game with profile P_2 (and keep on updating it until she can again produce something). The problem is, firstly, that there is no unique way of decomposing P as $P_1 \otimes P_2$, and secondly, P_1 might not correspond to any prefix of u. That is why it is needed to have explicitly the decomposition at hand in the game construction.

▶ **Lemma 6.** *Deciding whether Eve has a winning strategy in $\mathcal{G}_{\mathcal{T}}$ is in* ExpTime.

Proof sketch. Two-player ω-regular games are decidable (see, e.g., [16]). The claimed upper bound is achieved by representing the winning condition as a deterministic parity automaton, carefully analyzing its size, and then solving a parity game. ◀

▶ Remark 7. The converse of Theorem 5 is not true.

Clearly, if the converse was true, the RAT-synthesis problem would be decidable, which is a contradiction to Proposition 4. We also give a small example that illustrates that uniformizability does not imply the existence of a winning strategy.

▶ **Example 8.** Consider the identity function $f\colon \{a,b\}^\omega \to \{a,b\}^\omega$ such that all inputs with either finitely many a or b are in the domain. A (badly designed) letter-to-letter transducer $\mathcal{T}$ that recognizes f has five states S, A, B, A', B', where S is the starting state, A, B (resp. A', B') are used to recognize finitely many b (resp. a), and from S, the first input/output letter non-deterministically either enters A or A'. In a play in $\mathcal{G}_\mathcal{T}$, at some point, Eve must make her first output choice, i.e., she starts to build a run of $\mathcal{T}$. This choice fixes whether the run is restricted to A, B or A', B'. No matter Eve's choice, Adam can respond with an infinite sequence of either only a (for A, B) or b (for A', B'), making it impossible to build an accepting run. Thus, Eve has no winning strategy, but clearly the function f is computable.

While in the above example, the point of failure is clearly the bad presentation of the specification, this is not the case in general. Recall the proof sketch of Proposition 4, where we provide a reduction from Post's correspondence problem. A non-deterministic transducer constructed from a given PCP instance $u_1, v_1, \ldots, u_n, v_n$ can guess whether the input word contains infinitely many a, and accordingly either checks that the output begins with $u_{i_1} \cdots u_{i_k}$ for input sequences beginning with indices $i_1 \cdots i_k$, or checks that it does not begin with a prefix equal to $v_{i_1} \cdots v_{i_k}$. As detailed in the proof sketch, if the PCP instance has no solution, there is a computable uniformization, however, using the same argumentation as in the above example, such a transducer would make it impossible to have a winning strategy. In order to have a winning strategy, the transducer must be changed such that it checks at the same time whether the output starts with $u_{i_1} \cdots u_{i_k}$ and does not start with something equal to $v_{i_1} \cdots v_{i_k}$ for input sequences beginning with $i_1 \cdots i_k$. In general, depending on the PCP instance, it is not possible to make both checks in parallel.

We state a lemma about bounded delay.

▶ **Lemma 9.** *Let R be defined by a transducer $\mathcal{T}$. If there exists $\ell \geq 0$, such that Eve has a winning strategy in $\mathcal{G}_\mathcal{T}$ with at most ℓ consecutive* skip*-moves, then R is uniformizable by a function computable by a 1DFT.*

Intuitively, such a strategy yields a function computable by a 1DFT, because the needed lookahead (as it is bounded) can be stored in the state space.

4 A Sufficient Condition for Completeness

As we have seen in the previous section,

▶ Remark 10. Theorem 5 yields a semi-decision procedure for solving the RAT-synthesis problem (it is sound but not complete).

In this section, we show that the procedure is complete for two known and expressive classes of rational relations, namely the class of automatic relations (AUT), which are for example used as specifications in Church synthesis, as well as the class of deterministic rational relations (DRAT) [24] (to be formally defined below), see Corollary 13.

To arrive at these results, we define a structural restriction on transducers that turns out to be a sufficient condition for completeness. Let $\mathcal{T}$ be a transducer. An *input* (resp. *output*) state is a state p from which there exists an outgoing transition (p, u, v, q) such that $u \neq \varepsilon$ (resp. $v \neq \varepsilon$). The set of input (resp. output) states is called Q_i (resp. Q_o). A transducer $\mathcal{T}$ has *property* $\mathcal{P}$ if for all words $u \in \Sigma^*$, all $v_1, v_2 \in \Gamma^*$ such that v_1 is a prefix of v_2 the following holds:

$$\text{if } \mathcal{T}\colon p \xrightarrow{u/v_1} q, \mathcal{T}\colon p \xrightarrow{u/v_2} r, \text{ and } q, r \text{ are input states, then } q = r.$$

This property implies that, given $\alpha \in \Sigma^\omega, \beta \in \Gamma^\omega$ such that there is a run of $\mathcal{T}$ with input α and output β, for each prefix $u \in \Sigma^*$ of α there exists a unique prefix $v \in \Gamma^*$ of β that has to be produced while reading u before the remainder α' (let $\alpha = u\alpha'$) can be read. Furthermore, the target state of $\mathcal{T}\colon q_0 \xrightarrow{u/v}$ is unique and this state is exited only when further input (from α') has to be read. In simpler terms, given a prefix of an input word, it is uniquely determined how long the prefix of a given output word has to be so that further input can be processed and the reached state is unique. This implies that input prefixes together with (long enough) output prefixes are sufficient to determine the beginning of an accepting run if such a run exists. This allows us to show the following.

▶ **Theorem 11.** *Let R be defined by a transducer $\mathcal{T}$ with property $\mathcal{P}$. If R is uniformizable by a computable function, then Eve has a winning strategy in $\mathcal{G}_\mathcal{T}$.*

Proof sketch. In fact, we explain how to construct a winning strategy from a continuous (a computable function is always continuous, see Section 5) uniformizer f of R. Given $\alpha \in \mathrm{dom}(R)$ and $f(\alpha)$, we show that it is possible to decompose the input α into $u_0u_1\cdots$ and the output $f(\alpha)$ into $v_0v_1\cdots$ such that there exists an accepting run $\mathcal{T}\colon q_0 \xrightarrow{u_0/v_0} q_1 \xrightarrow{u_1/v_1} q_2\cdots$ where each q_i is an input state for $i > 0$. Moreover, this decomposition and run can be determined in a unique way and on-the-fly, in the sense that a factor u_i/v_i only depends on the factors u_0/v_0,…, u_{i-1}/v_{i-1}. This makes it possible for Eve to pick a corresponding state transformation sequence $(q_0, q_1, c_0)(q_1, q_2, c_1)\cdots$ which is only dependent on *the so far seen* actions of Adam spelling $u_0u_1\cdots$. The main idea to determine the u_i is to look at the indices j for which the longest common prefix of the sets $S_j = \{f(\alpha(:j)\beta) \mid \alpha(:j)\beta \in \mathrm{dom}(R)\}$ strictly increases. Given u_i, the output $v_0v_1\cdots v_i$ is any common prefix of the sets S_j, such that a run $\mathcal{T}\colon q_i \xrightarrow{u_i/v_i}$ is defined and its target is an input state. The fact that $\mathcal{T}$ has property $\mathcal{P}$ guarantees that each of these runs has the same target, thus, the next state transformation (q_i, q_{i+1}, c_i) is uniquely determined. ◀

We turn to the setting of closed domains.

▶ **Lemma 12.** *Let R with* dom(R) *closed be defined by a transducer $\mathcal{T}$ with property $\mathcal{P}$. If R is uniformizable by a computable function, then there exists a computable $\ell \geq 0$ such that Eve has a winning strategy in $\mathcal{G}_\mathcal{T}$ with at most ℓ consecutive* skip*-moves.*

Proof sketch. Intuitively, the reason why bounded lookahead suffices in the setting of closed domains is that (basically at each point of time during a play) Adam's moves describe a series of longer and longer finite input words that "converge" to a valid infinite input word from the domain. Hence, Eve can not wait arbitrarily long to make producing moves, as such a play describes a valid infinite input sequence and a finite output sequence. ◀

We formally introduce AUT and DRAT. A relation is *deterministic rational* if it is recognized by a transducer where Q_i and Q_o partition its state space, and its transition relation Δ is a function $(Q_i \times \Sigma \times \{\varepsilon\} \to Q) \cup (Q_o \times \{\varepsilon\} \times \Gamma \to Q)$. It is *automatic* if additionally Δ strictly alternates between Q_i and Q_o states. It is easy to see that every DRAT-transducer (and a fortiori every AUT-transducer) satisfies the property $\mathcal{P}$. In general, given any transducer $\mathcal{T}$, we do not know if it is decidable whether $\mathcal{T}$ has property $\mathcal{P}$.

Main result. We now state our main result: Asking for the existence of a uniformization which is computable by a Turing machine or computable by an input-deterministic two-way transducer (2DFT), are equivalent questions, as long as specifications are DRAT relations. Moreover, these questions are decidable.

▶ **Corollary 13.** *Let R be defined by a* DRAT*-transducer $\mathcal{T}$. The following are equivalent:*

1. *R is uniformizable by a computable function.*
2. *R is uniformizable by a function computable by a 2DFT.*
3. *Eve has a winning strategy in $\mathcal{G}_{\mathcal{T}}$.*

Moreover, if $\mathrm{dom}(R)$ *is closed, then it is equivalent to R being uniformizable by a function computable by a 1DFT.*

Note that the above result also holds for the slightly more general case of relations given by transducers with property $\mathcal{P}$. We highlight two facts regarding closed domains.

▶ Remark 14. The set of infinite words over a finite alphabet is closed, i.e., every total domain is closed. Furthermore, it is decidable whether a domain (e.g., given by a Büchi automaton) is closed. It is a well-known fact that the topological closure of a Büchi language is a Büchi language (one can trim the automaton and declare all states to be accepting) and therefore one can check closedness by checking equivalency with its closure.

▶ **Theorem 15.** *The* AUT*- and* DRAT*-synthesis problems are* ExpTime-complete.

Proof. Membership in ExpTime directly follows from Lemma 6 and Corollary 13. In [20] it was shown that this problem is ExpTime-hard in the particular case of automatic relations with total domain, so the lower bound applies to our setting. ◀

5 Discussion

Continuous functions. We have shown that checking the existence of a computable function uniformizing a relation given by transducer with property $\mathcal{P}$ is decidable (a consequence of Theorems 5 and 11 and Lemma 6). The proofs of Theorems 5 and 11 use another notion, which is easier to manipulate mathematically than computability, that of continuity. A function is called *continuous* if

$$\forall \alpha \in \mathrm{dom}(f)\ \forall i \in \mathbb{N}\ \exists j \in \mathbb{N}\ \forall \beta \in \mathrm{dom}(f)\colon |\alpha \wedge \beta| \geq j \rightarrow |f(\alpha) \wedge f(\beta)| \geq i. \tag{1}$$

▶ **Example 16.** Consider the function f_1 of Example 3. f_1 is continuous, because the ith output symbol only depends on the $max(i, n+1)$ first input symbols. Consider the function f_2 of Example 3. The function f_2 is clearly rational, but it is not continuous. We verify that f_2 is not continuous, let α_n denote $a^n b^\omega$, we have that $|\alpha_n \wedge a^\omega| = n$ and $|f_2(\alpha_n) \wedge f_2(a^\omega)| = 0$ for all $n \in \mathbb{N}$. Thus, f_2 is not continuous.

The notions of computability and continuity are closely related. If a function $f\colon \Sigma^\omega \to \Gamma^\omega$ is computable, it is also continuous. This is not difficult to see when comparing the definitions of computable and continuous functions. The converse does not hold because the continuity definition does not have any computability requirements (see [12] for a counter-example). However, regarding synthesis, the two notions coincide:

▶ **Theorem 17.** *Let R be defined by $\mathcal{T}$ with property $\mathcal{P}$. The following are equivalent:*

1. *R is uniformizable by a continuous function.*
2. *R is uniformizable by a computable function.*

Proof. Indeed, any computable uniformizer is continuous. Theorem 11 states that if there exists a computable uniformizer, then there exists a winning strategy in the delay game. However, in the proof of this theorem, we show a stronger statement: If there exists a continuous uniformizer, then there exists a winning strategy in the delay game. Such a strategy can be assumed to have finite-memory (as finite-memory suffices to win games with ω-regular conditions). We have shown in the proof (sketch) of Theorem 5 how to translate a finite-state winning strategy into an algorithm (a Turing machine) that computes a function f which uniformizes the relation. ◀

Conclusion and future work. We investigated the synthesis of algorithms (aka. Turing machine computable functions) from rational specifications. While undecidable in general, we have proven decidability for DRAT (and a fortiori AUT). Furthermore, we have shown that the whole computation power of Turing machines is not needed, two-way transducers are sufficient (and necessary). As TMs reading heads are read-only left-to-right, converting a 2DFT into a TM requires that the TM stores longer and longer prefixes of the input in the working-tape for later access. This is the only use the TM needs to make of the working tape. This is a naive translation, and sometimes the working tape can be flushed (some prefixes of the input may possibly not be needed anymore). More generally, it is an interesting research direction to fine-tune the class of functions targeted by synthesis with respect to some constraints on the memory, including quantitative constraints.

Related to the latter research direction is the following open question: is the synthesis problem of functions computable by input-deterministic one-way (aka. sequential) transducers from deterministic rational relations decidable? It is already open for automatic relations. We have shown that if a rational relation with closed domain is uniformizable by a computable function, then also by a sequential function. However, closedness is not a sufficient condition: e.g., the function which maps any $a^n x c^\omega$ to $x c^\omega$ for $x \in \{\#, \$\}$, is sequential; a sequential transducer just has to erase the a^n part, but its domain is not closed. This problem is interesting because sequential transducers only require bounded memory to compute a function (in contrast to two-way transducers that require access to unboundedly large prefixes of the input).

References

1 Shaull Almagor and Orna Kupferman. Good-enough synthesis. In *International Conference on Computer Aided Verification*, pages 541–563. Springer, 2020.

2 Rajeev Alur, Rastislav Bodík, Eric Dallal, Dana Fisman, Pranav Garg, Garvit Juniwal, Hadas Kress-Gazit, P. Madhusudan, Milo M. K. Martin, Mukund Raghothaman, Shambwaditya Saha, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Dependable Software Systems Engineering*, pages 1–25. IEEE, 2015.

3 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *2012 27th Annual IEEE Symposium on Logic in Computer Science*, pages 65–74. IEEE, 2012.

4 Roderick Bloem, Rüdiger Ehlers, and Robert Könighofer. Cooperative reactive synthesis. In Bernd Finkbeiner, Geguang Pu, and Lijun Zhang, editors, *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings*, volume 9364 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2015.

5 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. *Acta Informatica*, 54(1):41–83, 2017. doi:10.1007/s00236-016-0273-2.

6 J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions finite-state strategies. *Trans. Ameri. Math. Soc.*, 138:295–311, 1969.

7 Arnaud Carayol and Christof Löding. Uniformization in Automata Theory. In *Proceedings of the 14th Congress of Logic, Methodology and Philosophy of Science Nancy, July 19-26, 2011*, pages 153–178. London: College Publications, 2014.

8 Krishnendu Chatterjee and Thomas A. Henzinger. Assume-guarantee synthesis. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4424 of *Lecture Notes in Computer Science*, pages 261–275. Springer, 2007.

9 Christian Choffrut and Serge Grigorieff. Uniformization of rational relations. In *Jewels are Forever*, pages 59–71. Springer, 1999.

10 Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of model checking*, volume 10. Springer, 2018.

11 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.

12 Vrunda Dave, Emmanuel Filiot, Shankara Narayanan Krishna, and Nathan Lhote. Synthesis of computable regular functions of infinite words. In *CONCUR*, volume 171 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

13 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In *ICALP*, volume 55 of *LIPIcs*, pages 125:1–125:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

14 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`.

15 Carsten Gerstacker, Felix Klein, and Bernd Finkbeiner. Bounded synthesis of reactive programs. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 441–457. Springer, 2018. `doi:10.1007/978-3-030-01090-4_26`.

16 Erich Gradel and Wolfgang Thomas. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

17 Michael Holtmann, Łukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. In *International Conference on Foundations of Software Science and Computational Structures*, pages 252–266. Springer, 2010.

18 John E Hopcroft and Jeffrey D Ullman. An approach to a unified theory of automata. *The Bell System Technical Journal*, 46(8):1793–1829, 1967.

19 F Hosch and Lawrence Landweber. Finite delay solutions for sequential conditions. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1972.

20 Felix Klein and Martin Zimmermann. How much lookahead is needed to win infinite games? *Log. Methods Comput. Sci.*, 12(3), 2016. `doi:10.2168/LMCS-12(3:4)2016`.

21 K. Kobayashi. Classification of formal languages by functional binary transductions. *Information and Control*, 15(1):95–109, July 1969.

22 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016.

23 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *ACM Symposium on Principles of Programming Languages (POPL)*. ACM, 1989.

24 Jacques Sakarovitch. *Elements of automata theory.* Cambridge University Press, 2009.

25 Klaus Weihrauch. *Computable analysis: an introduction.* Springer Science & Business Media, 2012.

Confluence of Conditional Rewriting in Logic Form

Raúl Gutiérrez
Universidad Politécnica de Madrid, Spain

Salvador Lucas
DSIC & VRAIN, Universitat Politècnica de València, Spain

Miguel Vítores
VRAIN, Universitat Politècnica de València, Spain

Abstract

We characterize conditional rewriting as satisfiability in a Herbrand-like model of terms where variables are also included as fresh constant symbols extending the original signature. Confluence of conditional rewriting and joinability of conditional critical pairs is characterized similarly. Joinability of critical pairs is then translated into combinations of *(in)feasibility* problems which can be efficiently handled by a number of automatic tools. This permits a more efficient use of standard results for proving confluence of conditional term rewriting systems, most of them relying on auxiliary proofs of joinability of conditional critical pairs, perhaps with additional syntactical and (operational) termination requirements on the system. Our approach has been implemented in a new system: CONFident. Its ability to (dis)prove confluence of conditional term rewriting systems is witnessed by means of some benchmarks comparing our tool with existing tools for similar purposes.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Logic and verification; Theory of computation → Equational logic and rewriting

Keywords and phrases Confluence, Program analysis, Rewriting-based systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.44

Supplementary Material *Software (Online Tool)*: `http://zenon.dsic.upv.es/confident/`

Funding Partially supported by grant RTI2018-094403-B-C32 funded by MCIN/AEI/10.13039 /501100011033 and by "ERDF A way of making Europe", and PROMETEO/2019/098.

1 Introduction

Confluence is a property of (abstract) reduction relations → guaranteeing that, for all abstract objects s (often called *expressions* without loss of generality) which can be reduced into two different reducts t and t', respectively (written $s \to^* t$ and $s \to^* t'$), there is another expression u to which both t and t' are reducible, i.e., both $t \to^* u$ and $t' \to^* u$ hold. A weaker property is *local* confluence, where only a *single* reduction step is allowed on s, i.e., $s \to t$ and $s \to t'$. As usual, they are defined by the commmutation of the diagrams:

$$\begin{tikzcd}
& t \arrow[dr, dotted, "*"] & \\
s \arrow[ur] \arrow[dr] & & u \\
& t' \arrow[ur, dotted, "*"'] &
\end{tikzcd}$$

Local confluence

$$\begin{tikzcd}
& t \arrow[dr, dotted, "*"] & \\
s \arrow[ur, "*"] \arrow[dr, "*"'] & & u \\
& t' \arrow[ur, dotted, "*"'] &
\end{tikzcd}$$

confluence

These two properties of abstract reduction relations are connected by the well-known *Newman's Lemma*: if → is terminating (i.e., there is no infinite reduction sequence $t_1 \to t_2 \to \cdots$), then local confluence and confluence coincide (see, e.g., [23, Lemma 2.2.5]). Now, the following issues naturally arise: (i) How to define → from the specification of a program of a

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 44; pp. 44:1–44:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(rule-based) formalism (e.g., (conditional) term rewriting [3]) or programming language? (ii) How to prove/disprove (local) confluence of such a reduction relation? (iii) How to automate the proofs? In this paper we address these problems.

Regarding (i), we use a logical approach to define reduction relations. Given a specification $\mathcal{R}$, we obtain an inference system $\mathcal{I}(\mathcal{R})$ out from the generic description of the operational semantics of the underlying formalism or language. Then, $\to$ and $\to^*$ are defined by satisfiability of atoms $s \to t$ and $s \to^* t$ in a canonical model $\mathcal{M}_{\mathcal{R}}$ which is the Herbrand model (in an "extended" Herbrand universe where variables are treated as constants) of the atoms that can be proved using $\mathcal{I}(\mathcal{R})$. This general approach applies to many computational systems and programming languages, in particular to conditional term rewriting systems (CTRS, see, e.g., [23, Chapter 7]), and Maude [5]. In Section 3, we develop this approach with a particular focus on CTRSs (to keep things simpler). Most ideas, though, can be easily generalized. Regarding (ii), we represent confluence properties above in logic form, e.g.,

$$\begin{array}{rlll} \varphi_{\text{WCR}} & (\forall x)(\forall y)(\forall z)(\exists u) & x \to y \wedge x \to z \Rightarrow y \to^* u \wedge z \to^* u & \text{Local confluence} \\ \varphi_{\text{CR}} & (\forall x)(\forall y)(\forall z)(\exists u) & x \to^* y \wedge x \to^* z \Rightarrow y \to^* u \wedge z \to^* u & \text{Confluence} \end{array}$$

In Section 4, we show that (local) confluence is characterized as *satisfiability* in $\mathcal{M}_{\mathcal{R}}$, i.e., $\mathcal{R}$ is (locally) confluent iff $\mathcal{M}_{\mathcal{R}} \models \varphi_{\text{CR}}$ (resp. $\mathcal{M}_{\mathcal{R}} \models \varphi_{\text{WCR}}$) holds. Regarding (iii), in Section 6 we show how to translate confluence problems into combinations of *(in)feasibility* problems [11]. In this setting, automated proofs are possible by using several techniques and tools developed so far, see [21] for a summary of techniques and tools in this respect. Section 7 shows how these techniques are used to prove and disprove confluence of CTRSs. We have implemented our results as part of the new tool CONFident, which can be found here:

http://zenon.dsic.upv.es/confident/

Section 8 provides some details of its implementation and use. The good results of the aforementioned techniques are witnessed by our participation in the 2021 edition of the Confluence Competition (CoCo 2021) on which we report at the end of the section. Section 9 discusses some related work. Section 10 concludes. Proofs of technical results are given in an appendix.

2 Preliminaries

Given a binary relation $\mathsf{R} \subseteq A \times A$ on a set A, we often write $a \,\mathsf{R}\, b$ instead of $(a, b) \in \mathsf{R}$. The *transitive* closure of R is denoted by R^+, and its *reflexive and transitive* closure by R^*. An element $a \in A$ is *irreducible* (or an R-*normal form*), if there exists no b such that $a \,\mathsf{R}\, b$. Given $a \in A$, if there is no infinite sequence $a = a_1 \;\mathsf{R}\; a_2 \;\mathsf{R}\; \cdots \;\mathsf{R}\; a_n \;\mathsf{R} \cdots$, then a is R-*terminating* (or *well-founded*); also, R is said *terminating* if a is R-terminating for all $a \in A$. We say that R is (locally) *confluent* if, for every $a, b, c \in A$, whenever $a \;\mathsf{R}^* b$ and $a \;\mathsf{R}^* c$ (resp. $a \;\mathsf{R}\; b$ and $a \;\mathsf{R}\; c$), there exists $d \in A$ such that $b \;\mathsf{R}^* d$ and $c \;\mathsf{R}^* d$.

We use the standard notations in term rewriting (see, e.g., [23]). In this paper, $\mathcal{X}$ denotes a countable set of *variables* and $\mathcal{F}$ denotes a *signature*, i.e., a set of *function symbols* $\{f, g, \ldots\}$ (disjoint from $\mathcal{X}$), each with a fixed *arity* given by a mapping $ar : \mathcal{F} \to \mathbb{N}$. The set of terms built from $\mathcal{F}$ and $\mathcal{X}$ is $\mathcal{T}(\mathcal{F}, \mathcal{X})$. The set of *ground* terms (i.e., terms without variable occurrences) is denoted $\mathcal{T}(\mathcal{F})$. The set of variables occurring in t is $\mathcal{V}ar(t)$. By abuse of notation, we use $\mathcal{V}ar$ also with sequences of terms or other expressions to denote the set of variables occurring in them. Terms are viewed as labeled trees in the usual way. *Positions*

$p, q, \ldots$ are represented by chains of positive natural numbers used to address subterms $t|_p$ of t. The *set of positions* of a term t is $\mathcal{P}os(t)$. A substitution is a mapping from variables into terms which is homomorphically extended to a mapping from terms to terms.

A *conditional rule* (with label α) is written $\alpha : \ell \to r \Leftarrow C$, where $\ell \in \mathcal{T}(\mathcal{F}, \mathcal{X}) - \mathcal{X}$ and $r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ are called the *left-* and *right-hand sides* of the rule, respectively, and the *conditional part* C is a sequence $s_1 \approx t_1, \cdots, s_n \approx t_n$ with $s_1, t_1, \ldots, s_n, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for some $n \geq 0$. The case $n = 0$ corresponds to an *empty* conditional part. A Conditional Term Rerwiting System (CTRS) $\mathcal{R}$ is a set of conditional rules; if all rules $\ell \to r \Leftarrow C$ in $\mathcal{R}$ have an empty conditional part and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell)$ holds, then $\mathcal{R}$ is called a Term Rewriting System (TRS).

3 Term Rewriting as Satisfiability

In term rewriting variables occurring in terms t_i in reduction sequences $t_1 \to t_2 \to \cdots \to$ are treated *as constants* in the sense that they are not instantiated in any way. This is in contrast with variables occurring in rules of TRSs which are instantiated to implement reduction steps by means of matching substitutions. In the following we provide a formal presentation of this fact which permits the definition of a canonical model $\mathcal{M}_{\mathcal{R}}$ which captures the reduction of terms with variables, in contrast to the usual (ground) models developed elsewhere (e.g., [6]) which are better suited to capture *ground rewriting*, i.e., rewriting of *ground* terms.

▶ Remark 1 (Confluence and ground confluence). In general, confluence and *ground* confluence (i.e., confluence of $\to$ when restricted to ground terms) of (C)TRSs do *not* coincide. For instance, the TRS $\mathcal{R} = \{f(x) \to a, f(x) \to x\}$ over the signature $\mathcal{F} = \{a, f\}$ is ground confluent, but not confluent. If a new constant b is added to $\mathcal{F}$, then $\mathcal{R}$ is not ground confluent anymore.

In Section 4 we use $\mathcal{M}_{\mathcal{R}}$ to provide a characterization of confluence properties as satisfiability in $\mathcal{M}_{\mathcal{R}}$. In the following, as anticipated by the expression of (local) confluence using first-order formulas φ_{CR} and φ_{WCR}, we view term rewriting from a logical point of view. A first-order language with function symbols $f, g, \ldots$ from a signature $\mathcal{F}$ and predicate symbols $P, Q, \ldots$ from a signature Π is considered where atoms and formulas are built in the usual way. The pair $\mathcal{F}, \Pi$ is often called a *signature with predicates* [9]. In particular, rewriting expressions $s \to t$ (one-step reduction), $s \to^* t$ (zero or many-step reduction), $s \downarrow t$ (joinability), etc., are viewed as *atoms* with (binary) predicate symbols $\to$, $\to^*$, $\downarrow$, etc.

3.1 Operational Semantics of Conditional Rewriting in Logic Form

Conditions $s \approx t$ in conditional rules admit several *semantics*, i.e., forms to evaluate them see, e.g., [23, Definition 7.1.3]. *Oriented* CTRSs are those whose conditions $s \approx t$ are handled as *reachability* tests. *Join* CTRSs use *joinability* tests instead. *Semiequational* CTRSs use *convertibility* tests. For oriented CTRSs $\mathcal{R}$, an inference system $\mathcal{I}_O(\mathcal{R})$ is obtained from the following generic inference system $\mathfrak{I}_{\text{O-CTRS}}$:

$$\text{(Rf)} \quad \frac{}{x \to^* x} \qquad\qquad \text{(C)}_{f,i} \quad \frac{x_i \to y_i}{f(x_1, \ldots, x_i, \ldots, x_k) \to f(x_1, \ldots, y_i, \ldots, x_k)}$$

$$\text{for all } f \in \mathcal{F} \text{ and } 1 \leq i \leq k$$

$$\text{(T)} \quad \frac{x \to y \qquad y \to^* z}{x \to^* z} \qquad\qquad \text{(Rl)}_\alpha \quad \frac{s_1 \to^* t_1 \quad \cdots \quad s_n \to^* t_n}{\ell \to r}$$

$$\text{for } \alpha : \ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n \in \mathcal{R}$$

$$\begin{align}
(\forall x)\ x \to^* x \tag{4}\\
(\forall x,y,z)\ x \to y \wedge y \to^* z \Rightarrow x \to^* z \tag{5}\\
(\forall x,y,z)\ x \to y \Rightarrow f(x,z) \to f(y,z) \tag{6}\\
(\forall x,y,z)\ x \to y \Rightarrow f(z,x) \to f(z,y) \tag{7}\\
a \to b \tag{8}\\
(\forall x)\ f(x,a) \to^* f(b,b) \Rightarrow f(c,x) \to x \tag{9}\\
(\forall y)\ (y,y) \to b \tag{10}
\end{align}$$

Figure 1 Theory $\overline{\mathcal{R}}_O$ for the oriented semantics of $\mathcal{R}$ in Example 3.

by *specializing* $(\mathrm{C})_{f,i}$ for each k-ary symbol f in the signature $\mathcal{F}$ and $1 \leq i \leq k$ and $(\mathrm{Rl})_\alpha$ for all conditional rules $\alpha : \ell \to r \Leftarrow C$ in $\mathcal{R}$ [14, Section 4.5]. Inference rules in $\mathcal{I}_O(\mathcal{R})$ are *schematic*: each inference rule $\frac{B_1 \ \cdots \ B_n}{A}$ in $\mathcal{I}_O(\mathcal{R})$ can be used under any *instance* $\frac{\sigma(B_1) \ \cdots \ \sigma(B_n)}{\sigma(A)}$ of the rule by a substitution σ. For *join* CTRSs, we replace rule $(\mathrm{Rl})_\alpha$ by

$$(\mathrm{Rl})^J_\alpha \quad \frac{s_1 \to^* z_1 \quad t_1 \to^* z_1 \quad \cdots \quad s_n \to^* z_n \quad t_n \to^* z_n}{\ell \to r}$$

where $z_1, \ldots, z_n$ do not occur in ℓ, r, s_i, t_i for $1 \leq i \leq n$. In this way, we obtain $\mathfrak{I}_{\text{J-CTRS}}$ and $\mathcal{I}_J(\mathcal{R})$ from $\mathfrak{I}_{\text{J-CTRS}}$ as before. Note that the joinability predicate $\downarrow$ is not necessary.

▶ Remark 2 (Semi-equational CTRSs). For semi-equational CTRSs we would proceed similarly, defining a new rule $(\mathrm{Rl})^{SE}_\alpha$ borrowing $(\mathrm{Rl})_\alpha$ where $\leftrightarrow^*$ is used instead of $\to^*$, and adding more inference rules to deal with $\leftrightarrow^*$: first $\frac{x \to y}{x \leftrightarrow y}$, also $\frac{y \to x}{x \leftrightarrow y}$, and then $\frac{x \leftrightarrow y \qquad y \leftrightarrow^* z}{x \leftrightarrow^* z}$.

We obtain a theory $\overline{\mathcal{R}}_O$ (resp. $\overline{\mathcal{R}}_J$, etc.) from $\mathcal{I}_O(\mathcal{R})$ (resp. $\mathcal{I}_J(\mathcal{R})$, etc.) as follows [14, Section 4.5]: the inference rules $(\rho)\frac{B_1 \ \cdots \ B_n}{A}$ in $\mathcal{I}(\mathcal{R})$ are considered as *sentences* $\overline{\rho}$ of the form $(\forall \vec{x})\, B_1 \wedge \cdots \wedge B_n \Rightarrow A$, where $\vec{x}$ is the sequence of variables occurring in atoms $B_1, \ldots, B_n$ and A; if empty, we just write $B_1 \wedge \cdots \wedge B_n \Rightarrow A$.

▶ **Example 3.** Consider the CTRS $\mathcal{R}$

$$\begin{align}
a &\to b \tag{1}\\
f(c,x) &\to x \Leftarrow f(x,a) \approx f(b,b) \tag{2}\\
f(y,y) &\to b \tag{3}
\end{align}$$

The theory $\overline{\mathcal{R}}_O$ can be found in Figure 1. Note that this gives $\mathcal{R}$ the computational semantics of an *oriented* CTRS. Also, $\overline{\mathcal{R}}_J = \{(4), (5), (6), (7), (8), (10), (11)\}$ for

$$(\forall x)(\forall z)\ f(x,a) \to^* z \wedge f(b,b) \to^* z \Rightarrow f(c,x) \to x \tag{11}$$

i.e., we use (11) instead of (9). We usually just write $\overline{\mathcal{R}}$ to denote the (appropriate) theory associated to a (join, oriented,...) CTRS. In the following, given a first-order theory Th and a formula φ, $\mathsf{Th} \vdash \varphi$ means that φ is *deducible* from (or a *logical consequence* of) Th.

For all terms s, t, we write (i) $s \to_{\mathcal{R}} t$ (resp. $s \to^*_{\mathcal{R}} t$) iff there is a (well-formed)[1] proof tree for $s \to t$ (resp. $s \to^* t$) using $\mathcal{I}(\mathcal{R})$. Equivalently, we have (ii) $s \to_{\mathcal{R}} t$ (resp. $s \to^*_{\mathcal{R}} t$) iff $\overline{\mathcal{R}} \vdash s \to t$ (resp. $\overline{\mathcal{R}} \vdash s \to^* t$) holds. The first presentation (i) is well-suited for the analysis of the termination behavior of CTRSs: we say that $\mathcal{R}$ is *operationally terminating* if there is no (well-formed) infinite proof trees for goals $s \to t$ and $s \to^* t$ in $\mathcal{I}(\mathcal{R})$ [16]. However, the proof theoretic presentation (ii) is more important in the analysis of (in)feasibility of rewriting goals in Section 4. It also suffices to

[1] By a *well-formed* proof tree we mean a proof tree where proof conditions introduced by inference rules are developed from left to right, see [16].

define *termination* of CTRSs: a CTRS $\mathcal{R}$ is terminating if $\to_{\mathcal{R}}$ is terminating. Termination and operational termination of CTRSs differ, see [17, Section 3] for a deeper discussion about differences and connections between both notions.

We use termination and operational termination in some confluence results for CTRSs (Section 7). The tool MU-TERM [12] can be used for automatically proving and disproving termination and operational termination of CTRSs.[2]

▶ **Definition 4** (Joinable terms). *Given a CTRS $\mathcal{R}$ and terms s,t, we write $s \downarrow_{\mathcal{R}} t$ if and only if there is a term u such that $s \to^*_{\mathcal{R}} u$ and $t \to^*_{\mathcal{R}} u$. We often say that s and t are* joinable.

3.2 Dealing With Variables in Terms as (Fresh) Constants

Let $\mathcal{F}$ be a signature and $\mathcal{X}$ be a set of variables such that $\mathcal{F} \cap \mathcal{X} = \emptyset$. We let $\mathcal{F}_{\mathcal{X}} = \mathcal{F} \cup C_{\mathcal{X}}$ where variables $x \in \mathcal{X}$ are considered (different) *constant* symbols c_x of $C_{\mathcal{X}} = \{c_x \mid x \in \mathcal{X}\}$ and $\mathcal{F}$ and $C_{\mathcal{X}}$ are disjoint. Note that the set $\mathcal{T}(\mathcal{F},\mathcal{X})$ of terms with variables for the signature $\mathcal{F}$ is isomorphic to the set $\mathcal{T}(\mathcal{F}_{\mathcal{X}})$ of ground terms for $\mathcal{F}_{\mathcal{X}}$: given a term $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $t^{\downarrow} \in \mathcal{T}(\mathcal{F}_{\mathcal{X}})$ is obtained by replacing each occurrence of $x \in \mathcal{X}$ in t by c_x.[3] Vice versa: given $t \in \mathcal{T}(\mathcal{F}_{\mathcal{X}})$, $t^{\uparrow} \in \mathcal{T}(\mathcal{F},\mathcal{X})$ is obtained by replacing, for all $x \in \mathcal{X}$, each constant c_x in t by x.

▶ **Proposition 5.** *For all terms $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $(t^{\downarrow})^{\uparrow} = t$. For all terms $t \in \mathcal{T}(\mathcal{F}_{\mathcal{X}})$, $(t^{\uparrow})^{\downarrow} = t$.*

Also, given a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F},\mathcal{X})$, define $\sigma^{\downarrow} : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{\mathcal{X}})$ to be $\sigma^{\downarrow}(x) = \sigma(x)^{\downarrow}$ for all $x \in \mathcal{X}$ (given $\theta : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{\mathcal{X}})$, define $\theta^{\uparrow} : \mathcal{X} \to \mathcal{T}(\mathcal{F},\mathcal{X})$ similarly). The following result shows that rewriting with terms in $\mathcal{T}(\mathcal{F},\mathcal{X})$ can be simulated as ground rewriting in $\mathcal{T}(\mathcal{F}_{\mathcal{X}})$.

▶ **Proposition 6.** *Let $\mathcal{R} = (\mathcal{F}, R)$ be a CTRS and $s,t \in \mathcal{T}(\mathcal{F},\mathcal{X})$. Then, $s \to_{\mathcal{R}} t$ if and only if $s^{\downarrow} \to_{\mathcal{R}} t^{\downarrow}$ and $s \to^*_{\mathcal{R}} t$ if and only if $s^{\downarrow} \to^*_{\mathcal{R}} t^{\downarrow}$.*

In the following, given a condition C, i.e., $s_1 \approx t_1, \ldots, s_n \approx t_n$, we write $C^{\downarrow}$ to denote $s_1^{\downarrow} \approx t_1^{\downarrow}, \ldots, s_n^{\downarrow} \approx t_n^{\downarrow}$.

3.3 A Ground Model for Rewriting Terms with Variables

Given a signature with predicates $\mathcal{F}, \Pi$, an $\mathcal{F}, \Pi$-*structure* $\mathcal{A}$ (or just *structure* if $\mathcal{F}, \Pi$ is clear from the context) consists of a *domain* (also denoted) $\mathcal{A}$ together with an interpretation of the function symbols $f \in \mathcal{F}$ and predicate symbols $P \in \Pi$ as mappings $f^{\mathcal{A}}$ and relations $P^{\mathcal{A}}$ on $\mathcal{A}$, respectively. Then, the usual interpretation of first-order formulas with respect to $\mathcal{A}$ is considered [20, page 60]. An $\mathcal{F}, \Pi$-model for a theory Th, i.e., a set of first-order sentences (formulas whose variables are all *quantified*), is just a structure $\mathcal{A}$ that makes them all true, written $\mathcal{A} \models \mathsf{Th}$. A formula φ is a *logical consequence* of a theory Th (written $\mathsf{Th} \models \varphi$) iff every model $\mathcal{A}$ of Th is also a model of φ. The *canonical model* $\mathcal{M}_{\mathcal{R}}$ of a CTRS $\mathcal{R}$ is defined as follows.

▶ **Definition 7** (Canonical model for conditional rewriting). *Let $\mathcal{R}$ be a CTRS. The* canonical model *$\mathcal{M}_{\mathcal{R}}$ of $\mathcal{R}$ has domain $\mathcal{T}(\mathcal{F}_{\mathcal{X}})$; each k-ary symbol $f \in \mathcal{F}$ is interpreted as $f^{\mathcal{M}_{\mathcal{R}}}(t_1,\ldots,t_k) = f(t_1,\ldots,t_k)$ for all $t_1,\ldots,t_k \in \mathcal{T}(\mathcal{F}_{\mathcal{X}})$. Finally, predicate symbols $\to$ and $\to^*$ are interpreted as follows:*

$$\to^{\mathcal{M}_{\mathcal{R}}} = \{(s^{\downarrow}, t^{\downarrow}) \mid s,t \in \mathcal{T}(\mathcal{F},\mathcal{X}) \wedge s \to_{\mathcal{R}} t\} \qquad (\to^*)^{\mathcal{M}_{\mathcal{R}}} = \{(s^{\downarrow}, t^{\downarrow}) \mid s,t \in \mathcal{T}(\mathcal{F},\mathcal{X}) \wedge s \to^*_{\mathcal{R}} t\}$$

[2] Although the version of MU-TERM described in [12] did not allow proofs of termination of CTRSs, for the purpose of serving as a backbone for CONFident, we recently modified MU-TERM as to provide explicit use of the techniques described in [18], which can be used to prove and disprove termination of CTRSs. Thus, MU-TERM users can prove and disprove termination of CTRSs by following the instructions in `http://zenon.dsic.upv.es/muterm/?name=documentation#CTRSs`.

[3] We use $\downarrow$ as superindex denoting this *grounding* operation as in $t^{\downarrow}$, hopefully not leading to confusion with the infix use of $\downarrow$ as joinability operator, as in $s \downarrow t$.

Definition 7 generalizes to accomodate interpretations for $\leftrightarrow$ and $\leftrightarrow^*$ in semi-equational CTRSs in the obvious way.[4]

▶ **Theorem 8.** *For all CTRSs $\mathcal{R}$, $\mathcal{M}_{\mathcal{R}} \models \overline{\mathcal{R}}$.*

We have the following:

▶ **Proposition 9.** *Let $\mathcal{R} = (\mathcal{F}, R)$ be a CTRS, $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and $\vec{x} = x_1, \ldots, x_n$ denote the variables occurring in s and t, i.e., $\mathcal{V}ar(s) \cup \mathcal{V}ar(t) = \{x_1, \ldots, x_n\}$. Then,*

1. *We have that $\sigma(s) \to^*_{\mathcal{R}} \sigma(t)$ for all substitutions $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$, if and only if $(s^{\downarrow}, t^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$.*
2. *$\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x})\, s \to^* t$ if and only if $(s^{\downarrow}, t^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$.*

Proposition 9 shows that we can *remove* universal quantifiers from reachability formulas if variables x in the involved terms are replaced by the corresponding constants c_x.

4 Confluence of Rewriting as a Satisfiability Problem

In view of Section 3.1, it is perhaps natural to adopt a proof theoretical definition of (local) confluence of CTRSs as follows: *a CTRS is (locally) confluent if and only if $\overline{\mathcal{R}} \vdash \varphi_{CR}$ (resp. $\overline{\mathcal{R}} \vdash \varphi_{WCR}$) holds.* The following example (using a TRS) shows that this is *not* equivalent to the usual definition.

▶ **Example 10.** A well-known example of a *locally confluent* but *nonconfluent* TRS is $\mathcal{R} = \{\mathsf{b} \to \mathsf{a}, \mathsf{b} \to \mathsf{c}, \mathsf{c} \to \mathsf{b}, \mathsf{c} \to \mathsf{d}\}$. The theory $\overline{\mathcal{R}}$ for $\mathcal{R}$ is

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
(\forall x)\, x & \to^* & x & \mathsf{b} & \to & \mathsf{a} & \mathsf{c} & \to & \mathsf{b} \\
(\forall x, y, z)\, x \to y \wedge y \to^* z \Rightarrow x & \to^* & z & \mathsf{b} & \to & \mathsf{c} & \mathsf{c} & \to & \mathsf{d}
\end{array}$$

Unfortunately, φ_{WCR} is *not* a logical consequence of $\overline{\mathcal{R}}$ (i.e., $\overline{\mathcal{R}} \models \varphi_{\mathrm{WCR}}$ does *not* hold) and hence[5] it *cannot* be proved from $\overline{\mathcal{R}}$ (i.e., $\overline{\mathcal{R}} \vdash \varphi_{\mathrm{WCR}}$ does not hold): there is a model $\mathcal{A}$ of $\overline{\mathcal{R}}$ which is *not* a model of φ_{WCR}. The interpretation domain is $\mathcal{A} = \{0, 1, 2, 3, 4\}$, function symbols are interpreted by: $\mathsf{a}^{\mathcal{A}} = 0$, $\mathsf{b}^{\mathcal{A}} = 1$, $\mathsf{c}^{\mathcal{A}} = 2$, $\mathsf{d}^{\mathcal{A}} = 3$, and predicate symbols by

$$\begin{aligned}
\to^{\mathcal{A}} &= \{(1,0), (1,2), (2,1), (2,3), (4,0), (4,3)\} \\
(\to^*)^{\mathcal{A}} &= \{(1,0), (1,2), (2,1), (2,3), (4,0), (4,3)\} \cup \{(0,0), (1,1), (2,2), (3,3), (4,4)\} \cup \{(2,0), (1,3)\}
\end{aligned}$$

Although $\mathcal{A} \models \overline{\mathcal{R}}$ holds, with the valuation α given by $\alpha(x) = 4$, $\alpha(y) = 0$ and $\alpha(z) = 3$, $[x \to y \wedge x \to z]^{\mathcal{A}}_{\alpha}$ holds true, but $[y \to^* u \wedge z \to^* u]^{\mathcal{A}}_{\alpha}$ is false for all valuations of u. Thus, $\mathcal{A} \models \varphi_{WCR}$ does *not* hold. Hence $\overline{\mathcal{R}} \models \varphi_{WCR}$ does *not* hold either.

Instead, we use $\mathcal{M}_{\mathcal{R}}$ to define (local) confluence as *satisfiability* in $\mathcal{M}_{\mathcal{R}}$.

▶ **Theorem 11** (Confluence of CTRSs as satisfiability in $\mathcal{M}_{\mathcal{R}}$)**.** *A CTRS $\mathcal{R}$ is (locally) confluent if and only if $\mathcal{M}_{\mathcal{R}} \models \varphi_{CR}$ (resp. $\mathcal{M}_{\mathcal{R}} \models \varphi_{WCR}$) holds.*

Now, as a consequence of [14, Corollary 14] and Theorem 11, we have the following:

▶ **Corollary 12.** *Let $\mathcal{R}$ be a CTRS. If $\mathcal{M}_{\mathcal{R}} \vdash \varphi_{CR}$ (resp. $\mathcal{M}_{\mathcal{R}} \vdash \varphi_{WCR}$) holds, then $\mathcal{R}$ is (locally) confluent.*

Example 10 shows that the statement in Corollary 12 cannot be reversed.

[4] The theory $\overline{\mathcal{R}}_J$ associated to a join CTRS $\mathcal{R}$ uses predicates $\to$ and $\to^*$ only. Hence, no change in the definition of $\mathcal{M}_{\mathcal{R}}$ is necessary. According to Remark 2, though, for semi-equational CTRSs additional predicate symbols $\leftrightarrow$ and $\leftrightarrow^*$ are necessary. We just need to enrich $\mathcal{M}_{\mathcal{R}}$ with the corresponding interpretations for those new predicate symbols.

[5] By Gödel's completeness theorem, see, e.g., [20, Corollary 2.19], deducibility and logical consequence are equivalent, i.e., $\mathsf{Th} \vdash \varphi$ iff $\mathsf{Th} \models \varphi$.

5 Proofs of confluence using critical pairs

In proofs of confluence, joinability of critical pairs plays a main role. A *conditional critical pair* (CCP) is an expression $\langle s,t\rangle \Leftarrow C$ where $\langle s,t\rangle$ is the *peak* of the CCP, for terms s and t, and C is the *conditional part*, i.e., a sequence $s_1 \approx t_1,\ldots,s_n \approx t_n$ of conditions. They are obtained from CTRSs as follows, see, e.g., [7, Definition 3] and also [23, Definition 7.1.8(1)].

▶ **Definition 13** (Conditional critical pair). *Let $\mathcal{R}$ be a CTRS. Let $\alpha : \ell \to r \Leftarrow C$ and $\alpha' : \ell' \to r' \Leftarrow C'$ be rules of $\mathcal{R}$ sharing no variable (rename if necessary). Let $p \in \mathcal{P}os_{\mathcal{F}}(\ell)$ be a nonvariable position of ℓ such that $\ell|_p$ and ℓ' unify with* mgu *σ. Then, we call the expression $\langle \sigma(\ell[r']_p), \sigma(r')\rangle \Leftarrow \sigma(C), \sigma(C')$ a* conditional critical pair *(CCP) of $\mathcal{R}$. If α and α' are (possibly renamed versions of) the same rule, the case $p = \Lambda$ is not considered to obtain a CCP.*

CCPs $\langle s,t\rangle \Leftarrow C$ whose conditional part C is empty are called *critical pairs* and simply written $\langle s,t\rangle$ as in the usual notation and definition, see, e.g., [23, Definition 4.2.1]. TRSs have (unconditional) critical pairs only; the set of critical pairs of a TRS $\mathcal{R}$ is denoted $\mathsf{CP}(\mathcal{R})$. In the following, $\mathsf{CCP}(\mathcal{R})$ denotes the set of CCPs of a CTRS $\mathcal{R}$. Note that $\mathsf{CP}(\mathcal{R}) \subseteq \mathsf{CCP}(\mathcal{R})$, as ordinary, unconditional critical pairs are particular CCPs with an empty conditional part. Although conditions $s_i \approx t_i$ admit multiple interpretations (as joinability, reachability, etc.), joinability of a critical pair is homogeneously defined as follows [23, Definition 7.1.8(2)]:

▶ **Definition 14** (Joinable conditional critical pair). *Let $\mathcal{R}$ be a CTRS and $\pi : \langle s,t\rangle \Leftarrow C$ be a critical pair. We say that π is* joinable *if $\sigma(s) \downarrow_{\mathcal{R}} \sigma(t)$ holds for all substitutions σ such that $\sigma(C)$ holds. Otherwise, π is not joinable.*

An important aspect in the analysis of confluence is checking (conditional) critical pairs for (non)joinability. The following result provides a logical characterization of joinability of the CCPs of a CTRS $\mathcal{R}$ as satisfiability in $\mathcal{M}_{\mathcal{R}}$.

▶ **Proposition 15.** *Let $\mathcal{R}$ be a CTRS. A CCP $\pi : \langle s,t\rangle \Leftarrow C$ is joinable if and only if $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x})(\exists z)\, C \Rightarrow s \to^* z \wedge t \to^* z$ holds, where $\vec{x} = x_1,\ldots,x_m$ are the variables occurring in C,s,t and $z \notin \mathcal{V}ar(C,s,t)$.*

The following sections investigate how to prove and disprove joinability of conditional critical pairs by (dis)proving appropriate feasibility problems using existing tools like infChecker[6] to automatically prove and disprove such feasibility problems [11].

6 Joinability of Terms and Feasibility Problems

Given a set $\mathbb{P}$ of (binary) predicates, let $\mathbb{T} = \{\mathsf{Th}_{\bowtie} \mid \bowtie \in \mathbb{P}\}$ be a $\mathbb{P}$-indexed set of first-order theories $\mathsf{Th}_{\bowtie}$ defining predicates $\bowtie$. An *f-condition* is an atom $s \bowtie t$ where $\bowtie \in \mathbb{P}$ and $s,t \in \mathcal{T}(\mathcal{F},\mathcal{X})$. Sequences $\mathsf{F} = (\gamma_i)_{i=1}^n = (\gamma_1,\ldots,\gamma_n)$ of f-conditions are called *f-sequences.* We often drop "f-" when no confusion arises. Empty sequences are written ().

▶ **Definition 16** (Feasibility). *A condition $s \bowtie t$ is $(\mathbb{T},\sigma)$*-feasible *if $\mathsf{Th}_{\bowtie} \vdash \sigma(s) \bowtie \sigma(t)$ holds; otherwise, it is $(\mathbb{T},\sigma)$*-infeasible*. We also say that $s \bowtie t$ is $\mathbb{T}$*-feasible *(or $\mathsf{Th}_{\bowtie}$-feasible, or just feasible if no confusion arises) if it is $(\mathbb{T},\sigma)$-feasible for some substitution σ; otherwise, we call it* infeasible*.*

A sequence F is $\mathbb{T}$-feasible (or just feasible*) iff there is a substitution σ such that, for all $\gamma \in \mathsf{F}$, γ is $(\mathbb{T},\sigma)$-feasible. Note that () is trivially feasible.*

In the following, $\mathsf{Th}_{\bowtie} = \overline{\mathcal{R}}$ for all $\bowtie \in \{\to,\to^*,\downarrow,\ldots\}$.

[6] http://zenon.dsic.upv.es/infChecker/

6.1 Proving Conditional Joinability

Proposition 15 characterizes joinability of the CCPs of a CTRS $\mathcal{R}$ as the satisfiability of a logical sentence in $\mathcal{M}_{\mathcal{R}}$. In the following, we show how to advantageously use the results in [14, 11] to prove and disprove joinability of CCPs. The following consequence of Proposition 15 and [14, Corollary 14] provides a sufficient condition for joinability of CCPs.

▶ **Corollary 17.** *Let $\mathcal{R}$ be a CTRS and $\pi : \langle s,t\rangle \Leftarrow C$ be a critical pair. If $\overline{\mathcal{R}} \vdash (\forall\vec{x})(\exists z)\, C \Rightarrow s \to^* z \wedge t \to^* z$ holds, then π is joinable.*

This result can be used together with theorem provers like Prover9 [19] for a practical use in proofs of joinability of critical pairs. The following result is a consequence of Proposition 15.

▶ **Corollary 18.** *Let $\mathcal{R}$ be a CTRS and $\pi : \langle s,t\rangle \Leftarrow C$ be a critical pair. If $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is $\overline{\mathcal{R}}$-feasible, then π is joinable.*

▶ **Example 19.** Consider the following variant $\mathcal{R}$ of the CTRS in Example 3:

$$a \to b \tag{12}$$

$$f(c,x) \to a \Leftarrow f(x,a) \approx f(b,b) \tag{13}$$

$$f(y,y) \to b \tag{14}$$

Note that rule (13) is feasible, both under join and oriented semantics: $f(\underline{a},a) \to f(b,\underline{a}) \to f(b,b)$ (which implies $f(a,a) \downarrow f(b,b)$). The only CCP is $\pi : \langle a,b\rangle \Leftarrow f(c,a) \approx f(b,b)$. Since $a \to^* z, b \to^* z$ is $\overline{\mathcal{R}}$-feasible (both for the join and oriented semantics of $\mathcal{R}$), by Corollary 18, π is joinable.

6.2 Disproving Conditional Joinability

Regarding proofs of *non-joinability*, we show how to formulate it as an feasibility problem.

▶ **Proposition 20.** *Let $\mathcal{R}$ be a CTRS and $\pi : \langle s,t\rangle \Leftarrow C$ be a critical pair such that $C^{\downarrow}$ is $\overline{\mathcal{R}}$-feasible. If $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is $\overline{\mathcal{R}}$-infeasible, then π is not joinable.*

▶ **Example 21.** Consider the following CTRS $\mathcal{R}$

$$f(x,x) \to x \Leftarrow f(x,x) \approx b \tag{15}$$

$$f(y,y) \to b \tag{16}$$

There is only one critical pair $\pi : \langle x,b\rangle \Leftarrow f(x,x) \approx b$. Note that $f(c_x,c_x) \approx b$ is $\overline{\mathcal{R}}$-feasible due to the unconditional rule (this can be proved with infChecker). Non-joinability of π can be proved as the $\mathcal{R}$-infeasibility of

$$c_x \to^* z, b \to^* z \tag{17}$$

using infChecker. By Proposition 20, π is not joinable.

The following result characterizes joinability of CCPs $\langle s,t\rangle \Leftarrow C$ where the conditional part C and the peak $\langle s,t\rangle$ share no variable.

▶ **Proposition 22.** *Let $\mathcal{R}$ be a CTRS and $\pi : \langle s,t\rangle \Leftarrow C$ be a critical pair such that $\mathcal{V}ar(s,t) \cap \mathcal{V}ar(C) = \emptyset$. Then, π is joinable if and only if C is $\overline{\mathcal{R}}$-infeasible or $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is $\overline{\mathcal{R}}$-feasible.*

▶ **Example 23.** Consider the CTRS $\mathcal{R}$ in Example 3. As in Example 19 for (13), rule (2) is feasible, both under join and oriented semantics. There is a single critical pair $\pi : \langle c,b\rangle \Leftarrow f(c,a) \approx f(b,b)$.

- As a *join* CTRS, $\overline{\mathcal{R}}$-feasibility of $f(c,a) \downarrow f(b,b)$ together with $\overline{\mathcal{R}}$-infeasiblity of $c \to^* z, b \to^* z$ can both be proved with infChecker. Thus, π is not joinable.
- As an *oriented* CTRS, $\overline{\mathcal{R}}$-infeasibility of $f(c,a) \to^* f(b,b)$ can be proved with infChecker. Thus, π is joinable.

For TRSs, whose critical pairs have no conditional part, we have the following characterization of joinability as a consequence of Proposition 22.

▶ **Corollary 24.** *Let $\mathcal{R}$ be a (C)TRS. A critical pair $\langle s, t\rangle$ is joinable if and only if $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is $\overline{\mathcal{R}}$-feasible.*

Actually, Corollary 24 characterizes *joinability of terms s and t* (being part of a critical pair or not).

▶ **Example 25.** Consider the following TRS from COPS (`http://cops.uibk.ac.at/?q=999`)

$$a(b(x)) \to b(c(x)) \tag{18}$$
$$c(b(x)) \to b(c(x)) \tag{19}$$
$$c(b(x)) \to c(c(x)) \tag{20}$$
$$b(b(x)) \to a(c(x)) \tag{21}$$
$$a(b(x)) \to a(b(x)) \tag{22}$$
$$c(c(x)) \to c(b(x)) \tag{23}$$
$$a(c(x)) \to c(a(x)) \tag{24}$$

By Corollary 24, joinability of the critical pair $\langle a(a(c(x))), b(c(b(x)))\rangle$ can be *disproved* as the infeasibility of $a(a(c(c_x))) \to^* z, b(c(b(c_x))) \to^* z$, which is proved by infChecker.

7 Confluence of CTRSs

In the analysis of confluence of CTRSs, a crucial notion is that of *conditional* critical pairs associated to a CTRS $\mathcal{R}$. We have the following (well-known) fact.

▶ **Proposition 26.** *Let $\mathcal{R}$ be a CTRS. If $\mathsf{CCP}(\mathcal{R})$ contains a non-joinable CCP, then $\mathcal{R}$ is not (locally) confluent.*

▶ **Example 27.** For the TRS $\mathcal{R}$ in Example 25, since $\mathsf{CP}(\mathcal{R})$ contains a nonjoinable critical pair $\langle a(a(c(x))), b(c(b(x)))\rangle$, by Proposition 26 we conclude that $\mathcal{R}$ is not confluent.

▶ **Example 28.** As a consequence of Proposition 26, $\mathcal{R}$ in Example 3, when considered as a join CTRS, is not confluent. Except for CONFident, no tool available on the confluence platform CoCoWeb [13], which provides access to several confluence tools, was able to reach this conclusion, as join CTRSs are accepted (as part of COPS syntax), but currently unsupported by other confluence tools in the platform. CONFident is able to provide a negative answer using Proposition 22 to prove nonjoinability of the only CCP, and then Proposition 26 to conclude nonconfluence.

Dershowitz, Okada, and Sivakumar proved that *a terminating (*noetherian in their terminology*) join CTRSs is confluent if all its critical pairs are joinable overlays* [7, Theorem 4], where a (conditional) critical pair is an overlay if the critical position is the top position Λ [7, Definition 8].

▶ **Example 29.** Note that the CCP π for $\mathcal{R}$ in Example 19 is an overlay. It is joinable, as proved in the example (both for the join and oriented semantics). The CTRS $\mathcal{R}$ is terminating as the underlying TRS $\mathcal{R}_u = \{a \to b, f(c, x) \to a, f(x, x) \to b\}$ is clearly terminating. Thus, by [7, Theorem 4], $\mathcal{R}$ (viewed as a join CTRS) is confluent.

Unfortunately, this does *not* hold for oriented CTRSs.

▶ **Example 30.** The following oriented CTRS $\mathcal{R}$ [26, Counterexample 3.3]

$$a \to b \tag{25}$$
$$f(x) \to c \Leftarrow x \approx a \tag{26}$$

is terminating (the underlying TRS $\mathcal{R}_u = \{a \to b, f(x) \to c\}$ is clearly terminating), and has no (conditional) critical pair. However, $f(b) \leftarrow f(a) \to c$, but c is irreducible and $f(b)$ also is as the conditional part $x \approx a$ of rule (26), when instantiated by $b \approx a$ is not satisfiable by using a reachability test $b \to^* a$. Hence $f(b)$ and c are *not* joinable and $\mathcal{R}$ is not confluent.

Normal CTRSs are CTRSs where terms t in conditions $s \approx t$ of the conditional part of rules are *ground, irreducible terms.*

▶ Remark 31 (Normal join, oriented, and semiequational CTRSs). Nowadays, the notion of a normal CTRS $\mathcal{R}$ usually assumes that $\mathcal{R}$ is an oriented CTRS, see, e.g., [23, Definition 7.1.3]. Other authors, though, have defined the notion of a normal *join* CTRS as one whose joinability conditions $s \downarrow t$ in conditional rules always satisfy the restriction of t being an irreducible ground term [7, Definition 2]; then, the authors remark that normal join CTRSs can be seen as what we call normal CTRSs today. Hence, normal join and oriented CTRSs coincide. As for *semi-equational* CTRSs, if normality is required, then $s \leftrightarrow^* t$ if and only if $s \to^* t$ because t is irreducible. Therefore, when referring to normal join, oriented, or semiequational CTRSs we are actually dealing with one and the same kind of CTRSs.

For this reason, conditions of normal CTRSs can be equivalently handled as joinability conditions $s_i \downarrow t_i$. Neither $\mathcal{R}$ in Example 30 nor $\mathcal{R}$ in Example 19 are normal. The following result, which is a simple consequence of [7, Theorem 4], is useful:

▶ **Corollary 32.** *A terminating normal CTRS is confluent if all its critical pairs are joinable overlays.*

▶ **Example 33.** Consider the following normal CTRS

$$\begin{array}{rcll}
c & \to & b & (27)\\
d & \to & b & (28)\\
f(a,x) & \to & c \Leftarrow x \approx a & (29)\\
f(x,x) & \to & d \Leftarrow x \approx a & (30)\\
g(x) & \to & d \Leftarrow g(x) \approx b & (31)\\
g(a) & \to & f(a,a) & (32)
\end{array}$$

which is terminating, as the underlying TRS $\mathcal{R}_u$ is clearly terminating. The system has two (overlay) conditional critical pairs which are feasible and joinable:

$$\begin{array}{rcll}
\langle c,d\rangle & \Leftarrow & a \approx a & \text{with (29) and (30)} \quad (33)\\
\langle d,f(a,a)\rangle & \Leftarrow & g(a) \approx b & \text{with (31) and (32)} \quad (34)
\end{array}$$

As for (33), using (27) and (28) we join c and d into b. Regarding (34), we have $\underline{f(a,a)} \to_{(30)} d$. By Corollary 32, $\mathcal{R}$ is confluent. No tool in CoCoWeb is able to prove it.

An oriented CTRS $\mathcal{R}$ is called *deterministic* (DCTRS) if for each rule $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$ in $\mathcal{R}$ and each $1 \leq i \leq n$, we have $\mathcal{V}ar(s_i) \subseteq \mathcal{V}ar(\ell) \cup \bigcup_{j=1}^{i-1} \mathcal{V}ar(t_j)$. In the literature on confluence of DCTRSs, some results that use termination properties of CTRSs to guarantee confluence have been reported. For instance, [2, Theorem 4.1] establishes that *a quasi-reductive strongly deterministic CTRS is confluent if and only if its CCPs are all joinable.* Strongly deterministic CTRSs (SDCTRSs) are DCTRSs $\mathcal{R}$ where all conditions $s \approx t$ in the conditional part C of rules $\ell \to r \Leftarrow C \in \mathcal{R}$ are *strongly irreducible*, i.e., every instance $\sigma(t)$ of t by an irreducible substitution σ is irreducible [2, Definition 4.1]. Clearly, normal DCTRSs are SDCTRSs. Quasi-reductiveness (see, e.g., [23, Definition 7.2.36]) guarantees quasi-decreasingness of the DCTRS (see [23, Definition 7.2.39 and Lemma 7.2.40]). As proved in [16], quasi-decreasingness is equivalent to *operational termination.* Other results in the literature (see [23, Section 7.3]) usually require quasi-decreasingness, i.e, operational termination. Operationally terminating CTRSs are *terminating*, but not vice versa. For instance, viewed as an *oriented* CTRS, the (deterministic) CTRS $\mathcal{R}$ in Example 33 is terminating (this can be proved by using MU-TERM) but it is *not* operationally terminating (this can also be proved with MU-TERM). Therefore, the aforementioned results in [2, 23] *cannot* be used to prove confluence of $\mathcal{R}$ in Example 33. Further results for proving confluence of terminating CTRSs are reported in [10]; however, they apply to *join* CTRSs only.

Most confluence criteria for proving confluence of CTRSs involve checking (non)joinability of (feasible) CCPs, possibly in connection with other structural or syntactical requirements on the CTRS (e.g., left-linearity, etc.). The focus in this paper has been the investigation of (non)joinability criteria which can be used together with these confluence criteria. The following section discusses our implementation of those techniques and its impact in proofs of confluence of CTRSs in practice.

Table 1 Meaning of `CONDITIONTYPE` in COPS syntax.

`CONDITIONTYPE`	here `==` means
`ORIENTED`	$\to^*$
`JOIN`	$\downarrow$
`SEMI-EQUATIONAL`	$\leftrightarrow^*$

8 Implementation and Experimental Evaluation

CONFident 1.0 is written in Haskell and consists of 80 Haskell modules with around 14000 lines of code. The tool is accessible through its web interface (see Section 1). The input format is an extended version of the Confluence Competition (CoCo) format [21], which is the official format used in the *confluence* (CR) category of the competition. The input is a CTRS $\mathcal{R}$ in TPDB/COPS format[7]. As in COPS syntax, symbol `==` stands for $\approx$ above to specify the conditional part of rewrite rules. Its meaning depends on the `CONDITIONTYPE` section of the input specifying how the conditions of rules are evaluated [23, Definition 7.1.3] according to Table 1. Furthermore, in our extended version of TPDB/COPS syntax we can combine those relations by using them directly in the condition part of the rules: we use `->*` for $\to^*$, `->*<-` for $\downarrow$ and `<->*` for $\leftrightarrow^*$.

The implementation is based on a divide and conquer schema where, given an input problem, there is a set of techniques and an application strategy for those techniques. The techniques can simplify the problem, reduce it into a set of simpler problems or just give a positive or negative answer. We consider two types of problems: *Rewriting problems* and *Conditional Rewriting* problems. Each type of problem has its own strategy and processors. Although there are processors that can be applied to Rewriting and Conditional Rewriting problems indifferently, from the implementation point of view we prefer to implement them separately because we can apply simplifications when conditions are not considered. According to Section 3.1, when the system is parsed, the tool computes $\overline{\mathcal{R}}_J$, $\overline{\mathcal{R}}_O$, or $\overline{\mathcal{R}}_{\rm SE}$ (depending on the `CONDITIONTYPE` section) and then applies the appropriate strategy. Our proof strategy is based on experimentation: we try to first apply techniques that simplify the problems and reduce them into simpler problems (e.g., remove unnecesary rules and apply modularity results). When all simplification techniques have been applied, we analyze the problem in order to extract good properties that guide the strategy (linearity, weak normalization, termination, operational termination, strongly deterministic conditions, or right stability, see [16, 23] for definitions of these concepts). Then we calculate its conditional critical pairs and apply the techniques presented in the paper combined with classical techniques to check the joinability or non-joinability of the critical pairs. We also apply transformations to convert CTRSs into confluence equivalent TRSs and CS-TRSs [15]. If the final answer is YES or NO, the tool displays a report in plain text. Otherwise, MAYBE is returned. More details can be found in [28].

We participated in the CTRS (CR) category of CoCo 2021.[8] With a timeout of 60 seconds, the participating tools are expected to return a proof of confluence or nonconfluence (or a *maybe* answer) for each of the considered problems. The other participating tools this year were CO3 [22] and ACP [1]. The test set used in CoCo 2021 included 100 examples (see `http://cops.uibk.ac.at/results/?y=2021&c=CTRS`). The following table sumarizes the obtained results:[9]

CTRS CR Tool	Yes	No	Total
CONFident	37	24	61
CO3	28	19	47
ACP	29	15	44

Accordingly, CONFident was declared the winner of the competition.[10]

[7] See `http://zenon.dsic.upv.es/muterm/?name=documentation#formats`

[8] `http://project-coco.uibk.ac.at/2021/`

[9] The 2020 version of the tool ConCon `http://cl-informatik.uibk.ac.at/software/concon/` participated in CoCo 2021 as the winner of the CTRS category in 2020. Its results are displayed in the aforementioned web page.

[10] See `http://project-coco.uibk.ac.at/2021/results.php`

CONFident is able to obtain confluence proofs not only for *oriented* CTRSs (which is the focus of CoCo CTRS category) but also for *join* CTRSs as explained above (and currently unsupported by the tools participating in the confluence competition). Full proofs for the discussed examples of join CTRSs can also be found in the benchmarks section of the tool web site.

9 Related Work

Plaisted's presentation of conditional rewriting [24] is related to ours. Conditional rules are viewed as (universally quantified) formulas $C \Rightarrow \ell \to r$, which can be seen as first-order formulas. Semantic interpretations, though, consist of a *base* domain D_B (an "ordinary" domain as introduced in Section 3.3) and an *extended domain* $D_E = \mathcal{T}(\mathcal{F} \cup D_B)$ where values of the domain D_B are treated as *constants*. Symbols f have an interpretation[11] f^I, i.e., a mapping $f^I : D_E \times \cdots \times D_E \to D_E$ defined so that $f^I(t_1, \ldots, t_k) = f(t_1, \ldots, t_k)$. Conveniently, if $a \in D_B$, then $a^I = a$. Terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ are interpreted as usual, except that Plaisted also interprets variables $x \in \mathcal{X}$ as $x^I \in D_E$. The usual *valuation of variables* of first-order logic is therefore integrated as part of the interpretation I. In this respect, his semantic approach differs from the usual one in first order logic (indeed, he rather speaks of *term logic* when referring it). Predicates $\to$ and $\to^*$ are interpreted as subsets of $D_E \times D_E$. Atoms $s \to t$ and $s \to^* t$ are then interpreted as expected: $(s \to t)^I = s^I \to^I t^I$ and $(s \to^* t)^I = s^I (\to^*)^I t^I$. An interpretation I is a *rewriting model* of a CTRS $\mathcal{R}$ if I satisfies the formulas in $\mathcal{R}$ together with a number of *axioms* A which, essentially, are the ones we obtain from the inference rules (Rf), (T), and $(\mathrm{C})_{f,i}$ for $f \in \mathcal{F}$ and $1 \leq i \leq ar(f)$. Plaisted writes $\mathcal{R} \models_m \varphi$ if φ is true in *all* minimal rewriting models of $\mathcal{R}$.[12] Then, a CTRS $\mathcal{R}$ is said to be *confluent* if $\mathcal{R} \models_m \varphi_{\mathrm{CR}}$ holds. Finally, on page 219, the confluence property of a CTRS is proved equivalent to $\mathcal{R}_I \models_m \varphi_{\mathrm{CR}}$ for all minimal rewriting models I of $\mathcal{R}$, where $\mathcal{R}_I$ is the (possibly infinite) TRS (i.e., without conditional rules) obtained from $\mathcal{R}$ and I by considering rules $s \to t$ (where $s, t \in \mathcal{T}(\mathcal{F} \cup D_B)$) such that $s \to t$ is a ground instance of $\ell \to r$ for some conditional rule $\ell \to r \Leftarrow C$, where variables are replaced by terms in $\mathcal{T}(\mathcal{F} \cup D_B)$ and the corresponding instance C' of C is true in I. Similar definitions are provided for local confluence and joinability of critical pairs (which Plaisted calls to *pass the critical pair test*). Note that, since there can be infinitely many interpretations I for a given CTRS $\mathcal{R}$, proofs of confluence in term logic involve the consideration of infinitely many TRSs $\mathcal{R}_I$. In contrast, our definitions of confluence, local confluence, and joinability of CCPs use a single model $\mathcal{M}_{\mathcal{R}}$.

In the so-called *first-order theory of rewriting* (*FOThR* in the following), a restricted first-order language (without constant or function symbols), is used. The predicate symbols $\to$ and $\to^*$ are interpreted on the least (ground) Herbrand model $\mathcal{H}_{\mathcal{R}}$ for the signature $\mathcal{F}$ and predicates $\to$ and $\to^*$ [6]. In *FOThR* properties of TRSs $\mathcal{R} = (\mathcal{F}, R)$ are expressed by satisfiability in $\mathcal{H}_{\mathcal{R}}$ of *FOThR*. For instance $\mathcal{H}_{\mathcal{R}} \models \varphi_{\mathrm{CR}}$ means "*the TRS $\mathcal{R}$ is ground confluent*" (the restriction to ground confuence is due to the use of $\mathcal{H}_{\mathcal{R}}$, which consists of atoms $s \to t$ and $s \to^* t$ where $s, t \in \mathcal{T}(\mathcal{F})$). Decision algorithms for *FOThR* exist for restricted classes of TRSs $\mathcal{R}$ like left-linear right-ground TRSs, where variables are allowed in the left-hand side of the rules (without repeated occurrences of the same variable) but disallowed in the right-hand side [25]. However, a simple fragment of *FOThR* like the *First-Order Theory of One-Step Rewriting*, where only a single predicate symbol $\to$ representing one-step rewritings with $\mathcal{R}$ is allowed, has been proved undecidable even for *linear* TRSs [27]. In contrast, we use the full expressive power of first-order logic to represent not only TRSs but also CTRSs . Also in contrast to *FOThR*, where function symbols are not allowed in formulas, we can use *arbitrary* sentences involving arbitrary terms. This is crucial, for instance, to investigate joinability of CCPs $\langle s, t \rangle \Leftarrow C$, as s and t are arbitrary terms, and C usually involves nonvariable terms.

On the other hand, the idea of turning variables into constants to see terms with variables as ground terms of an extended signature is standard in algebraic specifications, see, e.g., [9, page 9]. However, as far as we know, such a model has not been used in the definition or verification of

[11] Plaisted interprets symbols in two different ways. This is due to the more general kind of conditional systems he considers, where the conditional part of rules can include first-order literals defined by an additional first-order theory. Our simplified presentation suffices to handle the CTRSs considered here.

[12] Plaisted obtains each of such minimal models as follows: given an interpretation I, he takes the least model of the Horn clauses obtained as the ground instances of rules $\alpha : \ell \to r \Leftarrow C$ when variables in α are replaced by terms in $\mathcal{T}(\mathcal{F} \cup D_B)$ (see the proof of his Theorem 1).

computational properties like confluence, which is the main focus of this paper. Also, the use of $\mathcal{M}_{\mathcal{R}}$ in Section 4 to define joinability of CCPs as satisfaction in $\mathcal{M}_{\mathcal{R}}$, and the translation in Section 6 of joinability problems into feasibility problems where terms with variables are "grounded" using $\_^{\downarrow}$ is, to the best of our knowledge, also new.

Research on confluence of CTRSs goes back to [4, 7], and many advances have been introduced in the last years, leading to the construction of several tools which are able to automatically prove it, see [21] and the references therein. To the best of our knowledge, though, our characterization of (local) confluence of CTRSs as the satisfiability of appropriate logical formulas in $\mathcal{M}_{\mathcal{R}}$ (Theorem 11) and its practical use in Section 7 is new. Also, the idea of decomposing proofs of confluence into (in)feasibility problems by taking into account the structure of the logic formula, and the use of constants instead of variables to improve these proves seems to be new.

10 Conclusions and Future Work

In this paper, we deal with computational (reduction) relations $\to$ and $\to^*$ associated to reduction-based systems $\mathcal{R}$ in logic form: reduction steps are defined by provability in a given inference system $\mathcal{I}(\mathcal{R})$ obtained from $\mathcal{R}$ and the generic system describing the operational semantics of the language of $\mathcal{R}$, or, equivalently, as logical consequences of a theory $\overline{\mathcal{R}}$ obtained similarly. We have characterized (local) confluence of CTRSs $\mathcal{R}$ as the satisfiability of appropriate first-order formulas φ_{WCR} and φ_{CR} in a canonical model $\mathcal{M}_{\mathcal{R}}$ where variables are treated as constants and terms with variables in $\mathcal{T}(\mathcal{F},\mathcal{X})$ are treated as ground terms in $\mathcal{T}(\mathcal{F}\cup C_{\mathcal{X}})$ (Theorem 11). We have also similarly characterized joinability of CCPs $\langle s,t\rangle \Leftarrow C$ (Proposition 15). Then, we show how to translate joinability problems into (combinations of) feasibility problems which can be solved using appropriate techniques and tools. For this purpose, the introduction of constants $c_x \in C_{\mathcal{X}}$ instead of variables $x \in \mathcal{X}$ in feasibility goals has been useful to obtain faster proofs.

We have developed a new tool implementing our results: CONFident. We participated in the 2021 edition of the Confluence Competition (CoCo) in the CTRS CR (confluence of CTRSs) category obtaining good results.

As for future work, we plan to apply our techniques to prove confluence of rewriting-based programming languages like Maude, whose conditional rules include componenents not explicitly considered here (matching conditions, etc.) but whose semantics can be defined by using the general approach sketched in Section 3. Since the analysis of confluence of rewrite theories (which provide the formal basis for the operational description of Maude programs) is also based in the analysis of joinability of the appropriate critical pairs [8], we think that our approach will be useful as well.

References

1 Takahito Aoto, Junichi Yoshida, and Yoshihito Toyama. Proving confluence of term rewriting systems automatically. In Ralf Treinen, editor, *Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings*, volume 5595 of *Lecture Notes in Computer Science*, pages 93–102. Springer, 2009. doi:10.1007/978-3-642-02348-4_7.

2 Jürgen Avenhaus and Carlos Loría-Sáenz. On conditional rewrite systems with extra variables and deterministic logic programs. In Frank Pfenning, editor, *Logic Programming and Automated Reasoning, 5th International Conference, LPAR'94, Kiev, Ukraine, July 16-22, 1994, Proceedings*, volume 822 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 1994. doi:10.1007/3-540-58216-9_40.

3 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

4 Jan A. Bergstra and Jan Willem Klop. Conditional rewrite rules: Confluence and termination. *J. Comput. Syst. Sci.*, 32(3):323–362, 1986. doi:10.1016/0022-0000(86)90033-4.

5 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007. doi:10.1007/978-3-540-71999-1.

6 Max Dauchet and Sophie Tison. The theory of ground rewrite systems is decidable. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 242–248. IEEE Computer Society, 1990. doi:10.1109/LICS.1990.113750.

7 Nachum Dershowitz, Mitsuhiro Okada, and G. Sivakumar. Confluence of conditional rewrite systems. In Stéphane Kaplan and Jean-Pierre Jouannaud, editors, *Conditional Term Rewriting Systems, 1st International Workshop, Orsay, France, July 8-10, 1987, Proceedings*, volume 308 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 1987. doi:10.1007/3-540-19242-5_3.

8 Francisco Durán and José Meseguer. On the church-rosser and coherence properties of conditional order-sorted rewrite theories. *J. Log. Algebraic Methods Program.*, 81(7-8):816–850, 2012. doi:10.1016/j.jlap.2011.12.004.

9 Joseph A. Goguen and José Meseguer. Models and equality for logical programming. In Hartmut Ehrig, Robert A. Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT'87: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Pisa, Italy, March 23-27, 1987, Volume 2: Advanced Seminar on Foundations of Innovative Software Development II and Colloquium on Functional and Logic Programming and Specifications (CFLP)*, volume 250 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 1987. doi:10.1007/BFb0014969.

10 Bernhard Gramlich and Claus-Peter Wirth. Confluence of terminating conditional rewrite systems revisited. In Harald Ganzinger, editor, *Rewriting Techniques and Applications, 7th International Conference, RTA-96, New Brunswick, NJ, USA, July 27-30, 1996, Proceedings*, volume 1103 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1996. doi:10.1007/3-540-61464-8_56.

11 Raúl Gutiérrez and Salvador Lucas. Automatically proving and disproving feasibility conditions. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 416–435. Springer, 2020. doi:10.1007/978-3-030-51054-1_27.

12 Raúl Gutiérrez and Salvador Lucas. mu-term: Verify termination properties automatically (system description). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 436–447. Springer, 2020. doi:10.1007/978-3-030-51054-1_28.

13 Nao Hirokawa, Julian Nagele, and Aart Middeldorp. Cops and CoCoWeb: Infrastructure for Confluence Tools. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 346–353. Springer, 2018. doi:10.1007/978-3-319-94205-6_23.

14 Salvador Lucas. Proving semantic properties as first-order satisfiability. *Artif. Intell.*, 277, 2019. doi:10.1016/j.artint.2019.103174.

15 Salvador Lucas. Applications and extensions of context-sensitive rewriting. *Journal of Logical and Algebraic Methods in Programming*, 121:100680, 2021. doi:10.1016/j.jlamp.2021.100680.

16 Salvador Lucas, Claude Marché, and José Meseguer. Operational termination of conditional term rewriting systems. *Inf. Process. Lett.*, 95(4):446–453, 2005. doi:10.1016/j.ipl.2005.05.002.

17 Salvador Lucas and José Meseguer. Dependency pairs for proving termination properties of conditional term rewriting systems. *J. Log. Algebr. Meth. Program.*, 86(1):236–268, 2017. doi:10.1016/j.jlamp.2016.03.003.

18 Salvador Lucas, José Meseguer, and Raúl Gutiérrez. The 2D Dependency Pair Framework for conditional rewrite systems. Part I: Definition and basic processors. *J. Comput. Syst. Sci.*, 96:74–106, 2018. doi:10.1016/j.jcss.2018.04.002.

19 William McCune. Prover9 & Mace4. Technical report, University of New Mexico, 2005–2010. URL: http://www.cs.unm.edu/~mccune/prover9/.

20 Elliott Mendelson. *Introduction to mathematical logic (4. ed.).* Chapman and Hall, 1997.

21 A. Middeldorp, J. Nagele, and K. Shintani. CoCo 2019: report on the eight confluence competition. *J.International Journal on Software Tools for Technology Transfer*, to appear, 2021. doi:10.1007/s10009-021-00620-4.

22 Naoki Nishida, Makishi Yanagisawa, and Karl Gmeiner. On Proving Confluence of Conditional Term Rewriting Systems via the Computationally Equivalent Transformation. In *3rd International Workshop on Confluence, IWC 2014, July 13, 2014, Vienna, Austria*, page 42, 2014.

23 Enno Ohlebusch. *Advanced topics in term rewriting.* Springer, 2002.

24 David A. Plaisted. A logic for conditional term rewriting systems. In Stéphane Kaplan and Jean-Pierre Jouannaud, editors, *Conditional Term Rewriting Systems, 1st International Workshop, Orsay, France, July 8-10, 1987, Proceedings*, volume 308 of *Lecture Notes in Computer Science*, pages 212–227. Springer, 1987. doi:10.1007/3-540-19242-5_16.

25 Franziska Rapp and Aart Middeldorp. Automating the first-order theory of rewriting for left-linear right-ground rewrite systems. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 36:1–36:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.FSCD.2016.36.

26 Taro Suzuki, Aart Middeldorp, and Tetsuo Ida. Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In Jieh Hsiang, editor, *Rewriting Techniques and Applications, 6th International Conference, RTA-95, Kaiserslautern, Germany, April 5-7, 1995, Proceedings*, volume 914 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 1995. doi:10.1007/3-540-59200-8_56.

27 Ralf Treinen. The first-order theory of linear one-step rewriting is undecidable. *Theor. Comput. Sci.*, 208(1-2):179–190, 1998. doi:10.1016/S0304-3975(98)00083-8.

28 Miguel Vítores. *CONFident: a tool for confluence analysis of rewriting systems.* PhD thesis, Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València, December 2021.

A Proofs of theorems

▶ **Proposition 6.** *Let $\mathcal{R} = (\mathcal{F}, R)$ be a CTRS and $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Then, $s \to_{\mathcal{R}} t$ if and only if $s^{\downarrow} \to_{\mathcal{R}} t^{\downarrow}$ and $s \to^*_{\mathcal{R}} t$ if and only if $s^{\downarrow} \to^*_{\mathcal{R}} t^{\downarrow}$.*

Proof. We develop the proof for oriented CTRSs. For join or equational CTRSs, it is similar. We proceed by multiple induction on the depth d of the proof trees used to prove each goal $s \to t$ (for $s \to_{\mathcal{R}} t$) and $s \to^* t$ (for $s \to^*_{\mathcal{R}} t$). If $d = 0$, then we consider two cases (we develop the *only if* part; the *if* part is analogous):

- $s \to t$ is proved using $(\text{Rl})_\alpha$ for an unconditional rule $\alpha : \ell \to r$, i.e., there is a substitution σ such that $s = \sigma(\ell)$ and $t = \sigma(r)$. Since $s^{\downarrow} = \sigma(\ell)^{\downarrow} = \sigma^{\downarrow}(\ell)$ and $t^{\downarrow} = \sigma(\ell)^{\downarrow} = \sigma^{\downarrow}(\ell)$, we have that $s^{\downarrow} \to t^{\downarrow}$ is proved using the same rule.
- $s \to t$ is proved using (Rf). In this case, $s = t$ and hence $s^{\downarrow} \to^* t^{\downarrow}$ is proved using (Rf).

If $d > 0$, then

- $s \to t$ is proved in one of the following two possible ways:
  - using rule $(\mathrm{C})_{f,i}$ where $s = f(s_1, \ldots, s_i, \ldots s_k)$, $t = f(s_1, \ldots, t_i, \ldots, s_k)$ for some terms $s_1, \ldots, s_k, t_i$, using a proof tree $\frac{T}{s \to t}$, where T is of depth $d - 1$ and $s_i \to t_i$ is the root of T. By the induction hypothesis, $s_i^{\downarrow} \to t_i^{\downarrow}$ can be proved and hence $f(s_i^{\downarrow}, \ldots, s_i^{\downarrow}, \ldots, s_k^{\downarrow}) = s^{\downarrow} \to t^{\downarrow} = f(s_i^{\downarrow}, \ldots, t_i^{\downarrow}, \ldots, s_k^{\downarrow})$ can be proved as well using $(\mathrm{C})_{f,i}$.
  - using $(\mathrm{Rl})_{\alpha}$ for some rule $\alpha : \ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$ and proof tree $\frac{T_1 \quad \cdots \quad T_n}{s \to t}$, where $s = \sigma(\ell)$ and $t = \sigma(r)$ for some substitution σ, and, for all $1 \leq i \leq n$ T_i, is a proof tree with root $\sigma(s_i) \to^* \sigma(t_i)$ and depth at most $d - 1$. By the induction hypothesis, $\sigma(s_i)^{\downarrow} \to^* \sigma(t_i)^{\downarrow}$ can be proved for all $1 \leq i \leq n$ using proof trees $T_i^{\downarrow}$ with root $\sigma(s_i)^{\downarrow} \to^* \sigma(t_i)^{\downarrow}$. Since for all terms $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\sigma(u)^{\downarrow} = \sigma^{\downarrow}(u)$, there is a proof of $s^{\downarrow} = \sigma^{\downarrow}(\ell) \to \sigma^{\downarrow}(\ell) = t^{\downarrow}$ using $(\mathrm{Rl})_{\alpha}$ with proof tree $\frac{T_1^{\downarrow} \quad \cdots \quad T_n^{\downarrow}}{s^{\downarrow} \to t^{\downarrow}}$.
- $s \to^* t$ is proved using (T) using a proof tree $\frac{T_1 \quad T_2}{s \to^* t}$ where T_1 is a proof tree with root $s \to u$ of depth at most $d - 1$ for some term u and T_2 is a proof tree with root $u \to^* t$ of depth at most $d - 1$. By the induction hypothesis, there are proof trees $T_1^{\downarrow}$ and $T_2^{\downarrow}$ with roots $s^{\downarrow} \to u^{\downarrow}$ and $u^{\downarrow} \to^* t^{\downarrow}$. Thus, $s^{\downarrow} \to^* t^{\downarrow}$ is proved by the proof tree $\frac{T_1^{\downarrow} \quad T_2^{\downarrow}}{s^{\downarrow} \to^* t^{\downarrow}}$. ◀

▶ **Theorem 8** *For all CTRSs $\mathcal{R}$, $\mathcal{M}_{\mathcal{R}} \models \overline{\mathcal{R}}$.*

Proof. We develop the proof for oriented CTRSs, for join and semi-equational CTRSs being similar. We consider the sentences derived from each of the four inference rules in $\mathfrak{I}_{\text{O-CTRS}}$:

- From rule (Rf) a single sentence $(\forall x)\; x \to^* x \in \overline{\mathcal{R}}$ is obtained. We need to prove that for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $(t^{\downarrow}, t^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ holds (remind that $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\mathcal{T}(\mathcal{F}_{\mathcal{X}})$ are isomorphic). Since for all terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $t \to^*_{\mathcal{R}} t$ can be proved in $\mathcal{I}(\mathcal{R})$ by using axiom (Rf), by definition of $\mathcal{M}_{\mathcal{R}}$, we have $(t^{\downarrow}, t^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ as required.
- From rule (T), a single sentence $(\forall x, y, z)\; x \to y \wedge y \to^* z \Rightarrow x \to^* z \in \overline{\mathcal{R}}$ is obtained. Then, $\mathcal{M}_{\mathcal{R}} \models (\forall x, y, z)\; x \to y \wedge y \to^* z \Rightarrow x \to^* z$ holds if and only if for all substitutions $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$, whenever both $(\sigma^{\downarrow}(x), \sigma^{\downarrow}(y)) \in \to^{\mathcal{M}_{\mathcal{R}}}$ and $(\sigma^{\downarrow}(y), \sigma^{\downarrow}(z)) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ hold, then $(\sigma^{\downarrow}(x), \sigma^{\downarrow}(z)) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ holds as well. If both $(\sigma^{\downarrow}(x), \sigma^{\downarrow}(y)) \in \to^{\mathcal{M}_{\mathcal{R}}}$ and $(\sigma^{\downarrow}(y), \sigma^{\downarrow}(z)) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ hold, then, by definition of $\mathcal{M}_{\mathcal{R}}$, we have $\sigma(x) \to_{\mathcal{R}} \sigma(y)$ and $\sigma(y) \to^*_{\mathcal{R}} \sigma(z)$. Hence, $\sigma(x) \to^*_{\mathcal{R}} \sigma(z)$ can be proved in $\mathcal{I}(\mathcal{R})$ and therefore $(\sigma^{\downarrow}(x), \sigma^{\downarrow}(z)) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ as desired.
- For all k-ary symbols $f \in \mathcal{F}$ and $1 \leq i \leq k$, from $(\mathrm{C})_{f,i}$ a sentence $(\forall x_1) \cdots (\forall x_k)(\forall y_i) x_i \to y_i \Rightarrow f(x_1, \ldots, x_i, \ldots, x_k) \to f(x_1, \ldots, y_i, \ldots, x_k)$ is obtained. It holds in $\mathcal{M}_{\mathcal{R}}$ because, for all terms $s_1, \ldots, s_k, t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, if $(s_i^{\downarrow}, t_i^{\downarrow}) \in \to^{\mathcal{M}_{\mathcal{R}}}$, then, by definition of $\mathcal{M}_{\mathcal{R}}$, $s_i \to_{\mathcal{R}} t_i$ can be proved in $\mathcal{I}(\mathcal{R})$, and using $(\mathrm{C})_{f,i}$ we know that $f(s_1, \ldots, s_i, \ldots, s_k) \to_{\mathcal{R}} f(s_1, \ldots, t_i, \ldots, s_k)$ can also be proved, i.e., $(f(s_1^{\downarrow}, \ldots, s_i^{\downarrow}, \ldots, s_k^{\downarrow}), f(s_1^{\downarrow}, \ldots, t_i^{\downarrow}, \ldots, s_k^{\downarrow})) \in \to^{\mathcal{M}_{\mathcal{R}}}$ holds.
- As for $(\mathrm{Rl})_{\alpha}$, with $\alpha : \ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \to t_n$, there is a sentence $(\forall \vec{x}) \bigwedge_{i=1}^{n} s_i \to^* t_i \Rightarrow \ell \to r$ in $\overline{\mathcal{R}}$. Then, $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x}) \bigwedge_{i=1}^{n} s_i \to^* t_i \Rightarrow \ell \to r$ holds if and only if for all substitutions $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$, whenever $(\sigma^{\downarrow}(s_i), \sigma^{\downarrow}(t_i)) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ holds for all $1 \leq i \leq n$, then $(\sigma^{\downarrow}(\ell), \sigma^{\downarrow}(r)) \in \to^{\mathcal{M}_{\mathcal{R}}}$ holds as well. By definition of $\mathcal{M}_{\mathcal{R}}$, if $(\sigma^{\downarrow}(s_i), \sigma^{\downarrow}(t_i)) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ holds for all $1 \leq i \leq n$, then $\sigma(s_i) \to^*_{\mathcal{R}} \sigma(t_i)$ holds for all $1 \leq i \leq n$. Therefore, $\sigma(\ell) \to_{\mathcal{R}} \sigma(r)$ can be proved in $\mathcal{I}(\mathcal{R})$, and hence $(\sigma^{\downarrow}(\ell), \sigma^{\downarrow}(r)) \in \to^{\mathcal{M}_{\mathcal{R}}}$, as desired. ◀

▶ **Proposition 9.** *Let $\mathcal{R} = (\mathcal{F}, R)$ be a CTRS, $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and $\vec{x} = x_1, \ldots, x_n$ denote the variables occurring in s and t, i.e., $\mathcal{V}ar(s) \cup \mathcal{V}ar(t) = \{x_1, \ldots, x_n\}$. Then,*

1. *We have that $\sigma(s) \to^*_{\mathcal{R}} \sigma(t)$ for all substitutions $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$, if and only if $(s^{\downarrow}, t^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$.*
2. *$\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x})\; s \to^* t$ if and only if $(s^{\downarrow}, t^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$.*

Proof.

1. As for the *if* part, if $(s^{\downarrow}, t^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ holds, then, by definition of $\mathcal{M}_{\mathcal{R}}$, $s \to^*_{\mathcal{R}} t$ holds. By closedness of $\to^*$ under substitution application, for all substitutions σ, we have $\sigma(s) \to^*_{\mathcal{R}} \sigma(t)$. Regarding the *only if* part, assume that for all substitutions σ, $\sigma(s) \to^*_{\mathcal{R}} \sigma(t)$ holds. In particular, for the empty substitution ϵ, we have $s = \epsilon(s) \to^*_{\mathcal{R}} \epsilon(t) = t$, i.e., $(s^{\downarrow}, t^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$.

2. The *if* part is as in the previous item, considering the definition of satisfiability in $\mathcal{M}_{\mathcal{R}}$ of a universally quantified formula. Regarding the *only if* part, if $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x})\, s \to^* t$ holds, then for all substitutions $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$, $(\sigma^{\downarrow}(s), \sigma^{\downarrow}(t)) \in (\to^*_{\mathcal{R}})^{\mathcal{M}_{\mathcal{R}}}$ holds. In particular, for the empty substitution ϵ, we have $\epsilon^{\downarrow}(s) = s^{\downarrow}$ and $\epsilon^{\downarrow}(t) = t^{\downarrow}$, i.e., $(s^{\downarrow}, t^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ holds. ◀

▶ **Theorem 11.** *A CTRS is (locally) confluent if and only if $\mathcal{M}_{\mathcal{R}} \models \varphi_{CR}$ (resp. $\mathcal{M}_{\mathcal{R}} \models \varphi_{WCR}$) holds.*

Proof. We develop the proof for confluence (i.e., φ_{CR}). For local confluence (i.e., φ_{WCR})) it is analogous. For the *if* part, if $\mathcal{M}_{\mathcal{R}} \models \varphi_{CR}$ holds, then, for all terms $s, t, u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, whenever both $(s^{\downarrow}, t^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ and $(s^{\downarrow}, u^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ hold, there is $v \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that both $(t^{\downarrow}, v^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ and $(u^{\downarrow}, v^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ hold. By using Proposition 9, we conclude that, if $s \to^*_{\mathcal{R}} t$ and $s \to^*_{\mathcal{R}} u$ hold, then $t \to^*_{\mathcal{R}} v$ and $u \to^*_{\mathcal{R}} v$. Hence, $\mathcal{R}$ is confluent.

As for the *only if* part, if $\mathcal{R}$ is confluent, then for all terms $s, t, u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, whenever $s \to^*_{\mathcal{R}} t$ and $s \to^*_{\mathcal{R}} u$, there is $v \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $t \to^*_{\mathcal{R}} v$ and $u \to^*_{\mathcal{R}} v$. By definition of $\mathcal{M}_{\mathcal{R}}$, this means that whenever $(s^{\downarrow}, t^{\downarrow}), (s^{\downarrow}, u^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$, we also have $(t^{\downarrow}, v^{\downarrow}), (u^{\downarrow}, v^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$. Thus, by Proposition 9, $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x}) s \to^* t \wedge s \to^* u$ implies $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x}) t \to^* v \wedge u \to^* v$, i.e., $\mathcal{M}_{\mathcal{R}} \models \varphi_{CR}$ holds. ◀

▶ **Proposition 15.** *Let $\mathcal{R}$ be a CTRS. A CCP $\pi : \langle s, t\rangle \Leftarrow C$ is joinable if and only if $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x})(\exists z)\, C \Rightarrow s \to^* z \wedge t \to^* z$ holds, where $\vec{x} = x_1, \ldots, x_m$ are the variables occurring in C, s, t and $z \notin \mathcal{V}ar(C, s, t)$.*

Proof. We treat the particular case of oriented CTRSs. For join or semi-equational CTRSs it is similar. Let $C = s_1 \approx t_1, \ldots, s_n \approx t_n$. As for the *only if* part, if π is joinable, then for all substitutions $\sigma \in \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $\sigma(C)$ holds, i.e., for all $1 \leq i \leq n$, $\sigma(s_i) \to^*_{\mathcal{R}} \sigma(t_i)$ holds, there is a term $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $\sigma(s) \to^*_{\mathcal{R}} u$ and $\sigma(t) \to^*_{\mathcal{R}} u$ holds as well. By Proposition 6, if $\sigma(C)$ holds, then $\sigma^{\downarrow}(C)$ holds as well. Furthermore, if $\sigma(s) \to^*_{\mathcal{R}} u$ and $\sigma(t) \to^*_{\mathcal{R}} u$, then $\sigma^{\downarrow}(s) \to^*_{\mathcal{R}} u^{\downarrow}$ and $\sigma^{\downarrow}(t) \to^*_{\mathcal{R}} u^{\downarrow}$. Therefore, for all substitutions $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$, whenever $\sigma^{\downarrow}(C)$ holds, then there is $z \in \mathcal{T}(\mathcal{F}_{\mathcal{X}})$ such that $\sigma^{\downarrow}(s) \to^* z \wedge \sigma^{\downarrow}(t) \to^* z$ holds as well, i.e., $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x})(\exists z)\, C \Rightarrow s \to^* z \wedge t \to^* z$ holds.

As for the *if* part, if $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x})(\exists z) C \Rightarrow s \to^* z \wedge t \to^* z$ holds, then by definition of satisfiability in $\mathcal{M}_{\mathcal{R}}$, for all substitutions $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$, if $(\sigma(s_i)^{\downarrow}, \sigma(t_i)^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ holds for all $1 \leq i \leq n$, then there is $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that both $(\sigma(s)^{\downarrow}, u^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ and $(\sigma(t)^{\downarrow}, u^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ hold as well. By definition of $\mathcal{M}_{\mathcal{R}}$, for all substitutions σ, whenever $\sigma(s_i) \to^*_{\mathcal{R}} \sigma(t_i)$ holds for all $1 \leq i \leq n$, we have $\sigma(s) \to^* u$ and $\sigma(s) \to^* u$, i.e., π is joinable. ◀

▶ **Corollary 17.** *Let $\mathcal{R}$ be a CTRS and $\pi : \langle s, t\rangle \Leftarrow C$ be a critical pair. If $\overline{\mathcal{R}} \vdash (\forall \vec{x})(\exists z)\, C \Rightarrow s \to^* z \wedge t \to^* z$ holds, then π is joinable.*

Proof. If $\overline{\mathcal{R}} \vdash (\forall \vec{x})(\exists z)\, C \Rightarrow s \to^* z \wedge t \to^* z$ holds, then $\overline{\mathcal{R}} \models (\forall \vec{x})(\exists z)\, C \Rightarrow s \to^* z \wedge t \to^* z$ holds as well. By Theorem 8, $\mathcal{M}_{\mathcal{R}} \models \overline{\mathcal{R}}$ holds. Hence, $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x})(\exists z)\, C \Rightarrow s \to^* z \wedge t \to^* z$ holds. By Proposition 15, π is joinable. ◀

▶ **Corollary 18.** *Let $\mathcal{R}$ be a CTRS and $\pi : \langle s, t\rangle \Leftarrow C$ be a critical pair. If $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is $\overline{\mathcal{R}}$-feasible, then π is joinable.*

Proof. If $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is $\overline{\mathcal{R}}$-feasible, there is a term $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $s^{\downarrow} \to^*_{\mathcal{R}} u^{\downarrow}$ and $t^{\downarrow} \to^*_{\mathcal{R}} u^{\downarrow}$, i.e., by Proposition 6, $s \to^*_{\mathcal{R}} u$ and $t \to^*_{\mathcal{R}} u$, hence $(s^{\downarrow}, u^{\downarrow}), (t^{\downarrow}, u^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$. By Proposition 9, $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x})\, s \to^* u \wedge t^{\downarrow} \to^* u$, i.e., $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x})(\exists z)\, s \to^* z \wedge t^{\downarrow} \to^* z$ holds. By Proposition 15, π is joinable. ◀

▶ **Proposition 20.** *Let $\mathcal{R}$ be a CTRS and $\pi : \langle s, t\rangle \Leftarrow C$ be a critical pair such that $C^{\downarrow}$ is $\overline{\mathcal{R}}$-feasible. If $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is $\overline{\mathcal{R}}$-infeasible, then π is not joinable.*

Proof. By contradiction. If π is joinable, then for all substitutions $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$, if $\sigma(C)$ holds, then there is a term u such that $\sigma(s) \to^* u$ and $\sigma(t) \to^* u$. Since $C^{\downarrow}$ is $\overline{\mathcal{R}}$-feasible, no instantiation of variables in C is necessary for the condition C of π to hold, i.e., $\epsilon(C)$ holds and therefore $\epsilon(s) = s \downarrow_{\mathcal{R}} t = \epsilon(t)$ holds as well. By Proposition 6, $s^{\downarrow} \downarrow_{\mathcal{R}} t^{\downarrow}$ holds, i.e., $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is $\overline{\mathcal{R}}$-feasible, leading to a contradiction. ◀

▶ **Proposition 22.** *Let $\mathcal{R}$ be a CTRS and $\pi : \langle s, t\rangle \Leftarrow C$ be a critical pair such that $\mathcal{V}ar(s,t) \cap \mathcal{V}ar(C) = \emptyset$. Then, π is joinable if and only if C is $\overline{\mathcal{R}}$-infeasible or $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is $\overline{\mathcal{R}}$-feasible.*

Proof. By Proposition 15, π is joinable if and only if $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{x})(\exists z) C \Rightarrow s \to^* z \wedge t \to^* z$ holds. Since $\mathcal{V}ar(s,t) \cap \mathcal{V}ar(C) = \emptyset$, this is equivalent to $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{y}_1)\neg C \vee (\forall \vec{y}_2)(\exists z) s \to^* z \wedge t \to^* z$, where $\vec{y}_1$ are the variables $\mathcal{V}ar(C)$ and $\vec{y}_2$ are the variables $\mathcal{V}ar(s,t)$, with $\vec{y}_1 \cap \vec{y}_2 = \emptyset$ and $\vec{x} = \vec{y}_1 \cup \vec{y}_2$. This is equivalent to (i) $\mathcal{M}_{\mathcal{R}} \models \neg(\exists \vec{y}_1) C$ or (ii) $\mathcal{M}_{\mathcal{R}} \models (\forall \vec{y}_2)(\exists z) s \to^* z \wedge t \to^* z$. By definition of satisfiability in $\mathcal{M}_{\mathcal{R}}$ and using Proposition 6, (ii) is equivalent to the existence of a term u such that both $(s^{\downarrow}, u^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ and $(t^{\downarrow}, u^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ hold.

- Now, for the *if* part, we show that $\overline{\mathcal{R}}$-infeasibility of C implies (i) and $\overline{\mathcal{R}}$-feasibility of $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ implies (ii). First, if C is $\overline{\mathcal{R}}$-infeasible, then there is no substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $\overline{\mathcal{R}} \vdash \sigma(C)$ holds. This clearly implies $\mathcal{M}_{\mathcal{R}} \models \neg(\exists \vec{y}_1) C$; otherwise, there would be a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $\mathcal{M}_{\mathcal{R}} \models \sigma(C)$ holds. By Proposition 6, though, this implies that $\overline{\mathcal{R}} \vdash \sigma(C)$ holds as well, leading to a contradiction. Second, if $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is $\overline{\mathcal{R}}$-feasible, then there is $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $\overline{\mathcal{R}} \vdash s^{\downarrow} \to^* u^{\downarrow}$ and $\overline{\mathcal{R}} \vdash t^{\downarrow} \to^* u^{\downarrow}$, i.e., $s \to^*_{\mathcal{R}} u$ and $t \to^*_{\mathcal{R}} u$ holds. Therefore, $\mathcal{M}_{\mathcal{R}} \models (\exists z) s \to^* z \wedge t \to^* z$ holds and π is joinable.
- For the *only if* part, if (i) holds, then there is no substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $\mathcal{M}_{\mathcal{R}} \models \sigma(C)$ holds. If C would be $\overline{\mathcal{R}}$-feasible, though, then, by [11, Theorem 1], $\overline{\mathcal{R}} \vdash (\exists \vec{y}_1) C$ holds. By using Theorem 8, we then conclude that $\mathcal{M}_{\mathcal{R}} \models (\exists \vec{y}_1) C$ holds, leading to a contradiction. Finally, if (ii) holds, then both $(s^{\downarrow}, u^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ and $(t^{\downarrow}, u^{\downarrow}) \in (\to^*)^{\mathcal{M}_{\mathcal{R}}}$ hold. By definition of $\mathcal{M}_{\mathcal{R}}$ and Proposition 6, we have $s^{\downarrow} \to^*_{\mathcal{R}} u$ and $t^{\downarrow} \to^*_{\mathcal{R}} u$, i.e., $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is $\overline{\mathcal{R}}$-feasible. ◀

▶ **Proposition 26.** *Let $\mathcal{R}$ be a CTRS. If* $\mathsf{CCP}(\mathcal{R})$ *contains a non-joinable CCP, then $\mathcal{R}$ is not (locally) confluent.*

Proof. If $\langle s, t\rangle \Leftarrow D \in \mathsf{CCP}(\mathcal{R})$ is not joinable, then, according to Definition 14, there is a substitution σ such that $\sigma(D)$ holds and $\sigma(s) \downarrow_{\mathcal{R}} \sigma(t)$ does not hold. Note that $s = \theta(\ell[r']_p)$ and $t = \theta(r')$ for some rules $\ell \to r \Leftarrow C$ and $\ell' \to r' \Leftarrow C'$, $p \in \mathcal{P}os_{\mathcal{F}}(\ell)$, *mgu* θ of $\ell|_p$ and ℓ', and $D = \theta(C), \theta(C')$. Since $\sigma(D) = \sigma(\theta(C)), \sigma(\theta(C'))$ holds, both $\sigma(\theta(C))$ and $\sigma(\theta(C'))$ hold as well (disregarding the join, oriented, or semiequational semantics for $\mathcal{R}$). Thus, $\sigma(\theta(\ell)) \to \sigma(s)$ using α' and $\sigma(\theta(\ell)) \to \sigma(t)$ using α. Since $\sigma(s)$ and $\sigma(t)$ are not joinable, $\mathcal{R}$ is not locally confluent. Hence, it is not confluent. ◀

▶ **Corollary 32.** *A terminating normal CTRS is confluent if all its critical pairs are joinable overlays.*

Proof. By [7, Theorem 4], a terminating conditional join CTRS whose critical pairs are all joinable overlays is confluent. Now, considering Remark 31, the statement of the corollary follows. ◀

On the Expressive Equivalence of TPTL in the Pointwise and Continuous Semantics

Raveendra Holla ✉
Citrix Systems India Pvt. Ltd., Bangalore, India

Nabarun Deka ✉
Indian Institute of Science Bangalore, India

Deepak D'Souza ✉
Indian Institute of Science Bangalore, India

Abstract

We consider a first-order logic with linear constraints interpreted in a pointwise and continuous manner over timed words. We show that the two interpretations of this logic coincide in terms of expressiveness, via an effective transformation of sentences from one logic to the other. As a consequence it follows that the pointwise and continuous semantics of the logic TPTL with the since operator also coincide. Along the way we exhibit a useful normal form for sentences in these logics.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Real-Time Logics, First-Order Logics

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.45

1 Introduction

Several real-time logics proposed in the literature have been interpreted over timed behaviours in two natural ways which have come to be known as the "pointwise" and "continuous" interpretations. In the pointwise semantics, formulas may be asserted only at points where an action occurs (the so-called "action points"), while in the continuous semantics formulas may be asserted at arbitrary time points. To illustrate these semantics, consider the popular timed temporal logic Metric Temporal Logic (MTL) [10, 1, 12], which extends the U operator of classical LTL with an interval index, to allow formulas of the form $\theta U_I \eta$ which says that, with respect to the current time point, there is a future time point where η is satisfied and which lies at a distance that falls within the interval I, and at all time points in between θ is satisfied. Consider a timed word σ depicted in Fig. 1 below, in which the first action is an a at time 2, followed subsequently by only b's. Then the MTL formula $\Diamond(\Diamond_{[1,1]}a)$ is satisfied in σ in the continuous semantics, but not in the pointwise semantics since there is no action point at time 1.

The Timed Temporal Logic (TPTL) of Alur and Henzinger [2, 3] is a well-known timed temporal logic for specifying real-time behaviors. The logic is interpreted over timed words and extends classical LTL with the "freeze" quantifier $x.\theta$ which binds x to the value of the current time point, along with the ability to constrain these time points using linear constraints of the form $x \sim y + c$. For example the formula $x.(\Diamond y.(a \wedge y = x + 2))$ says that with respect to the current time point, an action a occurs exactly two time units later. Then the TPTL formula $\Diamond x.\Diamond y.(a \wedge y = x + 1)$ is satisfied in σ in the continuous semantics, but not in the pointwise semantics, since there is no action point at time 1. It is not difficult to

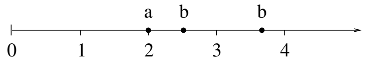


Figure 1 Timed word σ.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 45; pp. 45:1–45:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

see that for a typical timed temporal logic the continuous semantics is at least as expressive as the pointwise one, since one can ask for a time point to be an action point by asserting $\bigvee_{a\in\Sigma} a$ at each quantified time point.

There have been several results in the literature which show that for the logic MTL and its variants the continuous semantics is in fact strictly more expressive than the pointwise one. In particular, the logics MTL over infinite words [4, 5] and finite words [13]; MTL_S (MTL with the "since" operator S) and MTL_{S_I} (MTL with the S_I operator), over both infinite and finite words [7]; are all strictly more expressive in the continuous semantics than their pointwise counterparts. In addition, Ho et al [9] show the strict inclusion of MTL in a first-order logic $FO(<, +1)$ over finite words in the pointwise semantics, in contrast to their equivalence in the continuous semantics [11].

In this paper we show, somewhat surprisingly, that the logic TPTL_S (TPTL with the "since" operator) has the same expressive power in both the pointwise and continuous semantics. We do this by considering a natural first-order logic $\mathrm{FO}(<, +\mathbb{Q})$ interpreted over timed words, which is similar in flavour to TPTL_S. The logic allows atomic predicates of the form $a(x)$ which says that an a-event occurs at time point x, and constraints of the form $x < y + c$. The interpretation of the quantifier $\exists x$ depends on the pointwise or continuous semantics: in the pointwise it is interpreted as "there exists an action point x", while in the continuous semantics it is interpreted as "there exists a time point x." The main technical result in this paper is that the expressiveness of $\mathrm{FO}(<, +\mathbb{Q})$ in the pointwise and continuous interpretations coincide.

The main proof idea is to show that we can go from an arbitrary sentence in the continuous version of $\mathrm{FO}(<, +\mathbb{Q})$ to an equivalent sentence in $\mathrm{FO}(<, +\mathbb{Q})$ which uses only "active" quantifiers, where each $\exists x$ is qualified by an assertion that x is an action point. A sentence in which all quantifiers are active, is clearly equivalent to a pointwise formula.

The technique in this paper builds on the work reported in a preprint [6] by giving a more transparent argument for a key step of the proof via Thm. 2. In the next few sections of this paper we focus on the result for $\mathrm{FO}(<, +\mathbb{Q})$, and turn to its application to TPTL in Sec. 7.

2 Preliminaries

We begin with some preliminary definitions. Let $\mathbb{R}_{\geq 0}$ denote the set of non-negative real numbers, $\mathbb{Q}$ the set of rational numbers, and $\mathbb{N}$ the set of non-negative integers. We use the standard notation to represent intervals, which are convex subsets of $\mathbb{R}$. For example $[1, \infty)$ denotes the set $\{t \in \mathbb{R} \mid 1 \leq t\}$.

For an alphabet A we denote by A^ω the set of infinite words over A. Let Σ be a finite alphabet of actions, which we fix for the rest of this paper. An (infinite) timed word σ over Σ is an element of $(\Sigma \times \mathbb{R}_{\geq 0})^\omega$ of the form $(a_0, t_0)(a_1, t_1) \cdots$, satisfying the conditions that: for each $i \in \mathbb{N}$, $t_i < t_{i+1}$ (*monotonicity*), and for each $t \in \mathbb{R}_{\geq 0}$ there exists an $i \in \mathbb{N}$ such that $t < t_i$ (*progressiveness*). For convenience, we will also assume in this paper that $t_0 = 0$, so that the timed word begins with an action at time 0. We will sometimes represent the timed word σ above as a pair (α, τ), where $\alpha = a_0 a_1 \cdots$ and $\tau = t_0 t_1 \cdots$. Thus $\alpha(i)$ and $\tau(i)$ denote the the action and the time stamp respectively, in σ at position i. We write $T\Sigma^\omega$ to denote the set of all timed words over Σ.

We now introduce the linear constraints we use in this paper, and some notation for manipulating them. We assume a supply of variables $\mathit{Var} = \{x, y, \ldots\}$ which we will use in constraints as well as later in our logics. We use restricted linear constraints of the form $x \sim y + c$ or $x \sim c$, where x and y are variables in Var, $\sim$ is one of the relations $\{<, \leq, =, \geq, >\}$, and c is in $\mathbb{Q}$. We call these constraints *simple constraints*. In general, by (an unqualified) "constraint" we will mean a conjunction of simple constraints.

$0 \leq x \leq 1$	$0 \leq x \leq 1$	$0 \leq y - 1$	$1 \leq y \leq 2.2$
$x + 1 \leq y \leq x + 1.2$	$y - 1.2 \leq x \leq y - 1$	$y - 1.2 \leq 1$	
$0 \leq y$	$0 \leq y$	$y - 1.2 \leq y - 1$	
		$0 \leq y$	
(a)	(b)	(c)	(d)

Figure 2 Illustrating steps of the Fourier-Motzkin elimination method.

An *assignment* for variables is a map $\mathbb{I} : Var \to \mathbb{R}_{\geq 0}$. For $t \in \mathbb{R}_{\geq 0}$ and $x \in Var$ we will use $\mathbb{I}[t/x]$ to represent the assignment which sends x to t, and agrees with $\mathbb{I}$ on all other variables. When we are interested in a finite set of variables $\{x_1, \ldots, x_n\}$ we will write $[t_1/x_1, \ldots, t_n/x_n]$ to represent an assignment that maps each x_i to t_i. For an assignment $\mathbb{I}$ and a constraint δ, we write $\mathbb{I} \models \delta$ to mean that the constraint δ is satisfied in the assignment $\mathbb{I}$, and defined in the expected way.

As a final piece of notation, we will make use of the well-known Fourier-Motzkin method for eliminating variables from constraints. Given a conjunction π of simple constraints, some of which contain a variable x, the technique gives us a conjunction π' of simple constriants not containing x, such that the formula $\exists x \pi$ is logically equivalent to π' (assuming a standard first-order logic interpreted over rationals or reals). When we are interested in the domain of $\mathbb{R}_{\geq 0}$ (like in this paper), we assume that π implicitly contains the constraint $z \geq 0$ for each variable z in π. As an illustration of the method, consider the conjunction π of the constraints in Fig. 2(a). To eliminate x from π, we first rewrite the constraints involving x as lower and upper bounds on x, as shown in the first two constraints in Fig. 2(b), and carry forward the constraints not involving x (like the third one). Next we relate each lower bound of x to each upper bound of x, as shown in the first three constraints of Fig. 2(c), while carrying forward the constraints not involving x. Finally, we simplify the constraints by dropping looser bounds and removing redundant constraints like $0 \leq 1$, to obtain the constraint π' in Fig. 2(d). The constraint π' can be seen to be logically equivalent to $\exists x \pi$.

We will use the notation $\mathrm{FME}_x(\pi)$ to refer to the constraint π'. We refer the reader to [14] for the details of this technique.

3 The $\mathrm{FO}(<, +\mathbb{Q})$ logic

We now define our first order logic with simple constraints $\mathrm{FO}(<, +\mathbb{Q})$, which is interpreted over timed words over the alphabet Σ. The formulas of $\mathrm{FO}(<, +\mathbb{Q})$ are given by:

$$\varphi ::= a(x) \mid g \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x \varphi,$$

where $a \in \Sigma$, $x \in Var$, and g is a simple constraint.

We first define the *continuous* semantics for $\mathrm{FO}(<, +\mathbb{Q})$. Let φ be a formula in $\mathrm{FO}(<, +\mathbb{Q})$. Let $\sigma = (\alpha, \tau)$ be a timed word over Σ, and let $\mathbb{I}$ be an assignment for variables. Then the satisfaction relation $\sigma, \mathbb{I} \models_c \varphi$ (read "σ satisfies φ with the assignment $\mathbb{I}$ in the continuous semantics") is inductively defined as:

$$\begin{array}{lll}
\sigma, \mathbb{I} \models_c a(x) & \text{iff} & \exists i : \tau(i) = \mathbb{I}(x) \text{ and } \alpha(i) = a \\
\sigma, \mathbb{I} \models_c g & \text{iff} & \mathbb{I} \models g \\
\sigma, \mathbb{I} \models_c \neg\nu & \text{iff} & \sigma, \mathbb{I} \not\models_c \nu \\
\sigma, \mathbb{I} \models_c \nu \vee \psi & \text{iff} & \sigma, \mathbb{I} \models_c \nu \text{ or } \sigma, \mathbb{I} \models_c \psi \\
\sigma, \mathbb{I} \models_c \exists x \nu & \text{iff} & \exists t \in \mathbb{R}_{\geq 0} \text{ such that } \sigma, \mathbb{I}[t/x] \models_c \nu.
\end{array}$$

The derived connectives $\wedge$, $\supset$ (implies), $\forall$, etc are defined in the standard way. A variable x is said to occur *free* in a formula φ if there is an occurrence of x that is not within the scope of any $\exists x$ quantifier in φ. A *sentence* is a formula in which there are no free occurrences of variables. The satisfaction of a sentence in a timed word is independent of an assignment for variables. The timed language defined by an $\mathrm{FO}(<, +\mathbb{Q})$ sentence φ in the continuous semantics is given by $L^c(\varphi) = \{\sigma \in T\Sigma^\omega \mid \sigma \models_c \varphi\}$.

We can similarly define the *pointwise* semantics of the logic $\mathrm{FO}(<, +\mathbb{Q})$, where the quantification is over action points in the timed word. The satisfaction relation $\sigma, \mathbb{I} \models_{pw} \varphi$ is defined as above, except for the $\exists$ clause which is interpreted as follows:

$$\sigma, \mathbb{I} \models_{pw} \exists x \nu \quad \text{iff} \quad \exists t \in \mathbb{R}_{\geq 0} \text{ such that } t = \tau(i) \text{ for some } i \in \mathbb{N}, \text{ and } \sigma, \mathbb{I}[t/x] \models_{pw} \nu.$$

The timed language defined by a sentence φ in the pointwise semantics is given by $L^{pw}(\varphi) = \{\sigma \in T\Sigma^\omega \mid \sigma \models_{pw} \varphi\}$.

The formulas of the logic $\mathrm{FO}(<, +\mathbb{Q})$ can be seen to be essentially that of a first-order logic over the signature $(0, \{+c\}_{c \in \mathbb{Q}}, <, \{a\}_{a \in \Sigma})$, where each $+c$ is a function that adds the rational c to its argument, and each $a \in \Sigma$ is a unary predicate. The logic is interpreted over timed words in the expected way, with the domain being $\mathbb{R}_{\geq 0}$ in the continuous interpretation, and the set of action points in the pointwise interpretation. In the sequel we will write $\mathrm{FO}^c(<, +\mathbb{Q})$ to denote the logic with the continuous interpretation, and similarly $\mathrm{FO}^{pw}(<, +\mathbb{Q})$ to denote the pointwise interpretation.

4 A normal form for FO sentences

In this section we exhibit a normal form for $\mathrm{FO}^c(<, +\mathbb{Q})$ sentences which will be useful in our proofs. We begin with a normal form for formulas of the form $\exists x \varphi$. An $\mathrm{FO}^c(<, +\mathbb{Q})$ formula is said to be in $\exists$-*normal form* if it is of the form $\exists x(a(x) \wedge \pi(x) \wedge \nu)$, where $a \in \Sigma$, $\pi(x)$ is a conjunction of simple constraints each containing x, and ν is a conjunction of formulas of the form ψ or $\neg\psi$, where each ψ is again in $\exists$-normal form. In addition, we allow any of the components $a(x)$ and ν to be absent. We say a formula is in *negated* $\exists$-normal form if it is the negation of a formula in $\exists$-normal form. Fig. 3 depicts a sentence which is a boolean combination of $\exists$-normal form sentences.

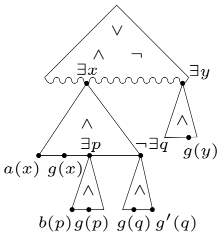


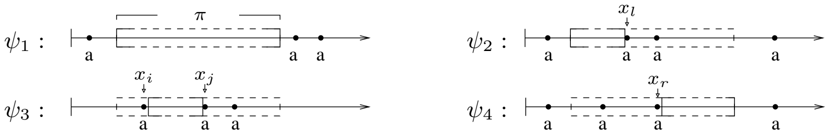
Figure 3 Boolean combination of sentences in $\exists$-normal form.

▶ **Theorem 1.** *Any* $\mathrm{FO}^c(<, +\mathbb{Q})$ *sentence can be equivalently expressed as a boolean combination of sentences in* $\exists$*-normal form.*

Proof. Let φ be an $\mathrm{FO}^c(<, +\mathbb{Q})$ sentence. Since φ is a sentence it must be a boolean combination of sentences of the form $\exists x \varphi'$. We transform φ into an equivalent sentence which is a boolean combination of sentences in $\exists$-normal form, by repeatedly transforming the formula tree of φ as follows:

1. In every subtree rooted at a $\exists$-node, in the formula tree of φ, push every $\neg$ operator downwards over $\vee$, $\wedge$, and all the way through g nodes, till it reaches a $\exists$-node or an a (action) node. After this step, the subtree below every $\exists$ node contains only conjunctions and disjunctions of a, $\neg a$, $\exists$, $\neg\exists$, and g nodes.
2. For convenience, in the next couple of steps, we will consider $\neg\exists$ as a single composite node in the formula tree. Pull all the $\vee$'s upwards in the resulting formula tree for φ, using the following identities: $\nu_1 \wedge (\nu_2 \vee \nu_3) \equiv (\nu_1 \wedge \nu_2) \vee (\nu_1 \wedge \nu_3)$, $\exists x(\nu_1 \vee \nu_2) \equiv (\exists x \nu_1) \vee (\exists x \nu_2)$ and $\neg\exists x(\nu_1 \vee \nu_2) \equiv (\neg\exists x \nu_1) \wedge (\neg\exists x \nu_2)$. It is not difficult to see that using these identities we obtain a formula φ' in which each $\exists$-node or $\neg\exists$-node contains only conjunctions of a, $\neg a$, $\exists$, $\neg\exists$, and g nodes.
3. In this step we pull up from a subtree rooted at an $\exists x$ node, all nodes which are independent of x, namely nodes of the form $b(y)$, $\neg b(y)$ (with $y \neq x$), and g where g does not contain x. This is done by recursively applying following equivalences starting from the lower most $\exists x$ or $\neg\exists x$ nodes: $\exists x(b(y) \wedge \nu) \equiv b(y) \wedge \exists x(\nu)$ and $\neg\exists x(b(y) \wedge \nu) \equiv \neg b(y) \vee \neg\exists x(\nu)$. We can use similar equivalences for $\neg b(y)$ and g to pull them up the tree. Finally, we move all the newly generated $\vee$'s up the tree using Step 2.
   After this step, the subtrees rooted at each $\exists x$ node is a conjunction of $a(x)$, $\neg a(x)$, $\exists$, $\neg\exists$ and $g(x)$ nodes.
4. We now update the formula tree with the following equivalences: $a(x) \wedge b(x) \equiv \perp$ and $a(x) \wedge \neg b(x) \equiv a(x)$, where $a, b \in \Sigma$ with $a \neq b$. After this step, the only action-related nodes in a subtree rooted at a $\exists x$ node are a single action node $a(x)$ or a conjunction of negation of actions of the form $\bigwedge_{a \in X} \neg a(x)$ for some $X \subseteq \Sigma$.
5. We can now replace formulas of the form $\bigwedge_{a \in A} \neg a(x)$ by a disjunction of formulas which contain at most one action, as described below. We then pull up the newly generated $\vee$ nodes up the tree using Step 2. After this step, the subtree rooted at every $\exists x$ node contains only conjunctions of $a(x)$, $\exists$, $\neg\exists$ and $g(x)$ nodes. We can collect the $g(x)$ nodes together to get a single conjunction of constraints $\pi(x)$. Thus finally each subtree rooted at $\exists$ node is in $\exists$-normal form.

To see how we can replace formulas of the form $\bigwedge_{a \in A} \neg a(x)$ by a disjunction of formulas in $\exists$-normal form, consider a formula ψ of the form $\exists x(\bigwedge_{a \in A} \neg a(x) \wedge \pi(x) \wedge \nu)$. Let $A(x)$ be shorthand for the formula $\bigvee_{a \in A} a(x)$. Then, $\psi = \exists x(\neg A(x) \wedge \pi(x) \wedge \nu)$. We claim that $\varphi \equiv \psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4$ with each ψ_i defined as follows. The figure below illustrates these cases. We view the constraint $\pi(x)$ as an interval determined by the values assigned to the variables other than x.



If an A-action does not occur anywhere in the interval $\pi(x)$, then φ is satisfied if ν is satisfied for any x in $\pi(x)$:

$$\psi_1 = \quad \neg\exists x(A(x) \wedge \pi(x)) \wedge \exists x(\pi(x) \ \wedge \ \nu)$$

If there are one or more actions $A(x)$ in $\pi(x)$ then φ is satisfied iff ν is satisfied before the first occurrence of $A(x)$, or between any two consecutive occurrences of $A(x)$, or after the last occurrence of $A(x)$, in $\pi(x)$. These three cases are formulated as follows:

$$
\begin{aligned}
\psi_2 = \;& \exists x_l(A(x_l) \wedge \pi(x_l) \wedge \neg\exists x'(A(x') \wedge \pi(x') \wedge x' < x_l) \wedge \exists x(\pi(x) \wedge x < x_l \;\wedge\; \nu)) \\
\psi_3 = \;& \exists x_i(A(x_i) \wedge \pi(x_i) \wedge \exists x_j(A(x_j) \wedge \pi(x_j) \wedge \neg\exists x'(A(x') \wedge \pi(x') \wedge x_i < x' < x_j) \\
& \qquad\qquad\qquad\qquad \wedge\exists x(\pi(x) \wedge x_i < x \wedge x < x_j \;\wedge\; \nu))) \\
\psi_4 = \;& \exists x_r(A(x_r) \wedge \pi(x_r) \wedge \neg\exists x'(A(x') \wedge \pi(x') \wedge x_r < x') \wedge \exists x(\pi(x) \wedge x_r < x \;\wedge\; \nu))
\end{aligned}
$$

This completes the proof of the normal form transformation. ◀

5 Equivalence of FO^c and FO^{pw} semantics

In this section our aim is to show that the logics FO$^{pw}(<, +\mathbb{Q})$ and FO$^c(<, +\mathbb{Q})$ are expressively equivalent. It is easy to translate an FO$^{pw}(<, +\mathbb{Q})$ sentence φ to an equivalent FO$^c(<, +\mathbb{Q})$ sentence by simply replacing every $\exists x\varphi'$ subformula, by $\exists x(\bigvee_{a\in\Sigma} a(x) \wedge \varphi'')$, where φ'' is obtained by similarly replacing $\exists$-subformulas in φ'.

In the converse direction, let us call an FO$^c(<, +\mathbb{Q})$ formula φ *actively quantified* (or simply *active*) if every $\exists$-subformula is of the form $\exists x(a(x) \wedge \varphi')$ for some action $a \in \Sigma$ and formula φ'. Then, an active FO$^c(<, +\mathbb{Q})$ formula clearly defines the same language of timed words, regardless of the semantics being pointwise or continuous. Hence, our aim in the rest of this section is to show how we can go from an arbitrary formula in FO$^c(<, +\mathbb{Q})$ to an equivalent active formula.

5.1 Proof Idea

A formula in the continuous semantics has the obvious advantage of being able to associate any value in $\mathbb{R}_{\geq 0}$ to its variables, whereas an actively quantified variable can refer only to the action points in a timed word. Consider the sentence below where x is passively quantified:

$$\exists x(0 \leq x \wedge x \leq 1 \wedge \exists y(a(y) \wedge x + 1 \leq y \wedge y \leq x + 1.2)). \tag{1}$$

In the continuous semantics this is essentially asking for an a action sometime in the interval $[1, 2, 2]$. However, if we interpret this sentence in the pointwise semantics we get a strictly stronger requirement of there being an action point in the interval $[0, 1]$ from which we have an a-action at a distance of 1 to 1.2. In our approach we transform the given sentence to an equivalent active sentence (all in the continuous semantics), which we can do as follows. The given sentence is equivalent to the sentence (2) below by simple logical manipulation. Then we apply Fourier-Motzkin elimination in the $\exists x$ part of (2) to get the sentence (3), which is now an equivalent active formula.

$$\exists y(a(y) \wedge \exists x(0 \leq x \wedge x \leq 1 \wedge x + 1 \leq y \wedge y \leq x + 1.2)) \tag{2}$$

$$\exists y(a(y) \wedge 1 \leq y \wedge y \leq 2.2) \tag{3}$$

As another example, consider the language of all timed words over a and b, where for every b in the interval [1,2], there is an a in [0,1] exactly one time unit earlier. This can be written easily in FO^c as:

$$\neg\exists x((\neg a(x) \wedge 0 \leq x \wedge x \leq 1 \wedge \exists y(b(y) \wedge y = x + 1)). \tag{4}$$

But if we interpret this sentence in the pointwise semantics it does not describe the same property. The given sentence is not in $\exists$-normal form and the normalization yields a disjunction of four formulas $\psi_1, \psi_2, \psi_3, \psi_4$, where x is the only variable which is passively

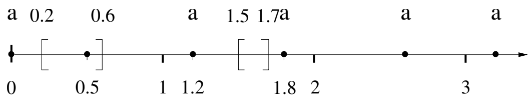


Figure 4 Timed word σ satisfying formula (5).

quantified. If we can eliminate x from ψ_1, ψ_2, ψ_3, and ψ_4 without introducing any new passively quantified variables, the disjunction of these actively quantified formulas recognizes the required language. The subformula involving x in each ψ_i looks like $\exists x(\pi(x) \wedge \exists y(b(y) \wedge y = x+1))$. This can be equivalently written as $\exists y(b(y) \wedge \exists x(\pi(x) \wedge y = x+1))$. We can now use Fourier-Motzkin elimination to eliminate x from $\pi(x) \wedge y = x+1$ to obtain a constraint $\pi'(y)$ on y. The above formula can now be expressed equivalently as $\exists y(b(y) \wedge \pi'(y))$, thereby eliminating the passively quantified variable x.

As a final example, consider the following modified version of formula (2):

$$\exists x(0 \leq x \wedge x \leq 1 \wedge \neg\exists y(a(y) \wedge x+1 \leq y \wedge y \leq x+1.2)). \tag{5}$$

For the sake of simplicity, let us consider a scenario where a is the only action in Σ. The above formula is true iff there is a point x in $[0,1]$ such that there is no action point a in the interval $[x+1, x+1.2]$. To eliminate the passively quantified variable x from this formula, we will consider the interval $\pi_1 = w_1 < x+1 < w_2 \wedge w_3 < x+1.2 < w_4$ and find an assignment to the variables w_1–w_4 such that for every point x which lies in the intersection of the intervals $[0,1]$ and π_1, there is no action point a in the interval $[x+1, x+1.2]$. Consider as an example the timed word σ shown in Fig. 4. This timed word satisfies the formula (5) with the valuation $x = 0.5$ since there is no action point a in the interval $[1.5, 1.7]$. Now consider the following assignment, $w_1 = 1.2$ is the first action point in σ before 1.5 and $w_2 = 1.8$ is the first action point in σ after 1.5. Similarly, $w_3 = 1.2$ is the first action point in σ before 1.7 and $w_4 = 1.8$ is the first action point in σ after 1.7. With this assignment, we get the interval $\pi_1 = 1.2 < x+1 < 1.8 \wedge 1.2 < x+1.2 < 1.8$ which is equivalent to $0.2 < x < 0.6$. It is easy to see that for any x in the intersection of the intervals $[0,1]$ and π_1 there is no action point a in the interval $[x+1, x+1.2]$. Hence, the timed word σ will also satisfy the equivalent formula:

$$\begin{aligned}
&\exists w_1 \exists w_2 \exists w_3 \exists w_4 (a(w_1) \wedge a(w_2) \wedge a(w_3) \wedge a(w_4) \\
&\qquad \wedge \forall x((0 \leq x \leq 1 \wedge \pi_1) \supset \neg\exists y(a(y) \wedge x+1 \leq y \wedge y \leq x+1.2))) &(6)\\
\equiv\ &\exists w_1 \exists w_2 \exists w_3 \exists w_4 (a(w_1) \wedge a(w_2) \wedge a(w_3) \wedge a(w_4) \\
&\qquad \wedge \neg\exists x(0 \leq x \leq 1 \wedge \pi_1 \wedge \exists y(a(y) \wedge x+1 \leq y \wedge y \leq x+1.2))) &(7)\\
\equiv\ &\exists w_1 \exists w_2 \exists w_3 \exists w_4 (a(w_1) \wedge a(w_2) \wedge a(w_3) \wedge a(w_4) \\
&\qquad \wedge \neg\exists y(a(y) \wedge \exists x(0 \leq x \leq 1 \wedge \pi_1 \wedge x+1 \leq y \wedge y \leq x+1.2))). &(8)
\end{aligned}$$

We get (7) from (6) using the equivalence $\forall x\varphi \equiv \neg\exists x\neg\varphi$. Finally, we eliminate x from the innermost part of formula (8) using Fourier-Motzkin elimination to get a formula which is completely actively quantified. We will prove in the later part of this section that it is always possible to identify the interval π_1 using the syntax of $\mathrm{FO}^c(<, +\mathbb{Q})$.

5.2 Equivalence Proof

We begin with some definitions. The *quantifier depth* of an FO formula is the maximum nesting depth of quantifiers in the formula. Given a formula $\varphi(x)$ (where x is free in φ) and a timed word σ, we will call an assignment $\mathbb{I}$ an *x-restricted assignment* for φ w.r.t. the timed word σ iff for every atomic subformula $y \sim x + c$ of φ, $\mathbb{I}(x) + c$ is not an action point of σ, and for every atomic subformula $y \sim x - c$ of φ, $\mathbb{I}(x) - c$ is not an action point of σ. Finally, consider a formula φ of the form $\exists x(\pi(x) \wedge \psi)$, where π is a conjunction of simple constraints and ψ is a formula in $\exists$-normal form. We say that a timed word σ *strongly satisfies* φ if there exists an x-restricted assignment $\mathbb{I}$ for ψ w.r.t. σ such that $\sigma, \mathbb{I} \models \pi(x) \wedge \psi$.

Also, observe that for any formula in $\exists$-normal form, we can replace all the equality atomic formulas, i.e. atomic formulas of the form $x = y + c$, with the equivalent formula $x \leq y + c \wedge x \geq y + c$. Hence, we will first remove all the equalities in our formula using this replacement. Furthermore, for simplicity, we will assume that the set of actions Σ is a singleton set i.e. $\Sigma = \{a\}$. This idea can be generalised to a finite set of actions Σ. Now we have the following theorem:

▶ **Lemma 2.** *Consider a formula of the form $\varphi = \exists x(\pi(x) \wedge \psi)$ where π is a conjunction of simple constraints and ψ is an actively quantified formula in $\exists$-normal form or negated $\exists$-normal form. Then, we can construct a formula θ which is a disjunction of formulas of the form:*

$$\exists w_1 \exists w_2 \cdots \exists w_n (\bigwedge_{i=1}^{i=n} a(w_i) \wedge \exists x(\pi(x) \wedge \pi_1(x, w_1, \ldots, w_n)) \wedge \forall x((\pi(x) \wedge \pi_1(x, w_1, \ldots, w_n)) \supset \psi)),$$

such that for any timed word σ which strongly satisfies φ, we also have $\sigma \models \theta$.

We will prove this theorem in the next section. In the rest of this section we see how to use it to prove the equivalence of the pointwise and continuous semantics of FO$(<, +\mathbb{Q})$.

▶ **Theorem 3.** *Given any formula φ in $\exists$-normal form of the form $\exists x(\pi(x) \wedge \psi)$ where ψ is actively quantified, we can construct an equivalent formula ν which is a disjunction of formulas that are either actively quantified formulas in $\exists$-normal form or conjunctions of simple constraints that do not contain x. In other words, we can eliminate the passive variable x from φ.*

Proof. We prove this by induction on the quantifier depth of the formula ψ.

Base case (quantifier depth 0): In this case our formula φ is of the form $\varphi = \exists x \pi(x)$. We can use Fourier Motzkin elimination here to eliminate the variable x and get a formula which is a conjunction of simple constraints.

Inductive case: Now assume that we have proved the theorem for quantifier depth up to n, and consider the case when ψ has quantifier depth $n + 1$. We consider three different cases for the form of ψ.

Case 1a (Single positive conjunct): $\psi = \exists y(a(y) \wedge \delta \wedge \psi')$. In this case, we have

$$\begin{aligned} \varphi &= \exists x(\pi(x) \wedge \exists y(a(y) \wedge \delta \wedge \psi')) \\ &\equiv \exists y(a(y) \wedge \exists x(\pi(x) \wedge \delta \wedge \psi')). \end{aligned}$$

By the induction hypothesis the formula $\exists x((\pi(x) \wedge \delta) \wedge \psi')$ can be expressed as an equivalent formula $\nu = \nu_1 \vee \cdots \vee \nu_k$ with each ν_i actively quantified in $\exists$-normal form or negated $\exists$-normal form. Hence, we get

$$\begin{aligned} \varphi &\equiv \exists y(a(y) \wedge \nu) \\ &\equiv \exists y(a(y) \wedge (\nu_1 \vee \cdots \vee \nu_k)) \\ &\equiv \exists y(a(y) \wedge \nu_1) \vee \cdots \vee \exists y(a(y) \wedge \nu_k), \end{aligned}$$

which completes the proof for this case.

Case 1b (Single negative conjunct): $\psi = \neg\exists y(a(y) \wedge \delta \wedge \psi')$. In this case, we construct ν as follows.

We apply Lemma 2 on the formula φ to get a disjunct θ such that for any timed word σ if σ strongly satisfies φ, we have that $\sigma \models \theta$.

Each formula in the disjunct θ is of the form:

$$\begin{aligned}\theta_i = \exists w_1 \cdots \exists w_n (& \bigwedge_{i=1}^{n} a(w_i) \\ & \wedge \exists x(\pi(x) \wedge \pi_i(x, w_1, \ldots, w_n)) \\ & \wedge \forall x((\pi(x) \wedge \pi_i(x, w_1, \ldots, w_n)) \supset \psi)).\end{aligned}$$

In the conjunct $\exists x(\pi(x) \wedge \pi_i(x, w_1, \ldots, w_n))$ we can eliminate the passive variable x using Fourier-Motzkin elimination. The third conjunct is $\forall x((\pi(x) \wedge \pi_i(x, w_1, \ldots, w_n)) \supset \psi)$. Substituting $\psi = \neg\exists y(a(y) \wedge \delta \wedge \psi')$, we get

$$\begin{aligned} & \forall x((\pi \wedge \pi_i) \supset \neg\exists y(a(y) \wedge \delta \wedge \psi')) \\ \equiv & \neg\exists x(\pi \wedge \pi_i \wedge \exists y(a(y) \wedge \delta \wedge \psi')) \\ \equiv & \neg\exists y(a(y) \wedge \exists x(\pi \wedge \pi_i \wedge \delta \wedge \psi')).\end{aligned}$$

Now rewrite the interval $\pi \wedge \pi_i \wedge \delta$ as π' and apply the induction hypothesis on the formula $\exists x(\pi' \wedge \psi')$ to replace it with an equivalent disjunct $\nu' = \nu'_1 \vee \nu'_2 \vee \cdots \vee \nu'_k$ where each disjunct is actively quantified. Hence, after these manipulations, θ is a disjunction of actively quantified formulas in $\exists$-normal form and we have that for any timed word σ, if σ strongly satisfies φ, we have $\sigma \models \theta$.

Now we have to take care of the corner cases where $\sigma \models \varphi$ but σ does not strongly satisfy φ. For this, we do the following:

For each atomic formula of the form $x \sim v + c$, where v is some variable other than x, appearing in the formula ψ, define a formula

$$\mu = \exists w(a(w) \wedge \pi(w + c) \wedge \psi[(w + c)/x]).$$

Define $\mathcal{D}_1$ to be the disjunction of all such μ's. Similarly, for each atomic formula of the form $x + c \sim v$ appearing in the formula ψ, define a formula

$$\mu = \exists x(a(w) \wedge \pi(w - c) \wedge \psi[(w - c)/x]).$$

Define $\mathcal{D}_2$ to be the disjunction of all such μ's. Finally, define

$$\nu = \mathcal{D}_1 \vee \mathcal{D}_2 \vee \theta.$$

Observe that all the formulas in the disjuncts $\mathcal{D}_1$ and $\mathcal{D}_2$ are actively quantified formulas in $\exists$-normal form. Hence, ν is a disjunction of actively quantified formula in $\exists$ normal form.

Now, we need to show that for any timed word σ, we have $\sigma \models \varphi \iff \sigma \models \nu$. Assume that $\sigma \models \varphi$, i.e. there is an assignment $x = t_1$ such that $\sigma, [t_1/x] \models \pi \wedge \psi$. Then, this can happen in three ways:

1. For some atomic formula $x \sim v + c$ occurring in ψ, $t_1 - c$ is an action point of σ. In this case, $\sigma \models \mathcal{D}_1$.
2. For some atomic formula $x + c \sim v$ occurring in ψ, $t_1 + c$ is an action point of σ. In this case, $\sigma \models \mathcal{D}_2$.
3. $x = t_1$ is an x-restricted assignment for φ w.r.t σ. In this case, from Lemma 2, we get that $\sigma \models \theta$.

The other direction is straightforward.

Case 2 (Multiple Conjuncts): Now we consider the case where ψ has more than one conjunct. For simplicity, let ψ be the conjunct $\psi_1 \wedge \psi_2$. Our original formula is thus $\varphi = \exists x(\pi(x) \wedge \psi_1 \wedge \psi_2)$. We apply Lemma 2 to the formulas $\varphi_1 = \exists x(\pi(x) \wedge \psi_1)$ and $\varphi_2 = \exists x(\pi(x) \wedge \psi_2)$ to get two formulas

$$\theta'_1 = \bigvee_{j=1}^{k} \theta_{1j} \text{ and } \theta'_2 = \bigvee_{j=1}^{m} \theta'_{2j},$$

where each θ'_{ij} is of the form:

$$\begin{aligned}\theta'_{ij} = \exists w_1 \cdots \exists w_n (& \bigwedge_{i=1}^{p} a(w_i) \\ & \wedge \exists x(\pi(x) \wedge \pi_{ij}(x, w_1, \ldots, w_n)) \\ & \wedge \forall x((\pi(x) \wedge \pi_{ij}(x, w_1, \ldots, w_n)) \supset \psi_i).\end{aligned}$$

Here note that p might be different for each (i, j) . Now for each $i \in \{1, 2, \ldots, k\}$ and $j \in \{1, 2, \ldots, m\}$ we construct a formula θ_{ij} below:

$$\begin{aligned}\exists w_1 \cdots \exists w_l \exists w'_1 \cdots \exists w'_n (& \bigwedge_{i=1}^{l} a(w_i) \bigwedge_{i=1}^{n} a(w'_i) \\ & \wedge \exists x(\pi(x) \wedge \pi_{1i}(x, w_1, \ldots, w_l) \wedge \pi_{2j}(x, w'_1, \ldots, w'_n)) \\ & \wedge \forall x((\pi(x) \wedge \pi_{1i}(x, w_1, \ldots, w_l) \wedge \pi_{2j}(x, w'_1, \ldots, w'_n)) \supset (\psi_1 \wedge \psi_2)),\end{aligned}$$

and define $\theta = \bigvee_{i=1}^{k} \bigvee_{j=1}^{m} \theta_{ij}$.

We will now show that if σ strongly satisfies φ, then $\sigma \models \theta$. Suppose σ strongly satisfies φ, then σ strongly satisfies both φ_1 and φ_2. Now by the single conjunct case, we know that there exists i_0 and j_0 such that $\sigma \models \theta'_{1i_0}$ and $\sigma \models \theta'_{2j_0}$. We will now show that $\sigma \models \theta_{i_0 j_0}$. Since $\sigma \models \theta_{1i_0}$, we have an assignment $\mathcal{W}$ such that

$$\begin{aligned}\sigma, \mathcal{W} \models (& \bigwedge_{i=1}^{n} a(w_i) \\ & \wedge \exists x(\pi(x) \wedge \pi_{1i_0}(x, w_1, \ldots, w_n)) \\ & \wedge \forall x((\pi(x) \wedge \pi_{1i_0}(x, w_1, \ldots, w_n)) \supset \psi_1).\end{aligned}$$

Similarly, since $\sigma \models \theta_{2j_0}$, we have an assignment $\mathcal{W}'$ such that

$$\begin{aligned}\sigma, \mathcal{W}' \models (& \bigwedge_{i=1}^{n} a(w_i) \\ & \wedge \exists x(\pi(x) \wedge \pi_{2j_0}(x, w_1, \ldots, w_n)) \\ & \wedge \forall x((\pi(x) \wedge \pi_{2j_0}(x, w_1, \ldots, w_n)) \supset \psi_2).\end{aligned}$$

Now, it is easy to see that

$$\begin{aligned}\sigma, \mathcal{W}, \mathcal{W}' \models (& \bigwedge_{i=1}^{l} a(w_i) \bigwedge_{i=1}^{n} a(w'_i) \\ & \wedge \exists x(\pi(x) \wedge \pi_{1i_0}(x, w_1, \ldots, w_l) \wedge \pi_{2j_0}(x, w'_1, \ldots, w'_n)) \\ & \wedge \forall x((\pi(x) \wedge \pi_{1i_0}(x, w_1, \ldots, w_l) \wedge \pi_{2j_0}(x, w'_1, \ldots, w'_n)) \supset (\psi_1 \wedge \psi_2)).\end{aligned}$$

Hence, $\sigma \models \theta_{i_0 j_0}$ and $\sigma \models \theta$.

Recall that we still have to eliminate x from the $\theta'_{ij}s$ which are of the form:

$$\begin{aligned}
\exists w_1 \cdots \exists w_l \exists w'_1 \cdots \exists w'_n (&\bigwedge_{i=1}^{l} a(w_i) \bigwedge_{i=1}^{n} a(w'_i) \\
&\wedge \exists x(\pi(x) \wedge \pi_{1i}(x, w_1, \dots, w_l) \wedge \pi_{2j}(x, w'_1, \dots, w'_n)) \qquad (9)\\
&\wedge \forall x((\pi(x) \wedge \pi_{1i}(x, w_1, \dots, w_l) \wedge \pi_{2j}(x, w'_1, \dots, w'_n)) \supset (\psi_1 \wedge \psi_2)). \qquad (10)
\end{aligned}$$

We can eliminate x from (9) using Fourier-Motzkin elimination. As for (10), we do the following manipulations (we drop the free variables w_i's to remove some clutter):

$$\begin{aligned}
&\forall x((\pi \wedge \pi_{1i} \wedge \pi_{2j}) \supset (\psi_1 \wedge \psi_2)) \\
\equiv &\forall x((\pi \wedge \pi_{1i} \wedge \pi_{2j}) \supset \psi_1) \wedge \forall x((\pi \wedge \pi_{1i} \wedge \pi_{2j}) \supset \psi_2) \\
\equiv &\neg \exists x(\pi \wedge \pi_{1i} \wedge \pi_{2j} \wedge \neg \psi_1) \wedge \neg \exists x(\pi \wedge \pi_{1i} \wedge \pi_{2j} \wedge \neg \psi_2). \qquad (11)
\end{aligned}$$

Now (11) has two formulas containing the passive variable x, but both of them can be handled by the single conjunct case (*Case 1* above). Hence, we have successfully eliminated x for the multiple conjunct case.

To handle the cases where $\sigma \models \varphi$ but σ does not strongly satisfy φ, we can do a construction similar to the one done for *Case 1b*.

This completes the proof of the theorem. ◀

▶ **Theorem 4.** *Every* FO^c *sentence can be transformed to an equivalent active sentence.*

Proof. Any FO^c sentence φ can be written as a boolean combination of sentences in $\exists$-normal form. Since translation is intact across the boolean operations, it is sufficient if we can eliminate all the passive variables from formulas in $\exists$-normal form. We will now show that given a formula φ in $\exists$-normal form which has passive variables, we can construct an equivalent formula which is a disjunction of actively quantified formulas in $\exists$-normal form or negated $\exists$-normal form. Observe that if we can show this, the theorem follows immediately.

We will show this by induction on the number of passive variables n of the formula φ. We assume all quantified variables of φ are distinct, by renaming them if necessary.

Base Case: Suppose φ has 1 passive variable. Let T be the formula tree of φ, and let N be the corresponding node in T. The subtree roooted at N is thus of the form $\exists x(\pi(x) \wedge \psi)$ where ψ is actively quantified. Now we can use Theorem 3 to replace the subtree at N with a disjunction of actively quantified formulas in $\exists$-normal form. We then pull up the disjuncts to the top of the tree to get a disjunction of actively quantified formulas in $\exists$-normal form.

Inductive Case: Now suppose we have shown our hypothesis for formulas with up to n passive variables. Let φ be a formula in $\exists$-normal form with $n+1$ passive variables. Let T be the formula tree of φ. We call a subtree of T rooted at node N a *maximal passive subtree* if N is a passive node and has no passive nodes as ancestors.

Now, if T has more than one maximal passive subtree, then the corresponding formula for each subtree is a formula in $\exists$-normal form with at most n passive variables. By our induction hypothesis, we can replace any such subtree with a disjunction of actively quantified formulas. We then pull out the disjunction to the top of T to get a finite disjunction of formulas in $\exists$-normal form, each of which has at most n passive variables. We can then apply the induction hypothesis on each of these to replace them with equivalent actively quantified formulas.

If T has exactly one maximal passive subtree, the formula corresponding to this subtree is of the form $\exists x(\pi(x) \wedge \psi)$ where $\psi = \psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_m$ where each ψ_i is in $\exists$-normal form or negated $\exists$-normal form. Observe that since $\exists x(\pi(x) \wedge \psi)$ has $n+1$ passive variables, each ψ_i can have atmost n passive variables. If ψ_i is in negated $\exists$-normal form, it is of the form $\neg\mu_i$ where μ_i is in $\exists$-normal form and has atmost n passive variables. We can now apply the induction hypothesis on μ_i and replace it with a disjunction of acitvely quantified formulas, i.e.

$$\mu_i = \nu_{i1} \vee \nu_{i2} \vee \cdots \vee \nu_{ik}.$$

Therefore,

$$\psi_i = \neg\mu_i = \neg\nu_{i1} \wedge \neg\nu_{i2} \wedge \cdots \wedge \neg\nu_{ik}.$$

We first replace all such ψ_i's. Now the remaining ψ_i's are in $\exists$-normal form and each have at most n passive variables. We apply the induction hypothesis on each of these and replace them with disjunctions of actively quantified formulas in $\exists$-normal form. Hence, we get

$$\exists x(\pi(x) \wedge \psi) = \exists x(\pi(x) \wedge \bigwedge_{i=1}^{k} \bigvee_{j=1}^{l_i} \nu_{ij}).$$

Let $S := \{1, 2, \ldots, l_1\} \times \{1, 2, \ldots, l_2\} \times \cdots \times \{1, 2, \ldots, l_k\}$. Then

$$\exists x(\pi(x) \wedge \psi) = \bigvee_{(a_1,\ldots,a_p)\in S} \exists x(\pi(x) \wedge \bigwedge_{i=1}^{p} \nu_{ia_i}). \tag{12}$$

Now each disjunct in (12) is in $\exists$-normal form with exactly one passive variable. We apply our induction hypothesis here and replace each of them with a disjunction of actively quantified formulas in $\exists$-normal form. We then pull up the disjuncts to the top of the tree T to get a disjunction of actively quantified formulas in $\exists$-normal form.

Thus we have shown that for any formula in $\exists$-normal form, we have an equivalent actively quantified formula, and we are done. ◀

To summarise:

▶ **Theorem 5.** *The logics* FO^c *and* FO^{pw} *are expressively equivalent. Moreover there is an effective procedure to translate a sentence in one logic to an equivalent one in the other.* ◀

6 Proof of Lemma 2

Here we will give a proof of Lemma 2 in a simplified setting. A more detailed exposition is given in the Appendix. For now, we will consider a simplified setting where our timed words are two way infinite timed words i.e. the timeline is not $[0, \infty)$ but instead $(-\infty, \infty)$ and for any timed word σ and a point t_0 there are action points before and after the point t_0. Recall the definition of x-restricted assignment and strong satisfaction from Section 5. Now in this setting, we have the following lemma which says that given a timed word σ, an interval $\delta(x, y)$ which is an interval for y determined by x (for example: $x + 1 \leq y \leq x + 2$), and a value $x = t_1$ for x, we can construct an interval $\pi_1(x)$ (using some other variables) such that for any $x = t_1'$ in the interval $\pi_1(x)$, the set of action points $a(y)$ in the intervals $\delta(t_1, y)$ and $\delta(t_1', y)$ are exactly the same, i.e. for any point in the interval π_1 the set of action points in the interval δ are preserved.

▶ **Lemma 6** (Preservation of action points). *Consider a formula $\varphi = a(y) \wedge \delta(x, y)$ where x, y are free variables, a is an action and δ is a conjunction of simple constraints. Then, we can construct an interval $\pi_1(x, w_1, \ldots, w_n)$ where the w_i's are newly introduced free variables, such that given any timed word σ and a x-restricted assignment $\mathbb{I}$ for φ w.r.t. σ (let $\mathbb{I}(x) = t_1$), there exists an assignment $\mathcal{W} = [b_1/w_1, \ldots, b_n/w_n]$ such that the b_i's are action points of σ and, for any t_1' which satisfies $[t_1'/x], \mathcal{W} \models \pi_1(x)$, and for any t_0, we have*

$$\sigma, [t_1/x, t_0/y] \models \varphi \iff \sigma, [t_1'/x, t_0/y] \models \varphi.$$

Furthermore, $[t_1/x], \mathcal{W} \models \pi_1(x, w_1, \ldots, w_n)$.

Proof. We can think of δ as an interval for y that is determined by the value of x. This lemma says that for any t_1' in the interval π_1, the set of action points $a(y)$ in the interval $\delta([t_1'/x])$ is the same as that of the interval $\delta([t_1/x])$.

We construct the interval π_1 using the right and left boundaries of δ. The right boundary of δ will be of the form $y \sim x \pm c$ where $\sim \in \{<, \leq\}$ and the left boundary will be of the form $y \sim x \pm c$ where $\sim \in \{>, \geq\}$. W.l.o.g, take the left boundary to be $y \geq x - c_1$ and the right boundary to be $y \leq x + c_2$. Define four new variables w_1, w_2, w_3 and w_4 and define π_1 as

$$\pi_1 := w_1 < x - c_1 < w_2 \wedge w3 < x + c_2 < w_4.$$

Now take any timed word σ and a x-restricted assignment $\mathbb{I}$ for φ w.r.t. σ, with $\mathbb{I}(x) = t_1$. The assignment $\mathcal{W}$ is defined as follows:

1. Let b_1 be the first action point of σ that precedes the point $t_1 - c_1$. Assign $w_1 := b_1$
2. Let b_2 be the first action point of σ that succeeds the point $t_1 - c_1$. Assign $w_2 := b_2$
3. Let b_3 be the first action point of σ that precedes the point $t_1 + c_2$. Assign $w_3 := b_3$
4. Let b_4 be the first action point of σ that succeeds the point $t_1 + c_2$. Assign $w_4 := b_4$

With this assignment, it is easy to see that for any $x = t_1'$ in the interval π_1 i.e. $[x/t_1'], \mathcal{W} \models \pi_1$, the set of action points in the intervals $\delta([x/t_1])$ and $\delta([x/t_1'])$ is the same. Also, $[x/t_1], \mathcal{W} \models \pi_1(x, w_1, \ldots, w_n)$. And hence, the lemma follows. ◀

With this lemma we will prove the version of Lemma 2 in our simplified setting, which we state below:

▶ **Lemma 7.** *Consider a formula of the form $\varphi = \exists x(\pi(x) \wedge \psi)$ where π is a conjunction of simple constraints and ψ is an actively quantified formula in $\exists$-normal form or negated $\exists$-normal form. Then, we can construct an equivalent formula μ which is of the form:*

$$\exists w_1 \cdots \exists w_n (\bigwedge_{i=1}^{n} a(w_i) \wedge \exists x(\pi(x) \wedge \pi_1(x, w_1, \ldots, w_n)) \wedge \forall x((\pi(x) \wedge \pi_1(x, w_1, \ldots, w_n)) \supset \psi))$$

such that for any timed word σ which strongly satisfies φ, we also have $\sigma \models \mu$

Proof. We will prove the lemma by inducting on the quantifier depth of the formula ψ.

Base Case (quantifier depth = 1): In this case, $\psi = \exists y(a(y) \wedge \delta)$ or $\psi = \neg\exists y(a(y) \wedge \delta)$. We apply the preservation of action points lemma on the formula $a(y) \wedge \delta$ to get the interval $\pi_1(w_1, w_2, \ldots, w_n)$ for which action points in the interval δ are preserved. We define μ as follows:

$$\exists w_1, w_2, \ldots, w_n (\bigwedge_{i=1}^{n} a(w_i) \wedge \exists x(\pi(x) \wedge \pi_1(x, w_1, \ldots, w_n)) \wedge \forall x((\pi(x) \wedge \pi_1(x, w_1, \ldots, w_n)) \supset \psi)).$$

Now pick any timed word σ such that σ strongly satisfies φ. The preservation of action points lemma will give us a valuation $\mathcal{W}$ for the variables $w_1, \ldots, w_n$. It is easy to see that $\sigma \models \mu$ with the valuation $\mathcal{W}$

Inductive Case (quantifier depth = n): In this case, $\psi = \exists y(a(y) \wedge \delta \wedge \nu)$ or $\psi = \neg\exists y(a(y) \wedge \delta \wedge \nu)$ where ν is of quantifier depth $n-1$. To construct θ, we first look at the formula $\varphi' = \exists x(\pi(x) \wedge \psi')$ where $\psi' = \nu$ if $\psi = \exists y(a(y) \wedge \delta \wedge \nu)$ and $\psi' = \neg\nu$ otherwise. Applying induction hypothesis to this formula, we get the formula

$$\mu' = \exists w'_1, w'_2, \ldots, w'_m(\bigwedge_{i=1}^{m} a(w'_i) \wedge \exists x(\pi(x) \wedge \pi'_1(x, w_1, \ldots, w_m)) \\ \wedge \forall x((\pi(x) \wedge \pi'_1(x, w_1, \ldots, w_m)) \supset \psi')).$$

We apply the preservation of action points lemma on the formula $a(y) \wedge \delta$ to get the interval $\pi_1(w_1, w_2, \ldots, w_n)$ for which action points in the interval δ are preserved. We construct μ using π_1 and μ' as follows:

$$\mu = \exists w_1, \ldots, w_n \exists w'_1, w'_2, \ldots, w'_m(\bigwedge_{j=1}^{n} a(w_j) \bigwedge_{i=1}^{m} a(w'_i) \\ \wedge \exists x(\pi(x) \wedge \pi_1(x, w_1, \ldots, w_n) \wedge \pi'_1(x, w'_1, \ldots, w'_m)) \\ \wedge \forall x((\pi(x) \wedge \pi_1(x, w_1, \ldots, w_n) \wedge \pi'_1(x, w'_1, \ldots, w'_m)) \supset \psi)).$$

Pick any timed word σ such that σ strongly satisfies φ i.e. there is a x-restricted assigment $\mathbb{I}(x) = t_1$ such that $\sigma, [t_1/x] \models \pi(x) \wedge \psi$. We need to show that for any $x = t'_1$ in the interval $\pi \wedge \pi_1 \wedge \pi'_1$ we have ψ. Now from the preservation of action points lemma, we know that for any $x = t'_1$ in the interval π_1, the action points in the interval δ are preserved. Also, by the induction, the interval π'_1 ensures that for any $x = t'_1$ in the interval π'_1, $\sigma, [t'_1/x] \models \psi'$. Hence, for any $x = t'_1$ in the intersection of these intervals, i.e. the interval $\pi \wedge \pi_1 \wedge \pi'_1$ we have $\sigma, [t'_1/x] \models \psi$. Hence, we get that $\sigma \models \mu$. ◀

7 Equivalence of FO and TPTL$_S$

We can now argue that the logic TPTL$_S$ is expressively equivalent in its continuous and pointwise semantics. We recall that the formulas of TPTL$_S$ [3, 5] over the alphabet Σ, are defined as follows: $\theta ::= a \mid g \mid \neg\theta \mid \theta \vee \theta \mid \theta U \theta \mid \theta S \theta \mid x.\theta$ where $a \in \Sigma$, x is a variable in *Var*, and g is a simple constraint.

In the pointwise semantics, for a TPTL$_S$ formula θ, timed word $\sigma = (\alpha, \tau)$ over Σ, $i \in \mathbb{N}$, and an assignment for variables $\mathbb{I}$, we define the satisfaction relation $\sigma, i, \mathbb{I} \models_{pw} \theta$, as:

$$\begin{array}{lll} \sigma, i, \mathbb{I} \models_{pw} a & \text{iff} & \alpha(i) = a \\ \sigma, i, \mathbb{I} \models_{pw} g & \text{iff} & \mathbb{I} \models g \\ \sigma, i, \mathbb{I} \models_{pw} \theta U \eta & \text{iff} & \exists k : i < k \text{ s.t. } \sigma, k, \mathbb{I} \models_{pw} \eta \text{ and } \forall j : i < j < k : \sigma, j, \mathbb{I} \models_{pw} \theta \\ \sigma, i, \mathbb{I} \models_{pw} \theta S \eta & \text{iff} & \exists k : 0 \leq k < i \text{ s.t. } \sigma, k, \mathbb{I} \models_{pw} \eta \text{ and } \forall j : k < j < i, \sigma, j, \mathbb{I} \models_{pw} \theta \\ \sigma, i, \mathbb{I} \models_{pw} x.\theta & \text{iff} & \sigma, i, \mathbb{I}[\tau(i)/x] \models_{pw} \theta, \end{array}$$

with boolean operators handled in the expected way. For a "closed" TPTL$_S$ formula θ, in which every occurrence of x is within the scope of a freeze quantifier "$x.$", we set the language defined by it to be $L^{pw}(\theta) = \{\sigma \in T\Sigma^\omega \mid \sigma, 0 \models_{pw} \theta\}$.

In the continuous semantics, for a TPTL$_S$ formula θ, a timed word $\sigma = (\alpha, \tau)$ over Σ, $t \in \mathbb{R}_{\geq 0}$, and an assignment $\mathbb{I}$, the satisfaction relation $\sigma, t, \mathbb{I} \models_c \theta$ is defined similarly, except that:

$$\begin{array}{lll}
\sigma, t, \mathbb{I} \models_c a & \text{iff} & \exists i : \alpha(i) = a \text{ and } \tau(i) = t \\
\sigma, t, \mathbb{I} \models_c \theta U \eta & \text{iff} & \exists t' : t < t' \text{ s.t. } \sigma, t', \mathbb{I} \models_c \eta \text{ and } \forall t'' : t < t'' < t', \sigma, t'', \mathbb{I} \models_c \theta \\
\sigma, t, \mathbb{I} \models_c \theta S \eta & \text{iff} & \exists t' : 0 \leq t' < t \text{ s.t. } \sigma, t', \mathbb{I} \models_c \eta \text{ and } \forall t'' : t' < t'' < t, \sigma, t'', \mathbb{I} \models_c \theta \\
\sigma, t, \mathbb{I} \models_c x.\theta & \text{iff} & \sigma, t, \mathbb{I}[t/x] \models_c \theta.
\end{array}$$

We use the standard syntactic abbreviations of $\Diamond, \diamondminus, \Box$ and $\boxminus$ defined in a reflexive manner: $\Diamond\theta = \theta \vee (\top U \theta)$, $\diamondminus\theta = \theta \vee (\top S \theta)$, $\Box\theta = \neg\Diamond\neg\theta$, and $\boxminus\theta = \neg\diamondminus\neg\theta$. We note that in TPTL_S it is possible to express the U and S operators using $\Diamond$ and $\diamondminus$ operators, in both the continuous and pointwise semantics. For instance, $\theta U \eta \ \equiv x.\Diamond y.(\eta \wedge x < y \wedge \boxminus z.(x < z \wedge z < y \Rightarrow \theta))$. Hence we concentrate only on these operators in the translations below.

▶ **Theorem 8.** *The logics* TPTL_S^c *and* TPTL_S^{pw} *are expressively equivalent.* ◀

Proof. We first show that we can translate a formula in TPTL_S to an equivalent one in $\mathrm{FO}(<, +\mathbb{Q})$, and vice-versa. For a closed formula θ in TPTL_S we show how to give a formula *tptl-fo*(θ) in $\mathrm{FO}(<, +\mathbb{Q})$, which has a single free variable z, such that for any timed word σ, $\sigma, t \models_c \theta$ if and only if $\sigma, [t/z] \models_c$ *tptl-fo*(θ) (and similarly in pointwise semantics). The translation *tptl-fo* is defined inductively on the structure of θ as follows:

$$\begin{array}{lll}
\textit{tptl-fo}(a) & = & a(z) \\
\textit{tptl-fo}(g) & = & g \\
\textit{tptl-fo}(\Diamond\theta') & = & \exists x(x \geq z \wedge \textit{tptl-fo}(\theta')[x/z]) \\
\textit{tptl-fo}(\diamondminus\theta') & = & \exists x(x \leq z \wedge \textit{tptl-fo}(\theta')[x/z]) \\
\textit{tptl-fo}(x.\theta') & = & (\textit{tptl-fo}(\theta'))[z/x]
\end{array}$$

with boolean operators handled in the expected manner. We can now translate a closed formula θ in TPTL_S to an $\mathrm{FO}(<, +\mathbb{Q})$ sentence $\varphi = \exists z(z = 0 \wedge \textit{tptl-fo}(\theta))$, with $L^c(\theta) = L^c(\varphi)$.

In the other direction, we translate an $\mathrm{FO}(<, +\mathbb{Q})$ sentence φ, to an equivalent closed TPTL_S formula *fo-tptl*(φ) as follows. We first transform φ into its normal form as given in Thm 1. The translation *fo-tptl* is defined inductively in a similar manner to *tptl-fo* above, with $\exists$-subformulas being translated via the rule:

$$\textit{fo-tptl}(\exists x(a(x) \wedge \pi(x) \wedge \nu)) \quad = \quad \Diamond x.(a \wedge \pi(x) \wedge \textit{fo-tptl}(\nu)) \vee \diamondminus x.(a \wedge \pi(x) \wedge \textit{fo-tptl}(\nu)).$$

It is easy to see that $L^c(\varphi) = L^c(\textit{fo-tptl}(\varphi))$.

We can now prove the non-trivial direction of the theorem. Consider a closed formula θ of TPTL_S^c. We go over to an equivalent $\mathrm{FO}(<, +\mathbb{Q})$ formula $\varphi = \textit{tptl-fo}(\theta)$, obtain an equivalent pointwise $\mathrm{FO}(<, +\mathbb{Q})$ formula φ' using Thm. 5, and finally obtain an equivalent pointwise TPTL_S formula θ', with $L^c(\theta) = L^{pw}(\theta')$. ◀

8 Conclusion

In this paper we have shown the expressive equivalence of, and in fact given an effective translation between, the pointwise and continuous versions of two natural logics $\mathrm{FO}(<, +\mathbb{Q})$ and TPTL_S over timed words. Some interesting directions include addressing a similar question for TPTL (i.e. without the since operator). One may be able to use an argument like [8] to say that TPTL is as expressive as TPTL_S in the pointwise setting. Another interesting question is about first-order logic with Presburger constraints in general. Here it appears that we can express strong properties in the continuous interpretation which seem difficult to express in the pointwise setting.

References

1 Rajeev Alur, Tomas Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. In *10th ACM Symposium on Principles of Distributed Computing*, pages 139–152, 1991.

2 Rajeev Alur and Thomas A. Henzinger. A Really Temporal Logic. In *IEEE Symposium on Foundations of Computer Science*, pages 164–169, 1989.

3 Rajeev Alur and Thomas A. Henzinger. A Really Temporal Logic. *J. ACM*, 41(1):181–204, 1994. doi:10.1145/174644.174651.

4 Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. In *Foundations of Software Technology and Theoretical Computer Science*, 2005.

5 Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. *Inf. Comput.*, 208(2):97–116, 2010. doi:10.1016/j.ic.2009.10.004.

6 Deepak D'Souza, Raveendra Holla, and Raj Mohan M. Equivalence of the pointwise and continuous semantics of first order logic with linear constraints, 2010. URL: https://www.csa.iisc.ac.in/~deepakd/papers/tptl.pdf.

7 Deepak D'Souza and Pavithra Prabhakar. On the expressiveness of MTL in the pointwise and continuous semantics. In *Formal Methods Letters, Software Tools for Technology Transfer, Vol. 9, No. 1*, pages 1–4. Springer, 2007.

8 Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 163–173, 1980.

9 Hsi-Ming Ho, Joël Ouaknine, and James Worrell. On the Expressiveness and Monitoring of Metric Temporal Logic. *Log. Methods Comput. Sci.*, 15(2), 2019. doi:10.23638/LMCS-15(2:13)2019.

10 Ron Koymans. Specifying Real-Time Properties with Metric Temporal Logic. In *Real Time Systems*, pages 2(4):255–299, 1990.

11 Joël Ouaknine, Alexander Rabinovich, and James Worrell. Time-Bounded Verification. In Mario Bravetti and Gianluigi Zavattaro, editors, *20th International Conference on Concurreny Theory*, volume 5710 of *Lecture Notes in Computer Science*, pages 496–510. Springer, 2009. doi:10.1007/978-3-642-04081-8_33.

12 Joël Ouaknine and James Worrell. On the Decidability of Metric Temporal Logic. In *20th Annual IEEE Symposium on Logic in Computer Science*, pages 188–197, 2005.

13 Pavithra Prabhakar and Deepak D'Souza. On the expressiveness of MTL with past operators. In *4th Intl. Conference on Formal Modelling and Analysis of Timed Systems*, pages 322–336, 2006. doi:10.1007/11867340_23.

14 Alexander Schrijver. *Theory of Linear and Integer Programming*, pages 155–157. John Wiley & Sons, 2000.

A Proof of Lemma 2

▶ **Lemma 9** (Preservation of action points). *Consider a formula $\varphi = a(y) \wedge \delta(x, y)$ where x, y are free variables, a is an action and δ is a conjunction of simple linear constraints. Then, we can construct a finite number of intervals $\pi_1(x, w_1, w_2, \dots, w_n), \dots, \pi_m(x, w_1, w_2, \dots, w_k)$ where the w_i's are newly introduced free variables, such that given any timed word σ and an x-restricted assignment, $x = t_1$ for φ w.r.t σ, there exists an $i \in \{1, \dots, m\}$ and an assignment $\mathcal{W} = \{w_1 = b_1, w_2 = b_2, \dots, w_n = b_n\}$ with the b_i's being action points of the timed word σ, such that for any t_1' which satisifes $x = t_1', \mathcal{W} \models \pi_i(x)$, and for any valuation $y = t_0$ for y, we have*

$$\sigma, x = t_1, y = t_0 \models \varphi \iff \sigma, x = t_1', y = t_0 \models \varphi.$$

Furthermore, $x = t_1, \mathcal{W} \models \pi_i$

Proof. Observe that the constraints in δ are defining an interval for y. Based on the boundaries of this interval, we can have four different cases:

1. Both the left end and right end of δ are of the form $y \sim x + c$
2. the left end of δ is of the form $y \sim x - c$ and the right end of δ is of the form $y \sim x + c$
3. Both the left and right end of δ are of the form $y \sim x - c$
4. the left end of δ is of the form $y \sim x + c$ and the right end is of the form $y \sim x - c$

We now provide the constructions for each of the cases.

Case 1: Let the left end of δ be $y \sim x + c_1$ and the right end be $y \sim x + c_2$. In this case, we have only 1 interval π_1. We introduce 4 new variables w_1, w_2, w_3, w_4 and construct $\pi_1 = w_1 < x + c_1 < w_2 \wedge w_3 < x + c_2 < w_4$

Case 2: Let the left end of δ be $y \sim x - c_1$ and the right end be $y \sim x + c_2$. In this case, we have two intervals:

$$\pi_1 = w_1 < x - c_1 < w_2 \wedge w_3 < x + c_2 < w4$$
$$\pi_2 = 0 < x < c_1 \wedge w_3 < x + c_2 < w4$$

Case 3: Let the left end of δ be $y \sim x - c_1$ and the right end be $y \sim x - c_2$. We have 4 intervals:

$$\pi_1 = w_1 < x - c_1 < w_2 \wedge w_3 < x - c_2 < w_4$$
$$\pi_2 = 0 < x < c_1 \wedge w_3 < x - c_2 < w_4$$
$$\pi_3 = w_1 < x - c_1 < w_2 \wedge 0 < x < c_2$$
$$\pi_4 = 0 < x < c_1 \wedge 0 < x < c_2$$

Case 4: Let the left end of δ be $y \sim x + c_1$ and the right end be $y \sim x - c_2$. We have 2 intervals:

$$\pi_1 = w_1 < x + c_1 < w_2 \wedge w_3 < x - c_2 < w_4$$
$$\pi_2 = w_1 < x + c_1 < w_2 \wedge 0 < x < c_2$$

We will prove the theorem for case 2. The proof is similar for the others. Pick any timed word σ and an x-restricted valuation $x = t_1$ for φ w.r.t σ. W.l.o.g, let the left end of δ be $y \geq x - c_1$ and the right end be $y \leq x + c_2$. We can have two cases:

Case 1 : $t_1 - c_1 > 0$. For this case, pick the interval $\pi_1 = w_1 < x - c_1 < w_2 \wedge w_3 < x + c_2 < w4$. we define the assignment $\mathcal{W}$ as follows:

- Set w_1 to be the first action point of σ that is less than $t_1 - c_1$
- Set w_2 to be the first action point of σ that is greater than $t_1 - c_1$
- Set w_3 to be the first action point of σ that is less than $t_1 + c_2$
- Set w_4 to be the first action point of σ that is greater than $t_1 + c_2$

Now we need to show that for any assignment $y = t_0$ for y and any t_1' such that $x = t_1', \mathcal{W} \models \pi_1(x)$, we have $\sigma, x = t_1, y = t_0 \models \varphi \iff \sigma, x = t_1', y = t_0 \models \varphi$. Pick any assignment $y = t_0$ for y and any t_1' such that $x = t_1', \mathcal{W} \models \pi_1(x)$. Assume $\sigma, x = t_1, y = t_0 \models \varphi$. Suppose that $\sigma, x = t_1', y = t_0 \not\models \varphi$, then,

$$\sigma, x = t_1', y = t_0 \not\models \delta$$
$$\implies \sigma, x = t_1', y = t_0 \not\models y \geq x - c_1 \text{ OR } \sigma, x = t_1', y = t_0 \not\models y \leq x + c_2$$

WLOG suppose $\sigma, x = t_1', y = t_0 \not\models y \geq x - c_1$ or in other words, $\sigma, x = t_1', y = t_0 \models y < x - c_1$. Now observe that $t_1', \mathcal{W} \models \pi_1(x)$ and hence, $t_1' - c_1 < w_2$. Furthermore,

$$\sigma, x = t_1, y = t_0 \models \varphi \implies \sigma, x = t_1, y = t_0 \models \delta$$
$$\implies \sigma, x = t_1, y = t_0 \models y \geq x - c_1 \implies t_0 \geq t_1 - c_1$$

Hence, we get the inequality, $t_1 - c_1 \leq t_0 < t_1' - c_1 < w_2$. This is saying that t_0 is an action point of σ which lies in between $t_1 - c_1$ and w_2. This directly contradicts our assignment of w_2. Hence, our assumption is wrong. Therefore, $\sigma, x = t_1', y = t_0 \models y \geq x - c_1$. An exactly similar proof will show the other direction.

Case 2: $t_1 - c_1 < 0$. For this case, pick the interval $\pi_2 = 0 < x < c_1 \wedge w_3 < x + c_2 < w_4$. We use the same assignment $\mathcal{W}$ as in case 1, but restricted to the variables w_3 and w_4. One can argue similar to above to show that the lemma holds in this case also.

Hence, we get our Lemma. ◀

▶ **Theorem 10.** *Consider a formula of the form $\varphi = \exists x(\pi(x) \wedge \psi)$ where π is a conjunction of simple constraints and ψ is an actively quantified formula in $\exists$-normal form or the negation of an actively quantified formula in $\exists$-normal form. Then, we can construct a formula θ which is a disjunction of formulas of the form:*

$$\theta_i = \exists w_1 \exists w_2 \cdots \exists w_n (\bigwedge_{i=1}^{n} a(w_i) \wedge \exists x(\pi(x) \wedge \pi_i(x, w_1, \ldots, w_n)) \\ \wedge \forall x((\pi(x) \wedge \pi_i(x, w_1, \ldots, w_n)) \supset \psi))$$

such that for any timed word σ which strongly satisfies φ we also have $\sigma \models \theta$.

Proof. Assume that σ strongly satisfies φ with an x-restricted assignment $x = t_1$ for φ w.r.t. σ. Observe that if $\sigma \models \theta$, then there is an i_0 such that $\sigma \models \theta_{i_0}$. We will not only show that $\sigma \models \theta_{i_0}$ for some i_0, but we will also further show that $\sigma \models \theta_{i_0}$ with an assignment $\mathcal{W}$ for the w_i's such that $x = t_1, \mathcal{W} \models \pi_{i_0}(x, w_1, \ldots, w_n)$ and the choice of the assignment $\mathcal{W}$ depends only on σ and the assignment $x = t_1$ for x.

We prove this by inducting on the quantifier depth of the formula ψ.

Base Case: Quantifier depth = 1
In this case, we have $\varphi = \exists x(\pi(x) \wedge \psi)$ where

$$\psi = \exists y(a(y) \wedge \delta) \text{ OR } \psi = \neg\exists y(a(y) \wedge \delta)$$

Now we apply the preservation of action points lemma on the formula $a(y) \wedge \delta$ to get a finite number of intervals $\pi_1, \pi_2, \ldots, \pi_m$. For each such interval, we construct a formula

$$\theta_i = \exists w_1 \exists w_2 \cdots \exists w_n (\bigwedge_{i=1}^{n} a(w_i) \wedge \exists x(\pi(x) \wedge \pi_i(x, w_1, \ldots, w_n)) \\ \wedge \forall x((\pi(x) \wedge \pi_i(x, w_1, \ldots, w_n)) \supset \psi)).$$

We define $\theta = \bigvee_{i=1}^{m} \theta_i$.

To prove the lemma, let $\psi = \exists y(a(y) \wedge \delta)$. Now pick any timed word σ such that σ strongly satisfies φ, i.e. there is an x- restricted assignment $x = t_1$ for ψ w.r.t σ, such that $\sigma, x = t_1 \models \pi(x) \wedge \exists y(a(y) \wedge \delta)$. Applying our preservation of action points lemma on the formula $a(y) \wedge \delta$, we get an $i_0 \in \{1, 2, \ldots, m\}$. We will now show that $\sigma \models \theta_{i_0}$. To show this, we have to show that there is an assignment $\mathcal{W} = \{w_1 = b_1, \ldots, w_n = b_n\}$ such that

$$\sigma, \mathcal{W} \models (\bigwedge_{i=1}^{i=n} a(w_i) \wedge \exists x(\pi(x) \wedge \pi_{i_0}(x, w_1, \ldots, w_n)) \\ \wedge \forall x((\pi(x) \wedge \pi_{i_0}(x, w_1, \ldots, w_n)) \supset \psi)).$$

The preservation of action points lemma gives us an assignment $\mathcal{W}$ which preserves the action points in the interval δ. We will use the same assignment here. $\sigma, \mathcal{W} \models a(w_j)$ holds for each j by the preservation of action points lemma. By our hypothesis, $\sigma, x = t_1 \models \pi(x)$ and by the preservation of action points lemma we also have $\sigma, x = t_1, \mathcal{W} \models \pi_{i_0}$. Hence, we also have $\sigma, \mathcal{W} \models \exists x(\pi \wedge \pi_{i_0})$. Also, observe that from the preservation of action points lemma, we get that the choice of $\mathcal{W}$ depends only on σ and the assignment $x = t_1$.

Now all that remains is to show that $\sigma, \mathcal{W} \models \forall x((\pi(x) \wedge \pi_{i_0}(x, w_1, \ldots, w_n)) \supset \psi)$. Pick any assignment $x = t_1'$ of the variable x such that $\sigma, x = t_1', \mathcal{W} \models \pi(x) \wedge \pi_{i_0}(x, w_1, \ldots, w_n)$. We need to show that $\sigma, x = t_1' \models \psi$. By our assumption, we have that $\sigma, x = t_1 \models \pi(x) \wedge \exists y(a(y) \wedge \delta)$. Hence, there is an assigment $y = t_0$ such that $\sigma, x = t_1, y = t_0 \models a(y) \wedge \delta$. By the preservation of action points lemma, we get that $\sigma, x = t_1', y = t_0 \models a(y) \wedge \delta$. Hence, we get that $\sigma, x = t_1' \models \psi$.

Hence, we get that $\sigma \models \theta_{i_0}$ and hence $\sigma \models \theta$. Observe that, in the process we also showed that $\sigma \models \theta_{i_0}$ with an assignment $\mathcal{W}$ such that $x = t_1, \mathcal{W} \models \pi_{i_0}$ and the choice of $\mathcal{W}$ depended only on σ and the assignment $x = t_1$ for x.

One can similarly argue that the lemma holds if $\psi = \neg\exists y(a(y) \wedge \delta)$.

Inductive Step: Let us assume that we have proved the lemma for quantifier depth of $1, 2, \ldots, n-1$. Now, consider a formula $\varphi = \exists x(\pi(x) \wedge \psi)$ where ψ is of quantifier depth n. We will split this proof into two cases based on whether ψ is a formula in $\exists$ normal form, or negation of a formula $\exists$ normal form. We will show the lemma for the first case, and the second case can be shown similarly.

Case 1 (Positive Case) : $\psi = \exists y(a(y) \wedge \delta \wedge \psi')$.

We will first prove it assuming ψ' is of the form ν or $\neg\nu$ where ν is in $\exists$ normal form of quantifier depth $n-1$. Later, we will give the construction for when ψ' is a conjunction of formulas of the form ν or $\neg\nu$, where ν is in $\exists$ Normal form, and the proof of the lemma is similar.

Now consider the formula $\varphi' = \exists x(\pi(x) \wedge \psi')$. By our induction hypothesis, there exists a formula $\theta' = \bigvee_{j=1}^{k} \theta_j'$ such that the lemma holds, and each θ_j' is of the form:

$$\begin{aligned}\theta_j' = \exists w_1', w_2', \ldots, w_n'(\bigwedge_{i=1}^{n} a(w_i') \wedge \exists x(\pi(x) \wedge \pi_j'(x, w_1', \ldots, w_n'))\\ \wedge \forall x((\pi(x) \wedge \pi_j'(x, w_1', \ldots, w_n')) \supset \psi')).\end{aligned}$$

We apply the preservation of action points lemma to the formula $a(y) \wedge \delta$ to get a finite number of intervals $\pi_1, \ldots, \pi_m$. We now construct $m \times k$ formulas denoted by θ_{ij} where $i \in \{1, 2, \ldots, m\}$ and $j \in \{1, 2, \ldots, k\}$ as follows:

$$\begin{aligned}\theta_{ij} = \exists w_1, w_2, \ldots, w_l \exists w_1', w_2', \ldots, w_n'(\bigwedge_{i=1}^{l} a(w_i) \bigwedge_{i=1}^{n} a(w_i')\\ \wedge \exists x(\pi(x) \wedge \pi_i(x, w_1, \ldots, w_l) \wedge \pi_j'(x, w_1', \ldots, w_n'))\\ \wedge \forall x((\pi(x) \wedge \pi_i(x, w_1, \ldots, w_l) \wedge \pi_j'(x, w_1', \ldots, w_n')) \supset \psi)).\end{aligned}$$

Now define $\theta = \bigvee_{i=1}^{m} \bigvee_{j=1}^{k} \theta_{ij}$.

Pick any timed word σ such that σ strongly satisfies φ. We need to show that $\sigma \models \theta$. Observe that

$$\begin{aligned}\sigma \models \varphi &\implies \sigma \models \exists x(\pi(x) \wedge \exists y(a(y) \wedge \delta \wedge \psi') \\ &\implies \sigma, x = t_1, y = t_0 \models \pi(x) \wedge a(y) \wedge \delta \wedge \psi'.\end{aligned}$$

Hence, $\sigma, y = t_0 \models \exists x(\pi(x) \wedge \psi')$ i.e. σ strongly satisfies φ' with the x-restricted assignment $x = t_1$ for φ' w.r.t σ. Now by the induction hypothesis, $\sigma, y = t_0 \models \theta' \implies \sigma, y = t_0 \models \theta'_{j_0}$ for some j_0. Furthermore, $\sigma, y = t_0 \models \theta'_{j_0}$ with an assignmnet $\mathcal{W}'$ such that $x = t_1, \mathcal{W}' \models \pi'_{j_0}$ and the assignment $\mathcal{W}'$ depends only on σ and the assignment $x = t_1$.

Applying the preservation of action points lemma on the formula $a(y) \wedge \delta$ gives us a $i_0 \in \{1, \ldots, m\}$. We will now show that $\sigma \models \theta_{i_0 j_0}$. Recall that

$$\begin{aligned}\theta_{i_0 j_0} = \exists w_1, w_2, \ldots, w_l \exists w'_1, w'_2, \ldots, w'_n(\bigwedge_{i=1}^{l} a(w_i) \bigwedge_{i=1}^{n} a(w'_i) \\ \wedge \exists x(\pi(x) \wedge \pi_{i_0}(x, w_1, \ldots, w_l) \wedge \pi'_{j_0}(x, w'_1, \ldots, w'_n)) \\ \wedge \forall x((\pi(x) \wedge \pi_{i_0}(x, w_1, \ldots, w_l) \wedge \pi'_{j_0}(x, w'_1, \ldots, w'_n)) \supset \psi)).\end{aligned}$$

We need to come up with an assignment $\mathcal{W}$ for the variables $w_1, \ldots, w_l$ and an assignment $\mathcal{W}'$ for the variables $w'_1, \ldots, w'_n$. The preservation of action points lemma on the formula $a(y) \wedge \delta$ using the timed word σ and the assignment $x = t_1$ for x gives us the assignment $\mathcal{W}$. Also, observe $\sigma, y = t_0 \models \theta'_{j_0}$. Hence,

$$\begin{aligned}\sigma, y = t_0 \models \exists w'_1, w'_2, \ldots, w'_n(\bigwedge_{i=1}^{n} a(w'_i) \wedge \exists x(\pi(x) \wedge \pi'_{j_0}(x, w'_1, \ldots, w'_n)) \\ \wedge \forall x((\pi(x) \wedge \pi'_{j_0}(x, w'_1, \ldots, w'_n)) \supset \psi')).\end{aligned}$$

This gives us a valuation $\mathcal{W}'$ such that

$$\begin{aligned}\sigma, y = t_0, \mathcal{W}' \models (\bigwedge_{i=1}^{n} a(w'_i) \wedge \exists x(\pi(x) \wedge \pi'_{j_0}(x, w'_1, \ldots, w'_n)) \\ \wedge \forall x((\pi(x) \wedge \pi'_{j_0}(x, w'_1, \ldots, w'_n)) \supset \psi')).\end{aligned}$$

Hence, we have $\sigma, \mathcal{W}, \mathcal{W}' \models \bigwedge_{i=1}^{l} a(w_i) \bigwedge_{j=1}^{n} a(w'_j)$. From the preservation of action points lemma, we know that, $\sigma, x = t_1, \mathcal{W} \models \pi_{i_0}(x, w_1, \ldots, w_l)$ and the choice of $\mathcal{W}$ depends only on σ and the assignment $x = t_1$. From the induction hypothesis, we know: $\sigma, x = t_1, \mathcal{W}' \models \pi(x) \wedge \pi'_{j_0}(x, w'_1, \ldots, w'_n)$. Combining these two together, we get, $\sigma, \mathcal{W}, \mathcal{W}' \models \exists x(\pi(x) \wedge \pi_{i_0} \wedge \pi'_{j_0})$.

All that remains is to show that $\sigma, \mathcal{W}, \mathcal{W}' \models \forall x((\pi \wedge \pi_{i_0} \wedge \pi'_{j_0}) \supset \psi)$.

To do this, pick any assignment $x = t'_1$ such that $\sigma, x = t'_1, \mathcal{W}, \mathcal{W}' \models \pi \wedge \pi_{i_0} \wedge \pi'_{j_0}$. Recall that $\psi = \exists y(a(y) \wedge \delta \wedge \psi')$. Hence, we need to show:

$$\sigma, x = t'_1, \mathcal{W}, \mathcal{W}' \models \exists y(a(y) \wedge \delta \wedge \psi').$$

From the preservation of action points lemma, we know that since $\sigma, x = t_1, y = t_0 \models a(y) \wedge \delta$, and since $\sigma, x = t'_1, \mathcal{W} \models \pi_{i_0}$, we have:

$$\sigma, x = t'_1, y = t_0, \mathcal{W}, \mathcal{W}' \models a(y) \wedge \delta.$$

We also already have that

$$\begin{aligned}
&\sigma, y = t_0, \mathcal{W}' \models (\bigwedge_{i=1}^{n} a(w'_i) \wedge \exists x(\pi(x) \wedge \pi'_{j_0}(x, w'_1, \ldots, w'_n)) \\
&\qquad\qquad \wedge \forall x((\pi(x) \wedge \pi'_{j_0}(x, w'_1, \ldots, w'_n)) \supset \psi')) \\
\implies &\sigma, y = t_0, \mathcal{W}' \models \forall x((\pi(x) \wedge \pi'_{j_0}(x, w'_1, \ldots, w'_n)) \supset \psi').
\end{aligned}$$

Now $\sigma, x = t'_1, \mathcal{W}, \mathcal{W}' \models \pi \wedge \pi'_{j_0}$. Hence, we get $\sigma, x = t'_1, y = t_0, \mathcal{W}, \mathcal{W}' \models \psi'$, which gives us $\sigma, x = t'_1, \mathcal{W}, \mathcal{W}' \models \exists y(a(y) \wedge \delta \wedge \psi')$. This is what we needed to show. Hence we are done.

Now consider $\psi = \exists y(a(y) \wedge \delta \wedge \psi')$ where $\psi' = \psi'_1 \wedge \psi'_2 \wedge \cdots \wedge \psi'$ and each ψ'_i is either a formula in $\exists$ normal form, or the negation of a formula in $\exists$ normal form. Consider the formulas: $\varphi'_i = \exists x(\pi \wedge \psi'_i)$ for $i = 1, 2, \ldots, n$. For each $i = 1, 2, \ldots, n$, we can apply the induction hypothesis to get a disjunction $\theta'_i = \bigvee_{j=1}^{k_i} \theta'_{ij}$ such that the lemma holds and where each θ'_{ij} is of the form:

$$\begin{aligned}
\theta'_{ij} = \exists w'_1, w'_2, \ldots, w'_n (&\bigwedge_{i=1}^{n} a(w'_i) \wedge \exists x(\pi(x) \wedge \pi'_{ij}(x, w'_1, \ldots, w'_n)) \\
&\wedge \forall x((\pi(x) \wedge \pi'_{ij}(x, w'_1, \ldots, w'_n)) \supset \psi'_i)).
\end{aligned}$$

We apply the preservation of action points lemma to the formula $a(y) \wedge \delta$ to get a finite number of intervals $\pi_1, \ldots, \pi_m$. We now construct $m \times k_1 \times k_2 \times \cdots \times k_n$ formulas denoted by $\theta_{ij_1j_2\ldots j_n}$ where $i \in \{1, 2, \ldots, m\}$ and $j_i \in \{1, 2, \ldots, k_i\}$ as follows:

$$\begin{aligned}
\theta_{ij_1j_2\ldots j_n} = \exists w_1, w_2, \ldots, w_l (&\bigwedge_{i=1}^{l} a(w_i) \wedge \exists x(\pi(x) \wedge \pi_i \wedge \pi'_{1j_1} \wedge \pi'_{2j_2} \wedge \cdots \wedge \pi'_{nj_n}) \\
&\wedge \forall x(\pi(x) \wedge \pi_i \wedge \pi'_{1j_1} \wedge \pi'_{2j_2} \wedge \cdots \wedge \pi'_{nj_n}) \supset \psi)).
\end{aligned}$$

Now define $\theta = \bigvee_{i=1}^{i=m} \bigvee_{j_1=1}^{j_1=k_1} \cdots \bigvee_{j_n=1}^{j_n=k_n} \theta_{ij_1j_2\ldots j_n}$.

It can be shown similar to the above that the theorem holds with this construction. ◀

Separating Regular Languages over Infinite Words with Respect to the Wagner Hierarchy

Christopher Hugenroth ✉
TU Ilmenau, Germany

Abstract

We investigate the separation problem for regular ω-languages with respect to the Wagner hierarchy where the input languages are given as deterministic Muller automata (DMA). We show that a minimal separating DMA can be computed in exponential time and that some languages require separators of exponential size. Further, we show that in this setting it can be decided in polynomial time whether a separator exists on a certain level of the Wagner hierarchy and that emptiness of the intersection of two languages given by DMAs can be decided in polynomial time. Finally, we show that separation can also be decided in polynomial time if the input languages are given as deterministic parity automata.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Regular languages

Keywords and phrases Separation, Regular, Wagner Hierarchy, Muller Automata, Parity Automata, Product Automata, Membership

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.46

Acknowledgements I want to thank Christof Löding who supervised my Master thesis on which this paper is based and Dietrich Kuske for clarifying discussions. Further, I want to thank the reviewers of former versions of this paper for their helpful suggestions.

1 Introduction

The membership problem for a class of languages asks to decide whether a given language is contained in this class. More generally, we can ask for a hierarchy of classes to decide membership for each of its classes. This problem has been studied for a variety of classes and hierarchies [11, 5, 6, 4, 12]. Solving the membership problem for a class helps us understand the expressive power of a class [8].

For example, the regular ω-languages are classified by the Wagner hierarchy [12, 7] which has been studied extensively. The hierarchy is infinite and refines both the Mostowski [7] and the Borel hierarchy with respect to the regular ω-languages [12]. The class of a language L is determined by the loop structure of any deterministic Muller automaton (DMA) that recognizes L, every DMA that recognizes L has the same structure. Using this, membership for the Wagner hierarchy can be decided efficiently if the language is given as a DMA [7].

Membership has also been studied with respect to the quantifier alternation hierarchy of first order logic for regular language of finite or infinite words [9], [8]. This hierarchy is infinite but the membership problem has only been solved for a few classes of the hierarchy. One approach towards solving membership for more classes uses the separation problem [8].

The Separation problem asks given two languages L_1, L_2 and a class C whether there is a language L in C that separates L_1 and L_2, i.e. $L_1 \subseteq L$ and $L_2 \cap L = \emptyset$. Separation is more general than membership because a language L is in C iff L and its complement can be separated by a language in C, so membership can be reduced to separation. Solving the separation problem for a class C requires understanding the discriminating power of C and therefore an even deeper understanding of a class than for solving membership [8].

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 46; pp. 46:1–46:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we study the separation problem for the Wagner hierarchy to gain a deeper understanding of this hierarchy. Further, we might get some insights on how to solve the membership problem for the Mostowski hierarchy for infinite trees [5, 6] and the aforementioned quantifier alternation hierarchy which are similar to the Wagner hierarchy.

We assume that the input languages for the separation problem are given as DMAs. First, we show that the loop structure of every separating DMA is similar. Using this, a DMA whose language separates the two input languages and which is minimal with respect to the Wagner hierarchy can be computed in exponential time. Next, we show that this result is optimal in the sense that there are infinitely many pairs of DMAs for which every separating DMA is exponentially larger than the DMAs in the pair.

Surprisingly, *deciding* whether a separator exists is possible in polynomial time. This can be done by analyzing the loop structure of a special product automaton. This also works if the input languages are given as deterministic parity automata (DPA).

We can also use the separation algorithm to decide in polynomial time whether the languages of two DMAs (DPAs) are disjoint. Meanwhile, the intersection of DMAs (DPAs) has exponential size in general [1], so our algorithm has better complexity than a naive algorithm for disjointness and, to the best of our knowledge, is the first to do so.

2 Preliminaries

We denote the *natural numbers* by $\mathbb{N} = \{0, 1, 2, \dots\}$. The strict linear order on the natural numbers is denoted by $<$. All numbers in this paper are natural, we do not state that for every number explicitly. The projection function to the i-th component is denoted by pr_i.

An *alphabet* Σ is a finite, non-empty set of symbols. A *finite word* over Σ is a mapping from $\{0, 1, \dots, k-1\}$ to Σ for some $k \in \mathbb{N}$. Here, k is the *length* of w. The *empty word* ε is the unique word of length 0. The *class of all finite words* over Σ is Σ^*. An *infinite word* α over Σ is a mapping from $\mathbb{N}$ to Σ. The *class of all infinite* words over Σ is denoted by Σ^ω.

For $w \in \Sigma^*$ and $i, j \in \mathbb{N}$ with $i, j < |w|$ we define $w[i, j]$ as $w[i, j] = w(i) \dots w(j)$ for $i \leq j$ and $w[i, j] = \varepsilon$ for $i > j$. For infinite words the definition is analogous.

A symbol $a \in \Sigma$ *occurs infinitely often* in a word $\alpha \in \Sigma^\omega$ if there are infinitely many $i \in \mathbb{N}$ with $\alpha(i) = a$. For an infinite word $\alpha \in \Sigma^\omega$ let $\mathrm{Inf}(\alpha)$ be the *set of letters that occur infinitely often* in α.

2.1 Automaton Structures

An *automaton structure* is a tuple $A = (Q, \Sigma, \delta, q_0)$ where Q is a finite, non-empty set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function and $q_0 \in Q$ is the initial state. All automata considered in this paper are deterministic.

The *run* of an automaton structure $A = (Q, \Sigma, \delta, q_0)$ on a finite word $w \in \Sigma^*$ from $p \in Q$ is the word $\rho \in Q^*$ with $\rho(0) = p$ and $\rho(i+1) = \delta(\rho(i), w(i))$ for all $i \in \mathbb{N}$ with $i < |w|$. The *run* of an automaton structure A on an infinite word $\alpha \in \Sigma^\omega$ from a state $p \in Q$ is an infinite word $\rho \in Q^\omega$ with $\rho(0) = p$ and $\rho(i+1) = \delta(\rho(i), \alpha(i))$ for all $i \in \mathbb{N}$. We denote such a run by $\rho_A(p, w)$ for $w \in \Sigma^*$, respectively $\rho_A(p, \alpha)$ for $\alpha \in \Sigma^\omega$.

We write $p \xrightarrow[P]{w} q$ if there is a run of A on w from p to q and the set of states visited on this run is exactly P, i.e. $P = \{p' \in Q \mid \text{ there is } 0 \leq i \leq |w| \text{ such that } \rho_A(p, w)(i) = p'\}$. We write $p \xrightarrow{w} q$ if there is some P such that $p \xrightarrow[P]{w} q$. With $p \to q$ we denote that there is a word $w \in \Sigma^*$ with $p \xrightarrow{w} q$. We assume that all states are reachable from the initial state q_0 for every automaton structure A, i.e. $q_0 \to q$ for all $q \in Q$.

A *loop* of an automaton structure $A = (Q, \Sigma, \delta, q_0)$ is a non-empty, strongly connected set of states. So, $P \subseteq Q$ is a loop if there is a state $p \in P$ and a finite word $v \in \Sigma^*$ with $v \neq \varepsilon$ and $p \xrightarrow[P]{v} p$. Equivalently, P is a loop if there is an infinite word $\alpha \in \Sigma^\omega$ such that the infinite run ρ of A on α from q_0 satisfies $\mathrm{Inf}(\rho) = P$. A loop P of A is a *strongly connected component* (SCC) if there is no loop P' of A with $P' \supsetneq P$.

Notice that for every loop P of A there is exactly one SCC S of A with $P \subseteq S$. If there are two SCCs S, S' with $P \subseteq S, S'$ we have $S \cap S' \neq \emptyset$. So, $S = S'$ since the union of non-disjoint loops is a loop again.

2.2 Deterministic Muller Automata

We denote the powerset of a set Q by $2^Q = \{P \mid P \subseteq Q\}$.

A *deterministic Muller automaton* (DMA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ where $A = (Q, \Sigma, \delta, q_0)$ is an automaton structure and $\mathcal{F} \subseteq 2^Q$ is an *acceptance condition*, we also write $\mathcal{A} = (A, \mathcal{F})$.

When we talk about a run or a loop of a DMA $\mathcal{A}$ we mean a run or a loop of its automaton structure. A loop P is accepting in $\mathcal{A}$ if $P \in \mathcal{F}$ and rejecting otherwise. A DMA $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ *accepts* an infinite word $\alpha \in \Sigma^\omega$ if the run $\rho_{\mathcal{A}}(q_0, \alpha)$ satisfies $\mathrm{Inf}(\rho_{\mathcal{A}}(q_0, \alpha)) \in \mathcal{F}$, i.e. the set of states visited infinitely often is an accepting loop. The language *accepted* by $\mathcal{A}$ is the set of words $L(\mathcal{A})$ accepted by $\mathcal{A}$.

The size $|A|$ of a DMA $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ is $|Q| + |\mathcal{F}|$. Notice that $|\mathcal{F}|$ might be exponentially larger than $|Q|$. For a DMA $(A, \mathcal{F})$ we can compute a DMA $(A, \mathcal{F}')$ such that $L(A, \mathcal{F}) = L(A, \mathcal{F}')$ and $\mathcal{F}'$ contains only loops in polynomial time with respect to $|(A, \mathcal{F})|$.

An ω-language L is regular iff there is a DMA $\mathcal{A}$ with $L(\mathcal{A}) = L$.

3 Wagner Hierarchy

Wagner defined chains and superchains for deterministic Muller automata. He proved that if L is an ω-regular language then every DMA that recognizes L has the same maximal superchain length [12]. So, the maximal length of superchains is an invariant of the language. Hence, the regular ω-languages can be classified according to this invariant. This classification forms the Wagner hierarchy.

3.1 Chains

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ be a DMA. For $m \geq 1$ an *m-chain* of $\mathcal{A}$ is a sequence of loops $c = (P_1, \dots, P_m)$ such that $P_i \subseteq P_{i+1}$ and $P_i \in \mathcal{F}$ iff $P_{i+1} \notin \mathcal{F}$ for all $0 < i < m$. We also speak of a chain c if it is clear from context that c is an m-chain for a certain $m \in \mathbb{N}$. A chain $c = (P_1, \dots, P_m)$ is *positive* if P_1 is accepting and *negative* if P_1 is rejecting.

An m'-chain $c' = (P'_1, \dots, P'_{m'})$ is *reachable* from an m-chain $c = (P_1, \dots, P_m)$ if there are states $p \in P_1$, $p' \in P'_1$ with $p \to p'$. We denote this by $c \to c'$. Notice that this is equivalent to saying that every (or some) state of $P'_{m'}$ can be reached from every (or some) state of P_m.

3.2 Superchains

A superchain is a reachability-ordered sequence of chains which alternate between positive and negative chains. Formally, an *(m, n)-superchain* is a sequence $s = (c_1, \dots, c_n)$ of m-chains such that c_i is positive iff c_{i+1} is negative and further $c_i \to c_{i+1}$ for all $0 < i < n$. An (m, n)-superchain $s = (c_1, \dots, c_n)$ is *positive* if c_1 is positive and *negative* otherwise.

▶ Remark 1. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ be a DMA and $m > 1$. The following are equivalent:

1. $\mathcal{A}$ has an m-chain.
2. $\mathcal{A}$ has an $(m-1, n)$-superchain for all $n \in \mathbb{N}$.
3. $\mathcal{A}$ has an $(m-1, |Q|+1)$-superchain.

Let $\leq$ be the lexicographic order on $\mathbb{N}^2$. An (m', n')-superchain is *longer* than an (m, n)-superchain if $m' > m$ or if $m' = m$ and $n' > n$, i.e. $(m', n') > (m, n)$. This order prioritizes m over n which ensures that for each DMA there is a maximal length of superchains in this DMA. If we prioritize n this is not the case as one can see with Remark 1.

3.3 Wagner hierarchy

Wagner showed that the superchains of a DMA $\mathcal{A}$ are an invariant of the language $L(\mathcal{A})$ [12] in the following sense: For $m, n \in \mathbb{N}$ and two DMAs $\mathcal{A}_1$, $\mathcal{A}_2$ with $L(\mathcal{A}_1) = L(\mathcal{A}_2)$ it holds that $\mathcal{A}_1$ has an (m, n)-superchain if and only if $\mathcal{A}_2$ has an (m, n)-superchain. So, the existence of a superchain in $\mathcal{A}$ is completely determined by $L(\mathcal{A})$.

The *Wagner hierarchy* classifies the regular ω-languages based on superchains. We consider downward-closed classes that Wagner defined. For all $m, n \in \mathbb{N}$ there are three classes C^n_m, D^n_m, E^n_m which are defined as follows:

$$\begin{aligned}
E^n_m &= \{L(\mathcal{A}) \mid \text{If } \mathcal{A} \text{ has an } (k,\ell)\text{-superchain then } (k,\ell) \leq (m,n)\} \\
D^n_m &= \{L(\mathcal{A}) \mid \text{If } \mathcal{A} \text{ has an } (k,\ell)\text{-superchain then } (k,\ell) \leq (m,n) \\
&\qquad \text{and all } (m,n)\text{-superchains in } \mathcal{A} \text{ are positive}\} \\
C^n_m &= \{L(\mathcal{A}) \mid \text{If } \mathcal{A} \text{ has an } (k,\ell)\text{-superchain then } (k,\ell) \leq (m,n) \\
&\qquad \text{and all } (m,n)\text{-superchains in } \mathcal{A} \text{ are negative}\}
\end{aligned}$$

It holds $C^n_m, D^n_m \subseteq E^n_m$ and E^n_m is closed under complement.

4 Wagner Separation

Let X^n_m be a Wagner class. We say that two languages $L_1, L_2 \subseteq \Sigma^\omega$ are *X^n_m-separable* if there is a language $L \in X^n_m$ with $L_1 \subseteq L$ and $L_2 \cap L = \emptyset$. Notice that L_1, L_2 are E^n_m-separable iff L_2, L_1 are E^n_m-separable. Further, L_1, L_2 are C^n_m-separable iff L_2, L_1 are D^n_m-separable, if L separates L_1, L_2 then $\Sigma^\omega \setminus L$ separates L_2, L_1. We investigate the following problem:

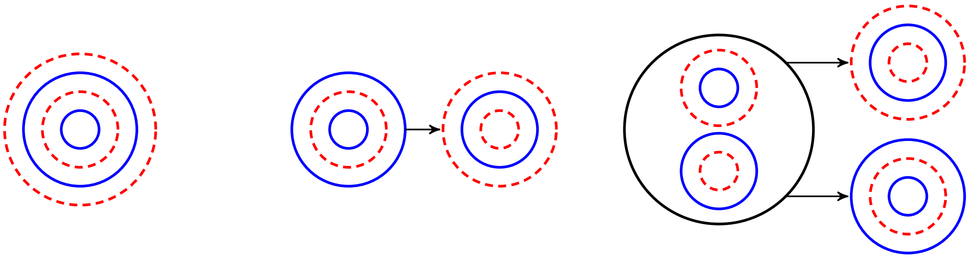
WagnerSeparation
Given: Two DMAs $\mathcal{A}_1, \mathcal{A}_2$, $X \in \{C, D, E\}$ and $m, n \in \mathbb{N}$.
Decide: Are the languages $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ X^n_m-separable?

In the following we define chains for DMAs that can recognize two languages and show how to construct a separating DMA that is minimal with respect to the Wagner hierarchy.

4.1 Generalization of Chains

Consider a DMA $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$. Wagner defined chains in $\mathcal{A}$ as sequences of loops that alternate between $\mathcal{F}$ and its complement. It turns out that a slight generalization of this concept is helpful for this paper. Let $\mathcal{F}_1, \mathcal{F}_2$ be two subsets of 2^Q. We say that $c = (P_1, \ldots, P_m)$ is an *m-chain with respect to* $(\mathcal{F}_1, \mathcal{F}_2)$ if

1. P_i is a loop for $1 \leq i \leq m$,
2. $P_i \subseteq P_{i+1}$ for $1 \leq i < m$,
3. $P_i \in \mathcal{F}_1 \cup \mathcal{F}_2$ for $1 \leq i \leq m$,
4. $P_i \in \mathcal{F}_1$ iff $P_{i+1} \in \mathcal{F}_2$ for $1 \leq i < m$.



(a) Blue-start, red-end 4-chain. **(b)** Blue-start $(3, 2)$-superchain. **(c)** A $(3, 2)$-forcing SCC.

Figure 1 Illustration of patterns in A. Each circle represents a loop in A. Notice that no matter how the black loop in (c) is colored there will be a $(3, 2)$-superchain in A.

An m-chain $c = (P_1, \ldots, P_m)$ with respect to $(\mathcal{F}_1, \mathcal{F}_2)$ is $\mathcal{F}_1$-start if $P_1 \in \mathcal{F}_1$ and otherwise it is $\mathcal{F}_2$-start. Notice that c is an m-chain in $\mathcal{A}$ (in the sense of Wagner) if and only if c is an m-chain with respect to $(\mathcal{F}, 2^Q \setminus \mathcal{F})$. Further, notice that there are arbitrarily long chains with respect to $(\mathcal{F}_1, \mathcal{F}_2)$ whenever these sets are not disjoint.

An *(m, n)-superchain with respect to* $(\mathcal{F}_1, \mathcal{F}_2)$ is a sequence $s = (c_1, \ldots, c_n)$ such that each c_j is an m-chain with respect to $(\mathcal{F}_1, \mathcal{F}_2)$, c_{j+1} is reachable from c_j and c_j is $\mathcal{F}_1$-start iff c_{j+1} is $\mathcal{F}_2$-start for all $1 \leq j < n$. The superchain $s = (c_1, \ldots, c_n)$ is $\mathcal{F}_1$-start if c_1 is $\mathcal{F}_1$-start and otherwise it is $\mathcal{F}_2$-start.

4.2 Blue and Red Chains

Let $L_B, L_R \subseteq \Sigma^\omega$ be two regular ω-languages. An automaton structure $A = (Q, \Sigma, \delta, q_0)$ *can accept* both languages L_B and L_R if there are acceptance conditions $\mathcal{F}_B$, $\mathcal{F}_R \subseteq 2^Q$ such that $L(A, \mathcal{F}_B) = L_B$ and $L(A, \mathcal{F}_R) = L_R$. For example, the product automaton structure of two automata can accept both their languages.

For the remainder of the section fix two languages L_B, L_R and an automaton structure $A = (Q, \Sigma, \delta, q_0)$ that can accept both L_B and L_R. There are unique sets of loops $\mathcal{F}_B, \mathcal{F}_R$ such that $L(A, \mathcal{F}_B) = L_B$ and $L(A, \mathcal{F}_R) = L_R$. We refer to $\mathcal{F}_B$ as the set of blue loops and to $\mathcal{F}_R$ as the set of red loops.

We study chains in A with respect to $(\mathcal{F}_B, \mathcal{F}_R)$ and their connection to chains in DMAs whose language separate L_B and L_R. In Section 4.4 we show that if there is an (m, n)-superchain in A with respect to $(\mathcal{F}_B, \mathcal{F}_R)$ then there is an (m, n)-superchain in every DMA that separates L_B and L_R. Additionally, there are loops in A that are neither blue nor red but nevertheless important for the existence of superchains in a separating DMA.

We say that an m-chain $c = (P_1, \ldots, P_m)$ is *blue-end* (*red-end*) if $P_m \in \mathcal{F}_B$ ($P_m \in \mathcal{F}_R$). An SCC S in A is blue-end (red-end) if all of the longest chains in S are blue-end (red-end). An SCC S is (m, n)-*forcing* for A if all of the following conditions hold:

1. A has a blue-end $(m - 1)$-chain $c_B = (P^B_1, \ldots, P^B_{m-1})$ and a red-end $(m - 1)$-chain $c_R = (P^R_1, \ldots, P^R_{m-1})$.
2. These chains are contained in S, i.e. $P^B_{m-1} \subseteq S$ and $P^R_{m-1} \subseteq S$.
3. A has an $(m, n-1)$-superchain s_B whose first chain is blue-end and an $(m, n-1)$-superchain s_R whose first chain is red-end.
4. These superchains can be reached from S.

See Figure 1 for an illustration. Notice that if A has an (m, n)-superchain with respect to $(\mathcal{F}_B, \mathcal{F}_R)$ then the SCC that contains its first chain is (m, n)-forcing. We say that A is at most (m, n)-forcing if for every SCC in A that is (k, ℓ)-forcing we have $(k, \ell) \leq (m, n)$.

4.3 Separator

In the following we show that we can define two DMAs that separate L_B and L_R (if possible) and show later that one of them is minimal with respect to the Wagner hierarchy. To construct these DMAs we use the automaton structure A. If there is a loop in A that is blue and red then $L_B \cap L_R \neq \emptyset$, so the languages cannot be separated. Otherwise, $L(A, \mathcal{F}_B)$ separates L_B and L_R.

We give an intuitive description of the separator before defining it formally. We define an automaton structure A_{sep} that consists of three copies of A such that each loop of A is in one of the copies. The languages L_B and L_R can be accepted by A_{sep} using the sets $\mathcal{F}_B^0$, $\mathcal{F}_R^0$ of loops whose projection to A is in $\mathcal{F}_B$, $\mathcal{F}_R$ respectively.

Each SCC is then added to $\mathcal{F}_B^0$ or $\mathcal{F}_R^0$, this yields sets $\mathcal{F}_B^1$ and $\mathcal{F}_R^1$. For example, a blue-end SCC is in $\mathcal{F}_B^1$. If A is at most (m, n)-forcing then each superchain in A_{sep} with respect to $(\mathcal{F}_B^1, \mathcal{F}_R^1)$ has at most length (m, n).

Then, it remains to fix the acceptance of the loops that are not an SCC and have no color (neither blue nor red). We fix the acceptance of these remaining loops based on the longest chains with respect to $(\mathcal{F}_B^1, \mathcal{F}_R^1)$. This yields the sets $\mathcal{F}_B^2, \mathcal{F}_R^2$, analogously we construct $\mathcal{G}_B^1, \mathcal{G}_R^1, \mathcal{G}_B^2, \mathcal{G}_R^2$. The DMA whose language separates L_B and L_R and which are minimal with respect to the Wagner hierarchy are $(A_{\text{sep}}, \mathcal{F}_B^2)$ and $(A_{\text{sep}}, \mathcal{G}_B^2)$.

Separator Construction

Recall that $A = (Q, \Sigma, \delta, q_0)$ is an automaton structure that can accept both L_B and L_R. Let $m, n \in \mathbb{N}$ such that A is at most (m, n)-forcing.

We construct an automaton structure $A_{\text{sep}} = (Q_{\text{sep}}, \Sigma, \delta_{\text{sep}}, q_{\text{sep}})$. The states $Q_{\text{sep}} = Q \times \{N, B, R\}$ consist of three copies of the states of A. The initial state $q_{\text{sep}} = (q_0, N)$ is in the N-copy. These copies are used to remember in a run whether an m-chain has been seen and if so it also remembers whether the most recent m-chain was blue-end or red-end. For $p \in Q$, $a \in \Sigma$, $q = \delta(p, a)$ and $C \in \{N, B, R\}$ let

$$\delta_{\text{sep}}((p, C), a) = \begin{cases} (q, B) & \text{, if there is an SCC } S \text{ in } A \text{ with } q \in S \\ & \text{ and there is a blue-end } m\text{-chain in } S \\ (q, R) & \text{, if there is an SCC } S \text{ in } A \text{ with } q \in S \\ & \text{ and there is a red-end } m\text{-chain in } S \\ (q, C) & \text{, otherwise} \end{cases}$$

The transition function δ_{sep} is well-defined because in no SCC in A there are both a blue- and a red-end m-chain since A is at most (m, n)-forcing. Notice that if a run in A_{sep} enters a new copy then the corresponding run in A has to leave an SCC because the m-chains in the SCC changed. So, every loop P in A_{sep} is either in $Q \times \{B\}$, in $Q \times \{R\}$ or in $Q \times \{N\}$.

We define sets $\mathcal{F}_B^1$ and $\mathcal{F}_R^1$ that cover every SCC and every set whose projection is colored. Let $P \subseteq Q_{\text{sep}}$ be a set. Then, $P \in \mathcal{F}_B^1$ if one of the following is true:

1. $\text{pr}_1(P)$ is blue in A
2. $\text{pr}_1(P)$ is colorless, $P \subseteq Q \times \{B\}$, P is an SCC
3. $\text{pr}_1(P)$ is colorless, $P \subseteq Q \times \{N\}$, P is an SCC, $\text{pr}_1(P)$ is a blue-end SCC
4. $\text{pr}_1(P)$ is colorless, $P \subseteq Q \times \{N\}$, P is an SCC and there is no (m, n)-superchain reachable from $\text{pr}_1(P)$ whose first chain is red-end

Similarly, $P \in \mathcal{F}_R^1$ if one of the following is true:

1. $\mathrm{pr}_1(P)$ is red in A
2. $\mathrm{pr}_1(P)$ is colorless, $P \subseteq Q \times \{R\}$, P is an SCC
3. $\mathrm{pr}_1(P)$ is colorless, $P \subseteq Q \times \{N\}$, P is an SCC, there is an (m, n)-superchain reachable from $\mathrm{pr}_1(P)$ whose first chain is red-end and $\mathrm{pr}_1(P)$ is not a blue-end SCC

▶ **Lemma 2.** *The longest superchains in A_{sep} w.r.t. $(\mathcal{F}_B^1, \mathcal{F}_R^1)$ have at most length (m, n).*

The sets $\mathcal{F}_B^1$ and $\mathcal{F}_R^1$ are defined to minimize the chain length but there are other sets that also do this. We define two such sets $\mathcal{G}_B^1$ and $\mathcal{G}_R^1$ that only differ in the third and fourth condition from $\mathcal{F}_B^1$ and $\mathcal{F}_R^1$. A loop P is in $\mathcal{G}_B^1$ if one of the following is true

1. $\mathrm{pr}_1(P)$ is blue in A
2. $\mathrm{pr}_1(P)$ is colorless, $P \subseteq Q \times \{B\}$, P is an SCC
3. $\mathrm{pr}_1(P)$ is colorless, $P \subseteq Q \times \{N\}$, P is an SCC, there is an (m, n)-superchain reachable from $\mathrm{pr}_1(P)$ whose first chain is blue-end and $\mathrm{pr}_1(P)$ is not a red-end SCC

Similarly, $P \in \mathcal{G}_R^1$ if one of the following is true

1. $\mathrm{pr}_1(P)$ is red in A
2. $\mathrm{pr}_1(P)$ is colorless, $P \subseteq Q \times \{R\}$, P is an SCC
3. $\mathrm{pr}_1(P)$ is colorless, $P \subseteq Q \times \{N\}$, P is an SCC, $\mathrm{pr}_1(P)$ is a red-end SCC
4. $\mathrm{pr}_1(P)$ is colorless, $P \subseteq Q \times \{N\}$, P is an SCC and there is no (m, n)-superchain reachable from $\mathrm{pr}_1(P)$ whose first chain is blue-end

▶ **Lemma 3.** *The longest superchains in A_{sep} w.r.t. $(\mathcal{G}_B^1, \mathcal{G}_R^1)$ have at most length (m, n).*

To obtain a separator that is minimal with respect to the Wagner hierarchy it is not only important how long superchains are but also how they start.

▶ **Lemma 4.** *If there are no blue-start (red-start) (m, n)-superchains in A then there are no $\mathcal{F}_B^1$-start ($\mathcal{F}_R^1$-start) (m, n)-superchains in A_{sep} with respect to $(\mathcal{F}_B^1, \mathcal{F}_R^1)$ or there are no $\mathcal{G}_B^1$-start ($\mathcal{G}_R^1$-start) (m, n)-superchains in A_{sep} with respect to $(\mathcal{G}_B^1, \mathcal{G}_R^1)$.*

We define the acceptance for every loop P that is not in $\mathcal{F}_B^1$ or $\mathcal{F}_R^1$ based on the longest chains with respect to $(\mathcal{F}_B^1, \mathcal{F}_R^1)$ that start with a strict superloop of P. Let P be a loop in A_{sep} such that there is an SCC S with $P \subsetneq S$. We define $P^{\subsetneq} = \{P' \mid P \subsetneq P' \subseteq S\}$. For P consider the chains with respect to $(\mathcal{F}_B^1 \cap P^{\subsetneq}, \mathcal{F}_R^1 \cap P^{\subsetneq})$. A loop $P \subseteq Q_{\mathrm{sep}}$ is in $\mathcal{F}_B^2$ if $P \in \mathcal{F}_B^1$ or if $P \notin \mathcal{F}_R^1$ and the longest chains with respect to $(\mathcal{F}_B^1 \cap P^{\subsetneq}, \mathcal{F}_R^1 \cap P^{\subsetneq})$ are all $(\mathcal{F}_B^1 \cap P^{\subsetneq})$-start. So, a set $P \subseteq Q_{\mathrm{sep}}$ is in its complement $\mathcal{F}_R^2 = 2^Q \setminus \mathcal{F}_B^2$ if $P \notin \mathcal{F}_B^1$ and $P \in \mathcal{F}_R^1$ or if $P \notin \mathcal{F}_B^1$ and the longest chains with respect to $(\mathcal{F}_B^1 \cap P^{\subsetneq}, \mathcal{F}_R^1 \cap P^{\subsetneq})$ are all $(\mathcal{F}_R^1 \cap P^{\subsetneq})$-start. We define $\mathcal{G}_B^2$ and $\mathcal{G}_B^R$ analogously.

We show that $\mathcal{F}_B^2$ is well-defined, the proof for $\mathcal{G}_B^2$ is analogous. The longest chains with respect to $(\mathcal{F}_B^1 \cap P^{\subsetneq}, \mathcal{F}_R^1 \cap P^{\subsetneq})$ are all $(\mathcal{F}_B^1 \cap P^{\subsetneq})$-end or all $(\mathcal{F}_R^1 \cap P^{\subsetneq})$-end because S is in $\mathcal{F}_B^1 \cap P^{\subsetneq}$ or in $\mathcal{F}_R^1 \cap P^{\subsetneq}$. So, the longest chains are also all $(\mathcal{F}_B^1 \cap P^{\subsetneq})$-start or all $(\mathcal{F}_R^1 \cap P^{\subsetneq})$-start. Further, there is always a chain in $P^{\subsetneq}$ because (S) is a chain.

▶ **Lemma 5.** *If there is a $\mathcal{F}_B^2$-start ($\mathcal{F}_R^2$-start) (k, ℓ)-superchain in A with respect to $(\mathcal{F}_B^2, \mathcal{F}_R^2)$ then there is a $\mathcal{F}_B^1$-start ($\mathcal{F}_R^1$-start) (k, ℓ)-superchain in A with respect to $(\mathcal{F}_B^1, \mathcal{F}_R^1)$.*

The analogous statement holds for $(\mathcal{G}_B^1, \mathcal{G}_R^1)$ and $(\mathcal{G}_B^2, \mathcal{G}_R^2)$. Finally, we can define the DMA $(A_{\mathrm{sep}}, \mathcal{F}_B^2)$ and $(A_{\mathrm{sep}}, \mathcal{G}_B^2)$ whose languages separate L_B and L_R if possible.

▶ **Lemma 6.** *If $L_B \cap L_R = \emptyset$ then $L(A_{sep}, \mathcal{F}_B^2)$, $L(A_{sep}, \mathcal{G}_B^2)$ separate L_B and L_R.*

4.4 Translating Chains

We show that the separator constructed in the previous section is minimal with respect to the Wagner hierarchy. Let $\mathcal{A}_B, \mathcal{A}_R, \mathcal{A}_{\text{sep}}$ be DMAs such that $L(\mathcal{A}_{\text{sep}})$ separates $L(\mathcal{A}_B)$ and $L(\mathcal{A}_R)$. Further, let A be an automaton structure that can accept both $L(\mathcal{A}_B)$ and $L(\mathcal{A}_R)$.

▶ **Lemma 7.** *If A has an (m, n)-forcing SCC then $\mathcal{A}_{sep}$ has an (m, n)-superchain.*

To prove Lemma 7 we consider the loops that appear in the chains and in the superchains of the (m, n)-forcing SCC. We find finite words that run repeatedly through these loops. These words have runs in any separating DMA $\mathcal{A}_{\text{sep}}$. These runs yield loops and ultimately superchains in $\mathcal{A}_{\text{sep}}$. An analysis of this proof yields the following result:

▶ **Lemma 8.** *If A has a blue-start (red-start) (m, n)-superchain then $\mathcal{A}_{sep}$ has a positive (negative) (m, n)-superchain.*

Combining all previous results yields the main result of this paper.

▶ **Theorem 9.** *Let $m, n \in \mathbb{N}$ and $L_B, L_R \subseteq \Sigma^\omega$. Let A be an automaton structure that can accept both L_B and L_R. The languages L_B and L_R are*

1. *E^n_m-separable iff A is at most (m, n)-forcing,*
2. *D^n_m-separable iff A is at most (m, n)-forcing and every (m, n)-superchain in A is blue-start,*
3. *C^n_m-separable iff A is at most (m, n)-forcing and every (m, n)-superchain in A is red-start.*

▶ **Corollary 10.** *L_B, L_R are C^n_m- and D^n_m-separable iff there are no (m, n)-superchains in A.*

5 Solving Separation

We use Theorem 9 to show how WagnerSeparation and related questions can be resolved.

5.1 Computing a Separator

We show that DMAs as defined in Section 4.3 can be computed in exponential time. For this we use product automata. Let A_1, A_2 be two automaton structures with $A_i = (Q_i, \Sigma, \delta_i, q_i)$ for $i \in \{1, 2\}$. The *product automaton* of A_1 and A_2 is the automaton structure $A_1 \times A_2 = (Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2))$, where δ applies the transition functions component wise, i.e. for $(p_1, p_2) \in Q_1 \times Q_2$ and $a \in \Sigma$ we have $\delta((p_1, p_2), a) = (\delta_1(p_1, a), \delta_2(p_2, a))$. The product automaton of two DMAs is defined as the product automaton of their automaton structures.

▶ **Theorem 11.** *Let $\mathcal{A}_1$, $\mathcal{A}_2$ with $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, q_i, \mathcal{F}_i)$ be two DMA. A separating DMA that is minimal with respect to the Wagner hierarchy can be constructed in exponential time.*

The product automaton structure $A_1 \times A_2$ can be computed in polynomial time and it can accept both $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$. The set of states of the automaton structure A_{sep} consists of three copies of $Q_1 \times Q_2$ which clearly can be computed in polynomial time. The transition function δ_{sep} and the acceptance conditions $\mathcal{F}^2_B$, $\mathcal{G}^2_B$ are defined based on the chain structure in the product automaton with respect to blue and red loops.

The chain structure with respect to the red and blue loops can be computed as follows. Iterate over all subsets of $Q_1 \times Q_2$ and color a subset blue if its projection to the first component is in $\mathcal{F}_1$. Color a subset red if its projection to the second component is in $\mathcal{F}_2$.

Next, construct a graph that has the colored loops as nodes. Introduce an edge between two loops if they have different colors and the first loop is a subset of the second. Then the paths of maximal length in this graph correspond to chains of maximal lengths. To compute the superchains of maximal length a new graph can be constructed with chains of maximal length as nodes. Further, there is an edge between two chains if the second chain is reachable from the first chain and they start with different colors. The superchains of maximal length then correspond to paths of maximal length in this graph. The constructed graphs are acyclic.

These graphs are of exponential size in $|\mathcal{A}_1|+|\mathcal{A}_2|$ or smaller. Using for example Dijkstra's algorithm, the paths of maximal length in an acyclic graph can be computed in polynomial time. Thus, the chain structure of A and therefore $(A_{\text{sep}}, \mathcal{F}_B^2)$, $(A_{\text{sep}}, \mathcal{G}_B^2)$ can be computed in exponential time.

5.2 Exponential blowup of Separators

We show that in some cases the size of every separator that is minimal with respect to the Wagner hierarchy is exponentially larger than the size of the input DMAs. So, if one wants to compute a complete separator (not just deciding the value of a certain bit) then exponential time is the best complexity one can hope for.

▶ **Theorem 12.** *For all $1 \leq m \leq k \in \mathbb{N}$ there are DMAs $\mathcal{A}_1, \mathcal{A}_2$ of size at most $2k$ such that every DMA $\mathcal{A}$ with no $(m+1)$-chain whose language separates $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ has size at least $2^{k-m} - k - 1$ and there is such a DMA.*

Proof. Let $\Sigma = \{1, \dots, k\}$. We define the DMA $\mathcal{A}_1$, $\mathcal{A}_2$ that have the same automaton structure A. Let $A = (Q, \Sigma, \delta, 1)$ with $Q = \{1, \dots, k\} \cup \{\bot\}$ and δ as follows: For a state $i \in Q$ and $j \in \Sigma$ let $\delta(i, j) = j$ if $i \neq j$ and $\delta(i, j) = \bot$ if $i = j$. Further, $\delta(\bot, j) = \bot$ for all $j \in \Sigma$, so $\bot$ is a sink state.

The DMA $\mathcal{A}_1 = (Q, \Sigma, \delta, 1, \mathcal{F}_1)$, $\mathcal{A}_2 = (Q, \Sigma, \delta, 1, \mathcal{F}_2)$ differ in their acceptance conditions. The set $\{1, \dots, k-m+1\}$ is in $\mathcal{F}_1$. If $\{1, \dots, i\}$ is in $\mathcal{F}_1$ then $\{1, \dots, i+1\}$ is in $\mathcal{F}_2$ for $i < m$. Similarly, if $\{1, \dots, i\} \in \mathcal{F}_2$ then $\{1, \dots, i+1\} \in \mathcal{F}_1$ for $i < m$. Further, $\{\bot\} \in \mathcal{F}_1$ and no other sets are in $\mathcal{F}_1$, $\mathcal{F}_2$.

Clearly, the automaton structure A can recognize both $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$. Further, the longest chain in A with respect to $(\mathcal{F}_1, \mathcal{F}_2)$ is the $\mathcal{F}_1$-first m-chain $c = (P_1, \dots, P_m)$ with $P_i = \{1, \dots, k-m+i\}$. Let $\mathcal{F} = \mathcal{F}_1 \cup 2^{\{1,\dots,k-m\}}$ and consider $\mathcal{A} = (Q, \Sigma, \delta, 1, \mathcal{F})$. Then $L(\mathcal{A})$ separates $L(\mathcal{A}_1)$, $L(\mathcal{A}_2)$ and $\mathcal{A}$ has no $(m+1)$-chain.

Let $\mathcal{A}_{\text{sep}} = (Q_{\text{sep}}, \Sigma, \delta_{\text{sep}}, q_{\text{sep}}, \mathcal{F}_{\text{sep}})$ be a DMA with no $(m+1)$-chain that separates $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ and let $r = |Q_{\text{sep}}|$. To prove $|\mathcal{A}_{\text{sep}}| \geq 2^{k-m} - m - 1$, we define an injective function $f : (2^{\{1,\dots,k-m\}} \setminus \{\emptyset, \{1\}, \dots, \{k\}\}) \to \mathcal{F}_{\text{sep}}$.

Definition of the function

For a set $P = \{p_1, \dots, p_{|P|}\} \subseteq \{1, \dots, k-m\}$ with $|P| > 1$ we define the word $w_P = p_1 \dots p_{|P|}$ where $p_1 < p_2 < \dots < p_{|P|}$. Let P be a non-empty subset of $\{1, \dots, k-m\}$ and $P_i = \{1, \dots, k-m+i\}$ with $1 \leq i \leq m$ and consider the words

$$w_0 = w_P^r,$$
$$w_i = (w_{i-1} w_{P_i})^r, \text{ for } 1 \leq i \leq m.$$

Consider the finite run ρ of $\mathcal{A}_{\text{sep}}$ on w_m. For $1 \leq i \leq m$ consider the state in which $\mathcal{A}_{\text{sep}}$ is before reading $w_i = (w_{i-1} w_{P_i})^r$ and the states after each $w_{i-1} w_{P_i}$. These are $r+1 = |Q_{\text{sep}}|+1$ many, so one state must occur twice. So, for $1 \leq i \leq m$ there are positions $0 \leq d_i < e_i \leq |w_m|$ in the word w_m and a number of repetitions $r_i \in \mathbb{N}$ with $\rho(d_i) \xrightarrow{(w_{i-1} w_{P_i})^{r_i}} \rho(e_i)$. Because

w_{i-1} is a proper infix of w_i there is a sequence of indices $d_{m+1} \leq \cdots \leq d_1 < e_1 \leq \cdots \leq e_{m+1}$ with $\rho(d_i) \xrightarrow{(w_{i-1}w_{P_i})^{r_i}} \rho(e_i)$ for $0 \leq i \leq m$. Consider $Q_i = \{\rho(j) \mid d_i \leq j \leq e_i\}$ for $1 \leq i \leq m$.

Similarly, we get d_0, e_0, r_0 with $\rho(d_0) \xrightarrow{(w_P)^{r_0}} \rho(e_0)$ and $d_1 \leq d_o < e_0 \leq e_1$. We set $f(P) = Q_0 = \{\rho(j) \mid d_0 \leq j \leq e_0\}$.

Image of the function

We show that $(Q_1, \ldots, Q_m)$ is an $(2^Q \setminus \mathcal{F}_{\text{sep}})$-start m-chain in $\mathcal{A}_{\text{sep}}$. Notice that each Q_i is a loop and that $Q_0 \subseteq Q_1 \subseteq \cdots \subseteq Q_m$. Further, notice that $P_i \in \mathcal{F}_1$ if i is even and $P_i \in \mathcal{F}_2$ if i is odd for $1 \leq i \leq m$. So, $w_m(w_i)^\omega \in L(\mathcal{A}_1)$ if i is even and $w_m(w_i)^\omega \in L(\mathcal{A}_2)$ if i is odd for $1 \leq i \leq m$. Because $L(\mathcal{A}_{\text{sep}})$ separates $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ we have $w_m(w_i)^\omega \in L(\mathcal{A}_{\text{sep}})$ iff i is odd. Let ρ_i be the run of $\mathcal{A}_{\text{sep}}$ on $w_m(w_i)^\omega$. Then, $\text{Inf}(\rho_i) = Q_i$ and therefore $Q_i \in \mathcal{F}_{\text{sep}}$ iff i is odd.

Now assume that $f(P) = Q_0 \notin \mathcal{F}_{\text{sep}}$ for some $P \subseteq \{1, \ldots, k-m\}$. Then $c = (Q_0, \ldots, Q_m)$ is an $(m+1)$-chain in $\mathcal{A}_{\text{sep}}$ which contradicts the assumption. Thus, $f(P) \in \mathcal{F}_{\text{sep}}$.

The function is injective

Assume that there are $P, P' \subseteq \{1, \ldots, k-m\}$ such that $|P|, |P'| > 1$, $P \neq P'$ and $f(P) = f(P')$. There are states $p \in f(P), p' \in f(P')$ and $r_0, r'_0 \in \mathbb{N}$ such that $p \xrightarrow[f(P)]{w_P^{r_0}} p$ and $p' \xrightarrow[f(P)]{w_{P'}^{r'_0}} p'$. We have $P \neq P'$, so $w_P \neq w_{P'}$. Without loss of generality there is a letter a in w_P that does not occur in $w_{P'}$. This letter is mapped to some state q in $f(P)$ and some letter b of $w_{P'}$ is mapped to this state as well because $f(P) = f(P')$. Thus, there are $x, x' \in \Sigma^*$, $a, b \in \Sigma$, $a \neq b$ and a state $q \in f(P)$ such that $q_{\text{sep}} \xrightarrow{xa} q$ and $q_{\text{sep}} \xrightarrow{x'b} q$.

So, the words xa and $x'b$ are mapped to the same state. But $xaa\alpha \notin L(\mathcal{A}_{\text{sep}})$ for all $\alpha \in \Sigma^\omega$ while there are $\alpha \in \Sigma^\omega$ with $xba\alpha \in L(\mathcal{A}_{\text{sep}})$. These words cannot be distinguished by $\mathcal{A}_{\text{sep}}$, contradiction.

Thus, f is an injective mapping from $2^{\{1,\ldots,k-m\}} \setminus \{\emptyset, \{1\}, \ldots, \{k\}\}$ to $\mathcal{F}_{\text{sep}}$ and therefore $2^{k-m} - k - 1 \leq |Q_{\text{sep}}| + |\mathcal{F}_{\text{sep}}| = |\mathcal{A}_{\text{sep}}|$. ◀

The proof idea is based on a construction in [10]. The construction can be extended to show a lower bound for a symmetric separation concept, that is $L_2 \subseteq L$ and $L \cap L_1 = \emptyset$ is also allowed, by adding a copy of A where $\mathcal{F}_1$ and $\mathcal{F}_2$ are swapped.

5.3 Deciding Separability

Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two DMAs with $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, q_i, \mathcal{F}_i)$ and consider their product automaton structure $A_1 \times A_2$. Further, let $\mathcal{F}_B = \{P \subseteq Q_1 \times Q_2 \mid \text{pr}_1(P) \in \mathcal{F}_1\}$ and $\mathcal{F}_R = \{P \subseteq Q_1 \times Q_2 \mid \text{pr}_2(P) \in \mathcal{F}_2\}$. According to Theorem 9 it suffices to determine the loop structure of $A_1 \times A_2$ with respect to $(\mathcal{F}_B, \mathcal{F}_R)$ to decide Wagner-separability.

However, there might be exponentially many loops in $\mathcal{F}_B$ or in $\mathcal{F}_R$. We show that it suffices to consider only certain maximal loops whose number is polynomial in $|\mathcal{A}_1| + |\mathcal{A}_2|$. A similar idea has already been used in [3].

The set $\mathcal{M} \subseteq 2^{Q_1 \times Q_2}$ *contains a set* $P \subseteq Q_1 \times Q_2$ if there are $P_1 \in \mathcal{F}_1$, $P_2 \in \mathcal{F}_2$ such that P is a maximal loop in $P_1 \times P_2$ with respect to set-inclusion. There are at most polynomially many sets of the form $P_1 \times P_2$ and each such set contains at most polynomially many maximal loops because the union of two loops is again a loop. Further, the set $\mathcal{M}_B$ contains a loop P if $P \in \mathcal{M}$ and $\text{pr}_1(P) \in \mathcal{F}_1$. Similarly, $\mathcal{M}_R$ contains a loop P if $P \in \mathcal{M}$ and $\text{pr}_2(P) \in \mathcal{F}_2$.

▶ **Lemma 13.** *Let S be an SCC of $A_1 \times A_2$ and $m \in \mathbb{N}$, $m \geq 1$.*
There is an $\mathcal{F}_B$-start ($\mathcal{F}_R$-start) m-chain in S with respect to $(\mathcal{F}_B, \mathcal{F}_R)$ iff
there is an $\mathcal{M}_B$-start ($\mathcal{M}_R$-start) $(m-1)$-chain in S with respect to $(\mathcal{M}_B, \mathcal{M}_R)$.

Proof. "⇒" Let $c = (P_1, \ldots, P_m)$ be an $\mathcal{F}_B$-start m-chain with respect to $(\mathcal{F}_B, \mathcal{F}_R)$. For $1 \leq i < m$ and $b = i \bmod 2$ the set $\mathrm{pr}_{2-b}(P_i) \times \mathrm{pr}_{1+b}(P_{i+1})$ contains a maximal loop P'_i with $\mathrm{pr}_{2-b}(P'_i) = \mathrm{pr}_{2-b}(P_i)$ because $P_i \subseteq \mathrm{pr}_{2-b}(P_i) \times \mathrm{pr}_{1+b}(P_{i+1})$ can be extended to such a maximal loop. Because $P_i \subseteq P_{i+1}$ we have $P'_i \subseteq P'_{i+1}$ for $1 \leq i < m$. So, $c' = (P'_1, \ldots, P'_{m-1})$ is an $(m-1)$-chain with respect to $(\mathcal{M}_B, \mathcal{M}_R)$. Further, $P_1 \in \mathcal{F}_B$ iff $P'_1 \in \mathcal{M}_B$, so c' is $\mathcal{M}_B$-start.

"⇐" Let $c = (P_1, \ldots, P_{m-1})$ be an $\mathcal{M}_B$-start $(m-1)$-chain with respect to $(\mathcal{M}_B, \mathcal{M}_R)$. Every chain with respect to $(\mathcal{M}_B, \mathcal{M}_R)$ is a chain with respect to $(\mathcal{F}_B, \mathcal{F}_R)$ because $\mathcal{M}_B \subseteq \mathcal{F}_B$ and $\mathcal{M}_R \subseteq \mathcal{F}_R$. So, c is an $(m-1)$-chain with respect to $(\mathcal{F}_B, \mathcal{F}_R)$. Further, c is $\mathcal{F}_B$-start.

Consider the case $P_{m-1} \in \mathcal{M}_B$. By the definition of $\mathcal{M}$ there are $R_1 \in \mathcal{F}_1, R_2 \in \mathcal{F}_2$ such that $P_{m-1} \subseteq R_1 \times R_2$. So, there is a loop P_m with $P_{m-1} \subseteq P_m \subseteq Q_1 \times R_2$ and $\mathrm{pr}_2(P_m) = R_2$. Thus, $P_m \in \mathcal{M}_R$ and $(P_1, \ldots, P_{m-1}, P_m)$ is an $\mathcal{F}_B$-start m-chain with respect to $(\mathcal{F}_B, \mathcal{F}_R)$.

The case that c is $\mathcal{M}_R$-start follows analogously. ◀

▶ **Theorem 14.** *The problem* WagnerSeparation *can be decided in polynomial time.*

Proof. According to Lemma 13, it suffices to determine the superchains with respect to $(\mathcal{M}_B, \mathcal{M}_R)$. This can be done in polynomial time as mentioned in Section 5.3. ◀

The intersection of two DMAs can cause an exponential blowup [2]. In contrast to this, Lemma 14 implies that the disjointness (emptiness of the intersection) of two DMAs can be checked in polynomial time.

▶ **Corollary 15.** *Given two DMAs $\mathcal{A}_1$, $\mathcal{A}_2$, it can be decided in polynomial time whether $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ is empty.*

Proof. $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) \neq \emptyset$ iff there are arbitrarily long chains in $A_1 \times A_2$ with respect to $(\mathcal{F}_B, \mathcal{F}_R)$ iff $L(\mathcal{A}_1)$, $L(\mathcal{A}_2)$ are not E^1_m-separable for $m = |Q_1| \cdot |Q_2| + 1$. The proof of these equivalences mirrors the proof of Remark 1. ◀

5.4 Wagner Separation for Parity Automata

In this section we consider the Wagner separation problem with deterministic parity automata as input. A *deterministic parity automaton* (DPA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Omega)$ where $A = (Q, \Sigma, \delta, q_0)$ is an automaton structure and $\Omega : Q \to \mathbb{N}$ is the priority function of $\mathcal{A}$. An infinite word $\alpha \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if the maximal priority seen infinitely often by the run ρ of A in α is even. That is, $\max(\{\Omega(q) \mid q \in \mathrm{Inf}(\rho)\})$ is even. For a set $P \subseteq Q$ let $\Omega(P) = \{\Omega(p) \mid p \in P\}$. Since every DPA can be transformed into an equivalent DPA with $\Omega(Q) = \{0, \ldots, |Q|\}$ we define the size of a DPA $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Omega)$ as $|Q|$.

An ω-language L is regular iff there is a DPA $\mathcal{A}$ with $L(\mathcal{A}) = L$, so DMAs and DPAs define the same class of languages. However, there are languages for which every DMA is exponentially larger than the smallest DPA for the language and there are languages for which every DPA is exponentially larger than the smallest DMA for the language [1]. So, it makes a difference with respect to computational complexity whether a language is given as a DMA or as a DPA.

WagnerSeparationParity
Given: Two DPAs $\mathcal{A}_1, \mathcal{A}_2$, $X \in \{C, D, E\}$ and $m, n \in \mathbb{N}$.
Decide: Are $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ X^n_m-separable?

As in the previous section, we use maximal chains to show that this problem can be solved in polynomial time. This implies that disjointness of two languages given as DPA can be decided in polynomial time, too.

▶ **Theorem 16.** *WagnerSeparationParity can be decided in polynomial time.*

Proof. Let $\mathcal{A}_1, \mathcal{A}_2$ be two DPAs, $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, q_i, \Omega_i)$ and A_i the corresponding automaton structure for $i \in \{1, 2\}$. Consider their product automaton and the acceptance conditions $\mathcal{F}_B = \{P \subseteq Q_1 \times Q_2 \mid \max(\Omega_1(\mathrm{pr}_1(P)))$ is even and P is a loop$\}$ and $\mathcal{F}_R = \{P \subseteq Q_1 \times Q_2 \mid \max \Omega_2(\mathrm{pr}_2(P)))$ is even and P is a loop$\}$. As shown in Section 5.3 the chain structure with respect to $(\mathcal{F}_B, \mathcal{F}_R)$ suffices to decide separability. However, $\mathcal{F}_1$ and $\mathcal{F}_2$ might be of exponential size.

For $q, q' \in Q_1 \times Q_2$, $k_1, k_2 \in \mathbb{N}$ we denote with $q \xrightarrow[\leq(k_1,k_2)]{} q'$ that there is a word $w \in \Sigma^*$ and a run ρ of $A_1 \times A_2$ on w from q to q' such that for all $0 \leq j \leq |\rho|$ we have $\mathrm{pr}_1(\rho(j)) \leq k_1$ and $\mathrm{pr}_2(\rho(j)) \leq k_2$. For $q \in Q_1 \times Q_2$, $k_1, k_2 \in \Omega(Q)$ consider the set $P_q^{k_1,k_2} = \{q' \mid q \xrightarrow[\leq(k_1,k_2)]{} q' \xrightarrow[\leq(k_1,k_2)]{} q\}$. Consider $\mathcal{M}_B = \{P_q^{k_1,k_2} \neq \emptyset \mid q \in Q_1 \times Q_2, k_1, k_2 \in \Omega(Q)$ and k_1 is even$\}$ and $\mathcal{M}_R = \{P_q^{k_1,k_2} \neq \emptyset \mid q \in Q_1 \times Q_2, k_1, k_2 \in \Omega(Q)$ and k_2 is even$\}$. These sets can be computed in polynomial time.

Let S be an SCC of $A_1 \times A_2$. We show that there is an m-chain c in S with respect to $(\mathcal{M}_B, \mathcal{M}_R)$ iff there is an m-chain c' in S with respect to $(\mathcal{F}_B, \mathcal{F}_R)$. Further, we show that c is $\mathcal{M}_B$-start iff c' is $\mathcal{F}_B$-start.

"⇒" We have $\mathcal{M}_B \subseteq \mathcal{F}_B$ and $\mathcal{M}_R \subseteq \mathcal{F}_R$, so every m-chain with respect to $(\mathcal{M}_B, \mathcal{M}_R)$ is an m-chain with respect to $(\mathcal{F}_B, \mathcal{F}_R)$ and it is $\mathcal{M}_B$-start iff it is $\mathcal{F}_B$-start.

"⇐" Let $c' = (P'_1, \ldots, P'_m)$ an m-chain in $A_1 \times A_2$ with respect to $(\mathcal{F}_B, \mathcal{F}_R)$. Let $p \in P'_1$, $k_1^i = \max(\Omega_1(\mathrm{pr}_1(P'_i)))$ and $k_2^i = \max(\Omega_2(\mathrm{pr}_2(P'_i)))$. Then $c = (P_p^{k_1^1,k_2^1}, \ldots, P_p^{k_1^m,k_2^m})$ is an m-chain with respect to $(\mathcal{M}_B, \mathcal{M}_R)$. Further, c is $\mathcal{M}_B$ start iff c' is $\mathcal{F}_B$-start and the chains are in the same SCC.

Thus, the chain structure with respect to $(\mathcal{M}_B, \mathcal{M}_R)$ is the same as it is with respect to $(\mathcal{F}_B, \mathcal{F}_R)$. So, separability can be decided using $(\mathcal{M}_B, \mathcal{M}_R)$ as done in Section 5.3. ◀

6 Conclusion

We have seen that separation of two languages given by two DMAs with respect to the Wagner hierarchy can be viewed as analyzing the loop structure in their product automaton. Using this result one can compute a separator in exponential time. We showed that there are languages of DMAs whose separator requires exponential size. So, if one wants to compute the complete separator then exponential time is optimal. However, we can decide separation with respect to the Wagner hierarchy in polynomial time using maximal loops. This implies that we can decide disjointness of two languages given as DMAs in polynomial time. A variation of the separation problem where the languages are given as DPAs can be solved in polynomial time as well.

A separating DMA can be large because it has to list all accepting loops explicitly. Meanwhile, we can decide separation efficiently because we can restrict ourselves to maximal loops. These maximal loops, in a sense, give a more succinct representation of the acceptance condition of a DMA.

In a follow-up paper we will investigate a new automaton model based on this succinct representation. We hope that this new model has better properties with respect to computational complexity than current automata models.

References

1 Udi Boker. On the (in) succinctness of Muller automata. In *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

2 Udi Boker. Why these automata types? In *LPAR*, volume 18, pages 143–163, 2018.

3 Olivier Carton and Ramón Maceiras. Computing the Rabin index of a parity automaton. *RAIRO-Theoretical Informatics and Applications*, 33(6):495–505, 1999.

4 Thomas Colcombet and Christof Löding. The nesting-depth of disjunctive μ-calculus for tree languages and the limitedness problem. In *International Workshop on Computer Science Logic*, pages 416–430. Springer, 2008.

5 Damian Niwiński and Igor Walukiewicz. Relating hierarchies of word and tree automata. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 320–331. Springer, 1998.

6 Damian Niwiński and Igor Walukiewicz. Deciding nondeterministic hierarchy of deterministic tree automata. *Electronic Notes in Theoretical Computer Science*, 123:195–208, 2005.

7 Dominique Perrin and Jean-Éric Pin. *Infinite words: automata, semigroups, logic and games*. Academic Press, 2004.

8 Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10, 2014.

9 Thomas Place and Marc Zeitoun. The tale of the quantifier alternation hierarchy of first-order logic over words. *ACM SIGLOG News*, 2(3):4–17, 2015.

10 Shmuel Safra. *Complexity of automata on infinite objects*. PhD thesis, The Weizmann Institute of Science, 1989.

11 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control.*, 8(2):190–194, 1965.

12 Klaus Wagner. On ω-regular sets. *Information and control*, 43(2):123–177, 1979.

Normal Sequences with Non-Maximal Automatic Complexity

Liam Jordon[1]

Department of Computer Science, Maynooth University, Maynooth, Ireland

Philippe Moser

Department of Computer Science, Maynooth University, Maynooth, Ireland

Abstract

This paper examines Automatic Complexity, a complexity notion introduced by Shallit and Wang in 2001 [29]. We demonstrate that there exists a normal sequence T such that $I(T) = 0$ and $S(T) \leq 1/2$, where $I(T)$ and $S(T)$ are the lower and upper automatic complexity rates of T respectively. We furthermore show that there exists a Champernowne sequence C, i.e. a sequence formed by concatenating all strings of length one followed by concatenating all strings of length two and so on, such that $S(C) \leq 2/3$.

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

Keywords and phrases Automatic Complexity, finite-state complexity, normal sequences, Champernowne sequences, de Bruijn strings, Kolmogorov complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.47

Related Version *Previous Version*: `https://arxiv.org/abs/2107.05979`

Funding *Liam Jordon*: Supported by the Irish Research Council's Government of Ireland Postgraduate Scholarship Programme. Grant number: GOIPG/2017/1200.

1 Introduction

Due to the uncomputability of Kolmogorov complexity, finite-state automata and transducers have acted as a popular setting to study the complexity of finite strings and infinite sequences. In this paper we examine the finite-state based complexity introduced by Shallit and Wang, which is analogous to Sipser's *Distinguishing Complexity* [30], known as *Automatic Complexity* [29]. For a string x of length n, its automatic complexity $A(x)$ is defined to be the minimum number of states required by any deterministic finite-state automaton such that x is the only string of length n the automaton accepts. A non-deterministic variation was first examined by Hyde [15]. In their paper, Shallit and Wang found upper and lower bounds for the automatic complexity of various sets of strings and of prefixes of the infinite Thue-Morse sequence. Expanding on this line of research, Kjos-Hanssen has recently studied the automatic complexity of Fibonacci and Tribonacci sequences [19].

We continue this line of research by examining the automatic complexity of some *normal* sequences. A binary sequence is normal number in the sense of Borel [3] if for all n, every string of length n occurs as a substring in the sequence with limiting frequency 2^{-n}. The complexity of normal sequences has been widely studied in the finite-state setting for many years and a review of several old and new results can be found in [21].

Depending on how finite-state complexity is measured, normal sequences may have high or low complexity. For instance, if complexity is defined as compressibility by lossless finite-state compressors, normal sequences have maximum complexity. For example, combining results

[1] Corresponding author

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 47; pp. 47:1–47:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of Schnorr and Stimm [28] and Dai, Lathrop, Lutz and Mayordomo [10] demonstrates that a sequence is normal if and only if it cannot be compressed by any lossless finite-state compressor (see [2] for a proof). This is also true when the finite-state compressor is equipped with a counter [1] or when the reading head is allowed to move in two directions [8]. Another definition examines the length of the minimal input required to output a string via finite-state transducers and has been used in [5, 7, 11, 12, 17]. It was demonstrated in Theorem 24 of [7] that in a complexity based on this approach, one can construct normal sequences with minimal complexity.

As automatic complexity is more of a "combinatorial" rather than an "information content" measurement[2], this leads to the question as to how low can the automatic complexity of normal sequences be? Previously the automatic complexity of finite strings produced by linear feedback shift registers which have a maximal number of distinct substrings (otherwise known as m-sequences) [20] along with sequences and finite strings which do not contain k-powers, i.e. substrings of the form x^k, have been studied [16, 19, 29]. Normal sequences by definition contain x^k as a substring infinitely often for every possible pair (x, k). Is there a trade-off between the randomness of normal sequences resulting in high complexity, in the sense that they contain every string as a substring infinitely often, and the fact that some of those substrings have the form x^k which results in low automatic complexity?

We explore this question by constructing a normal sequence T whose upper automatic complexity rate $S(T)$ is bounded above by $1/2$ and whose lower automatic complexity rate $I(T)$ is 0. We then study a specific class of normal sequences known as *Champernowne* sequences [9], i.e. sequences formed by concatenating all strings of length one followed by all strings of length two and so on. It is widely known that Champernowne sequences are incompressible by the Lempel-Ziv 78 algorithm and results by Lathrop and Strauss show that all sequences incompressible by Lempel-Ziv 78 are normal [22]. Due to this restriction on their construction, one may expect Champernowne sequences to have high automatic complexity. However, we demonstrate that there exists a Champernowne sequence C built via a method presented by Pierce and Shields in [25] that satisfies $S(C) \leq 2/3$. It has previously been seen that Champernowne sequences built via their method are compressible by a variation of the Lempel-Ziv 77 [25] and the PPM* [18] compression algorithms.

2 Preliminaries

We work with the binary alphabet $\{0,1\}$ in this paper. A finite *string* is an element of $\{0,1\}^*$. A *sequence* is an element of $\{0,1\}^\omega$. $\{0,1\}^{\leq\omega}$denotes the set $\{0,1\}^* \bigcup \{0,1\}^\omega$. The length of a string x is denoted by $|x|$. We say $|S| = \omega$ for $S \in \{0,1\}^\omega$. λ denotes the string of length 0. $\{0,1\}^n$ denotes the set of strings of length n. For $x \in \{0,1\}^{\leq\omega}$ and $0 \leq i < |x|$, $x[i]$ denotes the $(i+1)^{\text{th}}$ bit of x with $x[0]$ being the first bit. For $x \in \{0,1\}^{\leq\omega}$ and $0 \leq i \leq j < |x|$, $x[i..j]$ denotes the *substring* of x consisting of its $(i+1)^{\text{th}}$ through $(j+1)^{\text{th}}$ bits. For $x \in \{0,1\}^*$ and $y \in \{0,1\}^{\leq\omega}$, xy (sometimes written as $x \cdot y$) denotes the string (or sequence) x concatenated with y. For a string x, x^n denotes x concatenated with itself n times. For $x \in \{0,1\}^{\leq\omega}$, a substring y of x is called a k-*power* if $y = u^k$ for some string u. For $x \in \{0,1\}^*$ and $y, z \in \{0,1\}^{\leq\omega}$ such that $z = xy$, we call x a *prefix* of z and y a *suffix* of z. We write $x[i..]$ to denote the suffix of x beginning with its $(i+1)^{\text{th}}$ bit. The *lexicographic-length* ordering of $\{0,1\}^*$ is defined by saying for two strings x, y, x comes before y if either $|x| < |y|$ or else $|x| = |y|$ with $x[n] = 0$ and $y[n] = 1$ for the smallest n such that $x[n] \neq y[n]$.

[2] In an information content measurement, we would like there to be roughly 2^n objects with a complexity of n while it trivially holds that all strings of the form 0^n and 1^n have an automatic complexity of 2.

Given strings x, w we use the following notation to count the number of times w occurs as a substring in x. The number of occurrences of w as a substring of x is given by

$$\mathrm{occ}(w,x) = |\{i : x[i..i+|w|-1] = w\}|.$$

The block number of occurrences of w as a substring of x is given by

$$\mathrm{occ}_b(w,x) = |\{i : x[i..i+|w|-1] = w \wedge i \equiv 0 \bmod |w|\}|.$$

For example, $\mathrm{occ}(00, 0000) = 3$ while $\mathrm{occ}_b(00, 0000) = 2$.
Automatic complexity is based on finite automata.

▶ **Definition 1.** *A* deterministic finite-state automaton (DFA) *is a 4-tuple* $M = (Q, q_0, \delta, F)$*, where*

- Q *is a non-empty, finite set of* states*,*
- $q_0 \in Q$ *is the* initial state*,*
- $\delta : Q \times \{0,1\} \to Q$ *is the* transition function*,*
- $F \subseteq Q$ *is the set of* final / accepting states*.*

A DFA M can be thought of as a function $M : \{0,1\}^* \to Q$ such that for all $x \in \{0,1\}^*$ and $b \in \{0,1\}$, M is defined by the recursion $M(\lambda) = q_0$ and $M(xb) = \delta(M(x), b)$. If $M(x) \in F$, we say M *accepts* x. We write $L(M)$ to denote the *language* of M, i.e. the set of strings that M accepts.

Shallit and Wang define *automatic complexity* as follows.

▶ **Definition 2** ([29])**.** *Let* $x \in \{0,1\}^*$*. The automatic complexity of* x*, denoted by* $A(x)$*, is the minimal number of states required by any DFA* M *such that* $L(M) \bigcap \{0,1\}^{|x|} = \{x\}$*.*

We say a DFA M *uniquely accepts* a string x if $L(M) \bigcap \{0,1\}^{|x|} = \{x\}$.

Shallit and Wang compute the following two ratios to examine the automatic complexity of sequences.

▶ **Definition 3.** *The* lower *and* upper *rates for the automatic complexity of a sequence* T *are respectively given by*

$$I(T) = \liminf_{m\to\infty} \frac{A(T[0..m])}{m+1} \text{ and, } S(T) = \limsup_{m\to\infty} \frac{A(T[0..m])}{m+1}.$$

From the fact that for all $x \in \{0,1\}^*$ it trivially holds that $A(x) \le |x| + 2$, it follows that for all $T \in \{0,1\}^\omega$, $0 \le I(T) \le S(T) \le 1$.

Normal sequences and *de Bruijn* strings which we use to build *normal* sequences are defined as follows.

▶ **Definition 4.** *A sequence* $T \in \{0,1\}^\omega$ *is normal if for all* $x \in \{0,1\}^*$*,*

$$\lim_{m\to\infty} \frac{\mathrm{occ}(x, T[0..m])}{m+1} = 2^{-|x|}.$$

▶ **Definition 5** ([4, 27])**.** *A* de Bruijn string *of order* n *is a string* $u \in \{0,1\}^{2^n}$ *such that for all* $w \in \{0,1\}^n, \mathrm{occ}(w, u \cdot u[0..n-2]) = 1$*.*

For example, 0011 and 00010111 are de Bruijn strings of order 2 and 3 respectively. We generally use d_n to denote a de Bruijn string of order n. It is known that there are $2^{2^{n-1}-n}$ de Bruijn strings of order n unique up to cycling, i.e. the two de Bruijn strings 0011 and 0110 are considered the same string for example when counting.

3 Normal Sequences with Low Automatic Complexity

In our first result we construct a normal sequence T such that $I(T) = 0$, that is, infinitely many prefixes have close to minimal automatic complexity. We furthermore show that $S(T) \leq 1/2$, indicating that the sequence does not have high complexity. We require the following variation of a result by Nandakumar and Vangapelli.

▶ **Theorem 6** ([24]). *Let $f : \mathbb{N} \to \mathbb{N}$ be increasing such that for all n, $f(n) \geq n^n$. Then every sequence of the form $T = d_1^{f(1)} d_2^{f(2)} d_3^{f(3)} \cdots$ where d_n is a de Bruijn string of order n, is normal* [3].

▶ **Theorem 7.** *There is a normal sequence T such that $I(T) = 0$ and $S(T) \leq \frac{1}{2}$.*

Proof. We recursively define the sequence $T = T_1 T_2 \ldots$ and the function $f : \mathbb{N} \to \mathbb{N}$ as follows. For all $j \geq 1$, let d_j be a de Bruijn string of order j such that if j is odd, d_j begins with a 1 and if j is even, d_j begins with a 0. We set $f(1) = 2$ and $T_1 = d_1^{f(1)} = d_1^2$. For $j \geq 2$, we define $f(j) = |T_1 \ldots T_{j-1}|^{|T_1 \ldots T_{j-1}|}$ and $T_j = d_j^{f(j)}$. Note that $f(1) > 1$ and for all $j \geq 2$, $f(j) \geq |T_{j-1}|^{|T_{j-1}|}$ and that $|T_{j-1}| = 2^{j-1} f(j-1) \geq j$. Hence by Theorem 6, T is normal. For simplicity, we write $\overline{T_j}$ for the prefix $T_1 \cdots T_j$ of T.

We first show that $I(T) = 0$. Consider a prefix of the form $\overline{T_n}$. $\overline{T_n}$ is uniquely accepted by the DFA M_1 which has a state for each bit of $\overline{T_{n-1}}$ followed by a loop of length 2^n for the string d_n whose root state is the only accepting state, and an error state. M_1 can be seen in Figure 1. M_1 has $|\overline{T_{n-1}}| + 2^n + 1$ states. Thus we have that

$$\begin{aligned}\frac{A(\overline{T_n})}{|\overline{T_n}|} &\leq \frac{|\overline{T_{n-1}}| + 2^n + 1}{|T_n| + |\overline{T_{n-1}}|} = \frac{|\overline{T_{n-1}}| + 2^n + 1}{2^n f(n) + |\overline{T_{n-1}}|} \\ &\leq \max\left\{\frac{|\overline{T_{n-1}}|}{2^n |\overline{T_{n-1}}|^{|\overline{T_{n-1}}|}}, \frac{2^n + 1}{|\overline{T_{n-1}}|}\right\} \leq \max\left\{\frac{1}{2^n}, \frac{2^n + 1}{(n-1)^{n-1}}\right\}.\end{aligned}$$

Hence it follows that $I(T) = 0$.

Next consider an arbitrary prefix $T[0..m]$ of T. Let n be largest such that $\overline{T_n}$ is a prefix of $T[0..m]$ but $\overline{T_{n+1}}$ is not. Thus $T[0..m] = \overline{T_n} \cdot w$ for some $w \in \{0,1\}^*$ and is uniquely accepted by the DFA M_2 in Figure 1. M_2 has a state for each bit of $\overline{T_{n-1}}$, followed by a loop of length 2^n for the string d_n, followed by a state for each bit of w and an error state. M_2 has $|\overline{T_{n-1}}| + 2^n + |w| + 1$ states.

Consider when $1 \leq |w| \leq 2^n(f(n) - 1) + 2^{n+1}$. Note that

$$|T_{n+1}| = 2^{n+1} f(n+1) = 2^{n+1}(f(n+1) - 1) + 2^{n+1} > 2^n(f(n) - 1) + 2^{n+1}$$

so w can be this length. Note also that M_2 has at most $|\overline{T_n}| + 2^{n+1} + 1$ states for such w.

[3] Nandakumar and Vangapelli's original result was for when $f(n) = n^n$. However, their argument easily carries over for $f(n) \geq n^n$ also and this fact has been used by other authors such as in [6, 7].

Hence for such w we have that

$$\begin{aligned}\frac{A(T[0..m])}{m+1} &\leq \frac{|\overline{T_{n-1}}| + 2^n + |w| + 1}{|\overline{T_n}| + |w|}\\ &\leq \frac{|\overline{T_{n-1}}| + 2^n + 2^n(f(n)-1) + 2^{n+1} + 1}{|\overline{T_n}| + 2^n(f(n)-1) + 2^{n+1}}\\ &= \frac{|\overline{T_{n-1}}| + |T_n| + 2^{n+1} + 1}{|\overline{T_{n-1}}| + 2|T_n| + 2^n}\\ &= \frac{|\overline{T_{n-1}}| + 2^n(|\overline{T_{n-1}}|^{|\overline{T_{n-1}}|} + 2) + 1}{|\overline{T_{n-1}}| + 2(2^n|\overline{T_{n-1}}|^{|\overline{T_{n-1}}|}) + 2^n}\\ &\leq \max\left\{\frac{1 + 2^n(|\overline{T_{n-1}}|^{|\overline{T_{n-1}}|-1} + 2|\overline{T_{n-1}}|^{-1})}{1 + 2(2^n|\overline{T_{n-1}}|^{|\overline{T_{n-1}}|-1})}, \frac{1}{2^n}\right\}. \qquad (1)\end{aligned}$$

Note Equation (1) approaches $1/2$ as n increases.

Furthermore consider when $2^n(f(n)-1) + 2^{n+1} < |w| \leq |T_{n+1}|$. Instead of looping on d_n, it becomes more beneficial to loop on d_{n+1} via a DFA similar to M_1 in Figure 1 where the accepting state is a single state in the loop depending on the length of w. Thus for such prefixes $A(\overline{T_n} \cdot w) \leq |\overline{T_n}| + 2^{n+1} + 1$, i.e. it does not depend on w. Hence the ratio $A(T[0..m])/(m+1)$ decreases and approaches $I(T)$ for such w.

Therefore, by Equation (1), $S(T) \leq \frac{1}{2}$. ◀

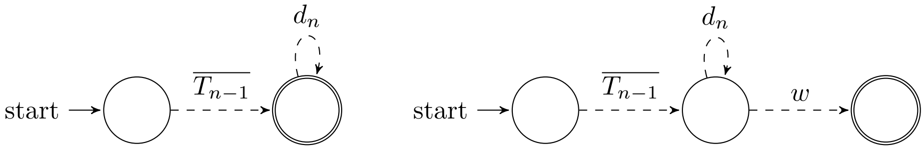


Figure 1 DFA M_1 (left) and M_2 (right) from Theorem 7. The error state (the state traversed to if the bit seen is not the expected bit) and arrows to it are not included. By the construction of T, $d_n[0] \neq w[0]$ to ensure determinism.

4 Automatic Complexity of Champernowne Sequences

In this section we present a Champernowne sequence with an upper automatic complexity rate bounded above by 2/3.

▶ **Definition 8.** *A sequence C is a* Champernowne *sequence if $C = C_1C_2C_3\ldots$, such that for each n, C_n is a concatenation of all strings of length n exactly once. That is, for all* $x \in \{0,1\}^n$, $\mathrm{occ}_b(x, C_n) = 1$.

Unlike Champernowne's original sequence which was a concatenation of all strings in lexicogrpahic-length order (0100011011000...), we emphasise that the set of Champernowne sequences do not require strings to be in length-lexicographic order for the construction. There are $2^n!$ possible choices for zone C_n in a Champernowne sequence. For instance, 00011011 and 11100001 are two possibilities for C_2.

We now describe Pierce and Shields' construction of Champernowne sequences from [25]. Suppose we wished to construct substring C_n. Let d_n be a de Bruijn string of order n. For $0 \leq j \leq 2^n - 1$, let $d_{n,j}$ represent a cyclic shift to the left of the first j bits of d_n. That is,

$d_{n,j} = d_n[j..2^n - 1] \cdot d_n[0..j-1]$. We write d_n instead of $d_{n,0}$ when no cyclic shift occurs. Note that each n can be written uniquely in the form $n = 2^s t$ where $s \geq 0$ and $t \geq 1$ where t is odd. Each substring C_n is broken into further substrings $C_n = B_{n,0} \cdot B_{n,1} \cdots B_{n,2^s-1}$ where $B_{n,j}$ is a concatenation of $d_{n,j}$ with itself t times. That is, $B_{n,j} = (d_{n,j})^t$. Hence, for example, if n is odd then $C_n = (d_n)^n$ and if $n = 2^k$ for $k \geq 1$, $C_n = d_n d_{n,1} \cdots d_{n,n-1}$.

To help the reader visualise this, the result of using the lexicographic least de Bruijn strings of order 3, 4 and 6 to build C_3, C_4 and C_6 via Pierce and Shields' method are provided in Figures 2 and 3. An algorithm to construct the lexicographic least de Bruijn strings was first provided by Martin in 1934 which requires exponential space [23]. Later works by Fredricksen, Kessler and Maiorana led to the FKM-algorithm which only requires $O(n)$ space to construct such strings [13, 14].

00010111	0000100110101111
00010111	0001001101011110
00010111	0010011010111100
	0100110101111000

Figure 2 Concatenating the three rows on the left hand side produces the substring C_3 and concatenating the four rows on the right hand side produces the substring C_4 if d_3 and d_4 are chosen to be the least lexicographic de Bruijn string of their order respectively.

```
0000001000011000101000111001001011001101001111010101110110111111
0000001000011000101000111001001011001101001111010101110110111111
0000001000011000101000111001001011001101001111010101110110111111
0000010000110001010001110010010110011010011110101011101101111110
0000010000110001010001110010010110011010011110101011101101111110
0000010000110001010001110010010110011010011110101011101101111110
```

Figure 3 Concatenating the six rows produces the substring C_6 where d_6 is chosen to be the lexicographic least de Bruijn string of order 6. The first three rows are B_0 while the second three rows are B_1.

In Figures 2 and 3 above, the bits shaded in blue indicate the bits of each zone read on a single traversal of the loops described in the proof of Theorem 11 and shown in Figure 5 in the case where either n or $n+1$ is 3, 4 or 6 respectively.

We re-present Pierce and Shields' proof that their construction builds Champernowne sequences using our notation below. The proof requires some basic results and definitions which can be seen in an undergraduate group theory course. We omit specifics as they are unimportant to the paper as a whole but point towards [26] for those interested.

▶ **Lemma 9** ([25]). *Let $C \in \{0,1\}^\omega$ be constructed via Pierce and Shields' construction. Then C is a Champernowne sequence.*

Proof. Let $C \in \{0,1\}^\omega$ be as described. In order to show that C is a Champernowne sequence we must show that for each zone C_n, for all $x \in \{0,1\}^n$, $\mathrm{occ}_b(x, C_n) = 1$.

Consider substring C_n. Let G_{2^n} be the cyclic group of order 2^n, i.e. $G_{2^n} = \langle x \,|x^{2^n} = e \rangle$, where $e = x^0$ is the identity element and x is the generator of the group. There exists a bijective mapping $f : G_{2^n} \to \{0,1\}^n$ such that for $0 \leq a < 2^n$, x^a is mapped to the substring of d_n of length n beginning in position a when d_n is viewed cyclically. That is, $f(e) = d_n[0..n-1]$, $f(x) = d_n[1..n], \ldots f(x^{2^n-1}) = d_n[2^n - 1] \cdot d_n[0..n-2]$.

Let $s \geq 0$ and $t \geq 1$ where t is odd such that $n = 2^s t$. Consider the subgroup $\langle x^n \rangle$ of G_{2^n}. From group theory it follows that

$$|\langle x^n \rangle| = \frac{2^n}{\gcd(n, 2^n)} = 2^{2^s t - s} = 2^{n-s}.$$

So

$$\langle x^n \rangle = \bigcup_{i=0}^{2^{n-s}-1} \{x^{in \bmod 2^n}\} = \{e, x^n, x^{2n}, \ldots x^{(2^{n-s}-1)n \bmod 2^n}\}.$$

Concatenating the result of applying f to each element of $\langle x^n \rangle$ beginning with e in the natural order gives the string

$$\sigma = f(e) \cdot f(x^n) \cdot f(x^{2n}) \cdots f(x^{(2^{n-s}-1)n \bmod 2^n}).$$

σ can be thought of as beginning with the prefix of d_n of length n, cycling through d_n in blocks of size n until the block containing d_n's suffix of length n is seen. As $(2^{n-s}n)/2^n = t$, we have that $\sigma = (d_n)^t = B_0$.

As $|G_{2^n}|/|\langle x^n \rangle| = 2^s$, there are 2^s cosets of $\langle x^n \rangle$ in G_{2^n}. As cosets are disjoint, each represents a different set of 2^{n-s} strings of $\{0,1\}^n$. Specifically each coset represents some $B_j = (d_{n,j})^t$ block. Therefore, for each $x \in \{0,1\}^n$, for some $j \in \{0, \ldots, 2^s - 1\}$, $\mathrm{occ}_b(x, B_j) = 1$ and $\mathrm{occ}_b(x, B_i) = 0$ for each $i \neq j$. Thus $\mathrm{occ}_b(x, C_n) = 1$. ◀

Before examining the main result of this section, we require the following result from number theory.

▶ **Theorem 10.** *For $a, b, c \in \mathbb{Z}$, consider the Diophantine equation $ax + by = c$. If there exists a solution to the equation (x_0, y_0) where $x_0, y_0 \in \mathbb{Z}$, then all other solutions (x', y') such that $x', y' \in \mathbb{Z}$ are of the form $x' = x_0 + (b/g)d$ and $y' = y_0 - (a/g)d$ where $d \in \mathbb{Z}$ is arbitrary and $g = \gcd(a, b)$.*

Henceforth, we write PSC to denote the set of Champernowne sequences constructed using Pierce and Shields' method such that for each zone C_n of the sequences, a de Bruijn string d_n of order n with prefix 0^n was used to construct it. In the following theorem we show that there exists sequences in PSC which have non-maximal automatic complexity as their upper automatic complexity rates are bounded above by 2/3. We also briefly discuss their lower automatic complexity rates in Section 4.1.

▶ **Theorem 11.** *There exists $C \in \mathrm{PSC}$ such that $S(C) \leq \frac{2}{3}$.*

Proof. Let $C \in \mathrm{PSC}$ and consider an arbitrary prefix $C[0..m]$ of C. Again we use $\overline{C_n}$ to denote the prefix $C_1 C_2 \cdots C_n$. Let n be largest such that $\overline{C_{n+1}}$ is a prefix of $C[0..m]$ but $\overline{C_{n+2}}$ is not.

To examine $A(C[0..m])$ we build automata which make use of two loops which exploit the repetitions of the de Bruijn strings in C_n and C_{n+1}. The automata have a single accepting state which depend on the length of the prefix being examined. There are four cases to consider which are dependent on the the value of n and can be seen in Figure 5.

- **Case 1**: n is a power of 2,
- **Case 2**: $n + 1$ is a power of 2,
- **Case 3**: n is even but not a power of 2,
- **Case 4**: $n + 1$ is even but not a power of 2.

Notation wise, we let v_n be the string such that $d_n = 0^n 1 v_n$. Note that $d_n[n] = 1$ as otherwise the string 0^n would appear twice as a substring of d_n. Also note that the final bit of v_n must be a 1.

Suppose we are in Case 3 where $n = 2^s t$ where $s \geq 1$ and $t \geq 3$ where t is odd. We examine Case 3 as later calculations which maximise the number of states needed require for the possibility that $n+2$ is even but not a power of 2.

Let p_{n+2} denote the prefix of C_{n+2} such that $C[0..m] = \overline{C_{n+1}} p_{n+2}$. The automaton for Case 3 in Figure 5 accepts $C[0..m]$ by reading the prefix $\overline{C_{n-1}} \cdot 0^n$ state by state, then traversing the first loop 2^s times, then reading 0^{2^s+1}, then traversing the second loop fully n times and then up to reading $1v_{n+1}$ on the $n+1^{\text{th}}$ traversal of it. Then, depending on the length of p_{n+2}, we read the final 0^{n+1} of the second loop and exit it to read the remainder of $C_{n+2}[n+1..]$ as needed. That is, if $|p_{n+2}| \leq n+1$ then the final state is contained in the last $n+1$ states (including the root state) of the second loop of the DFA, else once finishing the loop, we traverse through $|p_{n+2}| - (n+1)$ extra states to the accepting state.

To see that $C[0..m]$ is accepted by the DFA, note that the traversal through the DFA described above can be factored as $\overline{C_{n-1}} x p_{n+2}$ where

$$x = 0^n (1 v_n d_n^{t-1} 0^{n-1})^{2^s} 0^{2^s+1} (1 v_{n+1} 0^{n+1})^n (1 v_{n+1}).$$

Note that $x = C_n C_{n+1}$ since

$$\begin{aligned}
x &= 0^n (1 v_n d_n^{t-1} 0^{n-1})^{2^s} 0^{2^s+1} (1 v_{n+1} 0^{n+1})^n (1 v_{n+1}) \\
&= (0^n 1 v_n d_n^{t-1}) 0^{n-1} (1 v_n d_n^{t-1} 0^{n-1})^{2^s-1} 0^{2^s+1} (1 v_{n+1} 0^{n+1})^n (1 v_{n+1}) \\
&= B_0 (0^{n-1} 1 v_n d_n^{t-1} 0) 0^{n-2} (1 v_n d_n^{t-1} 0^{n-1})^{2^s-1} 0^{2^s+1} (1 v_{n+1} 0^{n+1})^n (1 v_{n+1}) \\
&= B_0 B_1 (0^{n-2} 1 v_n d_n^{t-1} 0^2) 0^{n-3} (1 v_n d_n^{t-1} 0^{n-1})^{2^s-2} 0^{2^s+1} (1 v_{n+1} 0^{n+1})^n (1 v_{n+1}) \\
&\cdots \qquad\qquad \text{(Keep repeating this process of sectioning into the } 2^s \text{ blocks)} \\
&= B_0 B_1 \cdots B_{2^s-1} 0^{n-2^s} 0^{2^s+1} (1 v_{n+1} 0^{n+1})^n (1 v_{n+1}) \\
&= C_n (0^{n+1} 1 v_{n+1})^{n+1} = C_n C_{n+1}.
\end{aligned}$$

Next we show that the DFA uniquely accepts $C[0..m]$. Note that all strings accepted have length

$$|\overline{C_{n-1}}| + n + (2^n t - 1)a + 2^s + 1 + 2^{n+1} b + |p_{n+2}| - (n+1)$$

where $a \geq 0$ and $b \geq 1$ if $|p_{n+1}| < (n+1)$, else b can possibly be 0 too. As stated $(a,b) = (2^s, n+1)$ is a solution to the Diophantine equation

$$|\overline{C_{n-1}}| + n + (2^n t - 1)a + 2^s + 1 + 2^{n+1} b + |p_{n+2}| - (n+1) = |C[0..m]|. \tag{2}$$

By Theorem 10, as the first loop has odd length and the second has even length, all solutions to (2) take the form $(2^s + 2^{n+1}c,\, n+1-(2^n t-1)c)$ where $c \in \mathbb{Z}$. As $2^s = n/t$ and $t \geq 3$, we have that $(n+1) - (2^n t - 1)c < 0$ when $c > 0$ and $2^s + 2^{n+1}c < 0$ when $c < 0$, it follows that $c = 0$ is the only possibility that gives non-negative integer solutions, i.e. $C[0..m]$ is uniquely accepted by the DFA.

The number of states of the automaton is bounded above by

$$|\overline{C_{n-1}}| + n + 2^n t + 2^s + 2^{n+1} + |p_{n+2}|.$$

As $2^s \leq n/3$, we then have that

$$\frac{A(C[0..m])}{m+1} \leq \frac{|\overline{C_{n-1}}| + 2^n t + \frac{4n}{3} + 2^{n+1} + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|}. \tag{3}$$

Hence for $|p_{n+2}| \leq 2^n(n-t) - \frac{n}{3} + 1 + 2^{n+2}(\frac{n+2}{2})$ we find that

$$\begin{aligned}
\frac{A(C[0..m])}{m+1} &\leq \frac{|\overline{C_{n-1}}| + 2^n t + \frac{4n}{3} + 2^{n+1} + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|} \\
&\leq \frac{|\overline{C_{n-1}}| + |C_n| + n + 2^{n+1} + 1 + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - \frac{n}{3} + 1 + 2^{n+2}(\frac{n+2}{2})} \\
&= \frac{|\overline{C_n}| + n + 2^{n+1} + 1 + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - \frac{n}{3} + 1 + 2^{n+2}(\frac{n+2}{2})}. \qquad (4)
\end{aligned}$$

We note that taking the limit of (4) as n increases has a value of 2/3.

However as $|p_{n+2}|$ increases, it becomes more advantageous to use a loop for the repetitions in C_{n+2} as opposed to the loop for C_n (similar to the proof of Theorem 7). Worst case scenario, $n+2$ has the form $2^{s'}t'$ for $s' \geq 1$ and $t' \geq 3$ where t' is odd. This results in an automaton of Case 4 in Figure 5 where the accepting state is one of that states of the second loop. One can show the prefix is uniquely accepted as before with a similar argument. Such an automaton requires at most $|\overline{C_n}| + 2^{n+1} + n + 1 + 2^{n+2}(\frac{n+2}{2})$ states (as $t' \leq (n+2)/2$).

Hence for $j \geq 1$ such that $2^n(n-t) - \frac{n}{3} + 1 + 2^{n+2}(\frac{n+2}{2}) \leq |p_{n+2}| + j < |C_{n+2}|$ we use the automaton from Case 4 and get that

$$\frac{A(C[0..m])}{m+1} \leq \frac{|\overline{C_n}| + n + 2^{n+1} + 1 + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - \frac{n}{3} + 1 + 2^{n+2}(\frac{n+2}{2}) + j}. \qquad (5)$$

One can see that for such p_{n+2}, the number of states of the automaton used to calculate (5) remains constant and the ratio decreases as j increases.

Similar calculations for the other three cases show that none achieve an upper bound greater than 2/3 (see appendix). Therefore $S(C) \leq 2/3$. ◀

4.1 Discussion on Lower Bounds for Champernowne Sequences

Let $C \in \mathrm{PSC}$ satisfy Theorem 11. As part of Theorem 11, we do not provide any insight into the value of $I(C)$. Currently many of the techniques for calculating lower bounds rely on the absence of k-powers (such as proofs in [19, 29]). In particular, Shallit and Wang show that for every x without k-powers, x satisfies $A(x) \geq (|x|+1)/k$. However, as C is a Champernowne sequence, long enough prefixes contain k-powers, i.e. there eventually is a substring x such that $x = u^k$ for some string u.

However, one can easily identify an upper bound for $I(C)$ as follows. Consider prefixes of the form $\overline{C_{n+1}}$ where n is a power of 2, i.e. we are in Case 1. The automaton in Figure 5 for Case 1 where the final state is contained appropriately in the second loop will uniquely accept the prefix and simple calculations give us that $I(C) \leq 1/4$.

One prefix x of C such that $A(x)/|x| < 1/4$ which is in Case 1 is $\overline{C_{65}}$. The automaton for Case 1 in Figure 5 which uniquely accepts $\overline{C_{65}}$ has $n_1 = |\overline{C_{63}}| + 2^{64} + 2^{65} + 128$ states. However, the number of states can be reduced further by using two more loops for zones C_{62} and C_{63} instead of having a state for each of their bits. Consider the DFA $\widehat{M}$ shown in Figure 4:

$\widehat{M}$ reads the prefix $\overline{C_{61}} \cdot 0^{62}$. It then traverses a loop for $1v_{62}(d_{62})^{30}0^{61}$. It then reads 0^3 and enters a loop for the string $(1v_{63}0^{63})^{21}$. Following this it reads $01v_{64}0^{63}$ and then enters a loop for the string $(1v_{64}0^{63})^7$. It then reads 0^{65} and enters a loop for the string $(1v_{65}0^{65})^5$, with the final state being that state of this loop after reading $(1v_{65}0^{65})^4 \cdot 1v_{65}$. $\widehat{M}$ can be thought of as combining the DFAs from Figure 5, but altering the length of the loops for the zones. Strings of length $|\overline{C_{65}}|$ that $\widehat{M}$ accepts satisfy the equation

$$|\overline{C_{61}}| + (2^{62} \cdot 31 - 1)a + (21 \cdot 2^{63})b + (7 \cdot (2^{64} - 1))c + (5 \cdot 2^{65})d + 65 + 2^{64} = |\overline{C_{65}}| \qquad (6)$$

where it must hold that $d \geq 1$.

$a = 2$, $b = 3$, $c = 9$ and $d = 13$ is the only non-negative integer solution to Equation (6) and so $\widehat{M}$ uniquely accepts C_{65}. $\widehat{M}$ has $n_2 = |\overline{C_{61}}| + 31 \cdot 2^{62} + 7 \cdot 2^{63} + 8 \cdot 2^{64} + 5 \cdot 2^{65} + 120$ states which is less than n_1. Hence $\widehat{M}$ gives us that $A(\overline{C_{65}})/|\overline{C_{65}}| < 0.173 < 1/4$.

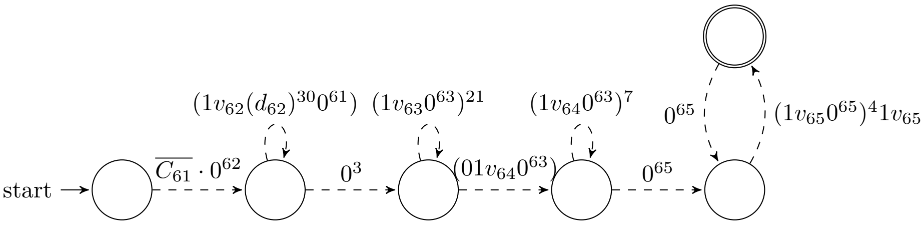

start
$\overline{C_{61}} \cdot 0^{62}$
$(1v_{62}(d_{62})^{30}0^{61})$
0^3
$(1v_{63}0^{63})^{21}$
$(01v_{64}0^{63})$
$(1v_{64}0^{63})^7$
0^{65}
0^{65}
$(1v_{65}0^{65})^4 1v_{65}$

Figure 4 Automaton $\widehat{M}$ for $\overline{C_{65}}$. The dashed arrows represent the missing states belonging to their labels. The error state (the state traversed to if the bit seen is not the expected bit) and arrows to it are not included.

While the above demonstrates that more than two loops can be used, the size of the loops are limited in each case. The following proposition demonstrates that if reading a zone C_j where j is odd, any loop traversed used to read a substring x of C_j, if $|x| \geq j$ then $|x|$ must be a multiple of 2^j.

▶ **Proposition 12.** *Let $C \in \mathrm{PSC}$. Let j be odd and C' be a prefix of C containing the substring C_j. Let $p_0, p_1, \ldots p_{2^j j}$ be the sequence of states an automaton which uniquely accepts C' traverses while reading C_j. If there is some subsequence of states $p_i, \ldots p_{i+l}, \ldots p_{i+2l}$ where $l \geq j$ and for all $0 \leq m \leq l$, $p_{i+m} = p_{i+l+m}$, then l must be a multiple of 2^j.*

Proof. Let C, C' and j be as above. First suppose a loop of length l is traversed where $j \leq l \leq 2^j - 1$. Let x be the substring of C_j read during the loop. Hence $x = yz$ where $|y| = j$ and $|z| \leq 2^j - j - 1$. Suppose the loop is traversed twice in a row indicating that $x^2 = yzyz$ is a substring of C_j. Consider yzy which has length at most $2^j + j - 1$. By the construction of C_j, yzy is a prefix of $d_{j,k} \cdot d_{j,k}[0..j-2]$ for some k. By the nature of de Bruijn stings, $d_{j,k} \cdot d_{j,k}[0..j-2]$ contains every string of length j as a substring exactly once. However, y is a substring of length j contained twice giving us a contradiction.

Next suppose a loop of length l is traversed where $d \cdot 2^j < l < (d+1) \cdot 2^j$ for some $d \leq \lfloor j/2 \rfloor$ as if $d > \lfloor j/2 \rfloor$, traversing the loop twice would result in a string longer than C_n being read. Let x be the string read while traversing the loop. Hence $x = yz$ where $|y| = d \cdot 2^j$ and $1 \leq |z| < 2^j$. Suppose the loop is traversed twice in a row indicating that $x^2 = yzyz$ is a substring of C_j. By the construction of C_j, $yzy = (d_{j,k})^d z (d_{j,k})^d$ for some k where z is a prefix of $d_{j,k}$. This forces $z = \lambda$ or $z = d_{j,k}$ which is a contradiction. ◀

Similar results to the above proposition can be shown for n even also. For instance, if n is a power of 2, loops of length larger than n in zone C_n have to be a multiple of $2^n - 1$. Details can be found in the appendix.

▶ **Question 13.** *Let $C \in \mathrm{PSC}$ satisfy Theorem 11. Is there a limit to the number of beneficial loops we can use to ensure prefixes of C are uniquely accepted? Finding the value of $I(C)$ is left as an open question. For instance, is $I(C) > 0$?*

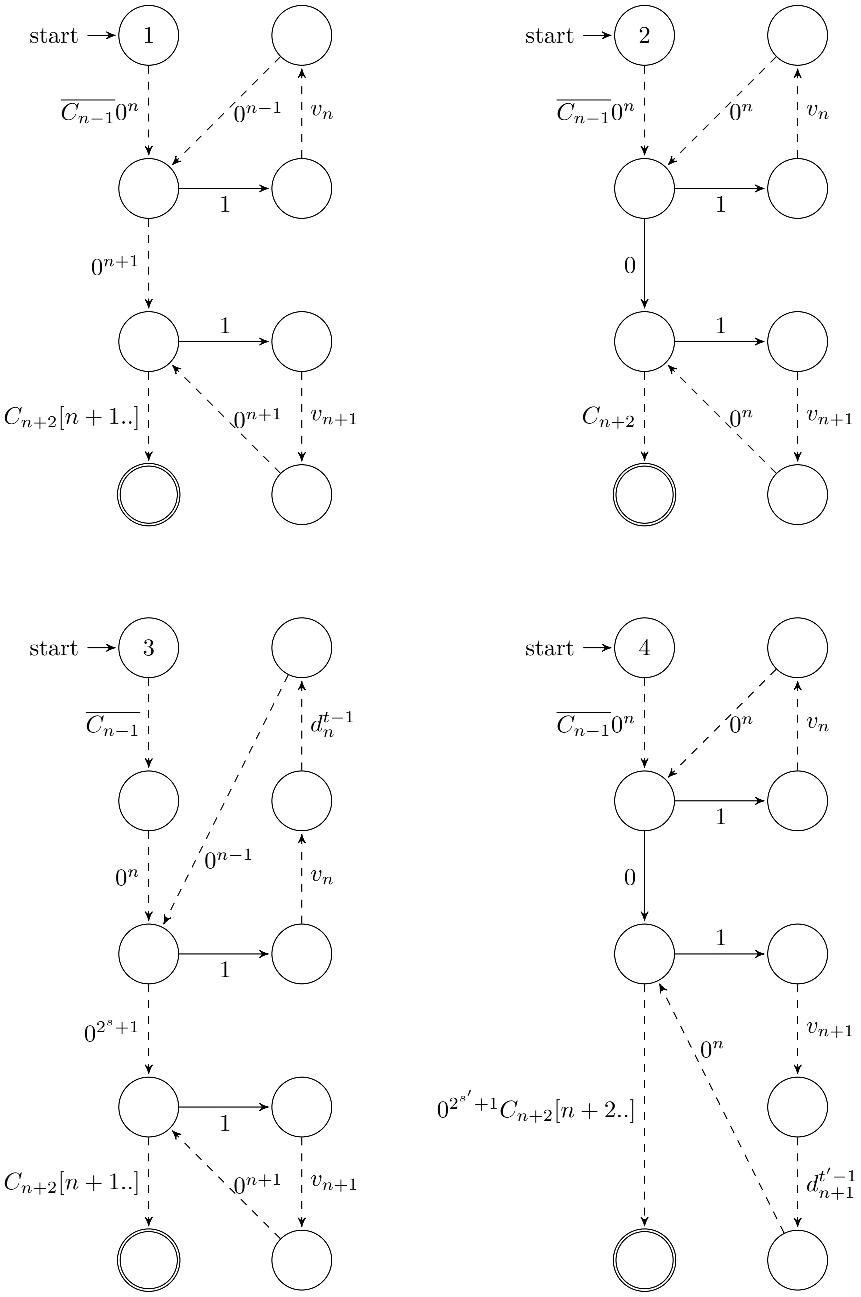


Figure 5 Automaton for Case 1 (top left), Case 2 (top right), Case 3 (bottom left) and Case 4 (bottom right). The dashed arrows represent the missing states belonging to their labels. The error state (the state traversed to if the bit seen is not the expected bit) and arrows to it are not included in each of the four diagrams. We point the reader to Figures 2 and 3 to help visualise the bits read on a single traversal of a loop.

References

1 Verónica Becher, Olivier Carton, and Pablo Ariel Heiber. Normality and automata. *J. Comput. Syst. Sci.*, 81(8):1592–1613, 2015. doi:10.1016/j.jcss.2015.04.007.

2 Verónica Becher and Pablo Ariel Heiber. Normal numbers and finite automata. *Theor. Comput. Sci.*, 477:109–116, 2013. doi:10.1016/j.tcs.2013.01.019.

3 Émile Borel. Les probabilités dénombrables et leurs applications arithmétiques. *Rendiconti del Circolo Matematico di Palermo*, 27(1):247–271, 1909. doi:10.1007/BF03019651.

4 Nicolaas Govert De Bruijn. A combinatorial problem. In *Proc. Koninklijke Nederlandse Academie van Wetenschappen*, volume 49, pages 758–764, 1946.

5 Cristian S. Calude, Kai Salomaa, and Tania Roblot. Finite state complexity. *Theor. Comput. Sci.*, 412(41):5668–5677, 2011. doi:10.1016/j.tcs.2011.06.021.

6 Cristian S. Calude and Ludwig Staiger. Liouville, computable, Borel normal and Martin-löf random numbers. *Theory Comput. Syst.*, 62(7):1573–1585, 2018. doi:10.1007/s00224-017-9767-8.

7 Cristian S. Calude, Ludwig Staiger, and Frank Stephan. Finite state incompressible infinite sequences. *Inf. Comput.*, 247:23–36, 2016. doi:10.1016/j.ic.2015.11.003.

8 Olivier Carton and Pablo Ariel Heiber. Normality and two-way automata. *Inf. Comput.*, 241:264–276, 2015. doi:10.1016/j.ic.2015.02.001.

9 D. G. Champernowne. The construction of decimals normal in the scale of ten. *J. Lond. Math. Soc.*, s1-8(4):254–260, 1933. doi:10.1112/jlms/s1-8.4.254.

10 Jack J. Dai, James I. Lathrop, Jack H. Lutz, and Elvira Mayordomo. Finite-state dimension. *Theor. Comput. Sci.*, 310(1-3):1–33, 2004. doi:10.1016/S0304-3975(03)00244-5.

11 David Doty and Philippe Moser. Finite-state dimension and lossy decompressors. *CoRR*, abs/cs/0609096, 2006. arXiv:cs/0609096.

12 David Doty and Philippe Moser. Feasible depth. In *CiE 2007: Computation and Logic in the Real World - Siena, Italy*, volume 4497 of *LNCS*, pages 228–237. Springer, 2007. doi:10.1007/978-3-540-73001-9_24.

13 Harold Fredricksen and Irving J. Kessler. An algorithm for generating necklaces of beads in two colors. *Discret. Math.*, 61(2-3):181–188, 1986. doi:10.1016/0012-365X(86)90089-0.

14 Harold Fredricksen and James Maiorana. Necklaces of beads in k colors and k-ary de Bruijn sequences. *Discret. Math.*, 23(3):207–210, 1978. doi:10.1016/0012-365X(78)90002-X.

15 Kayleigh Hyde. Nondeterministic finite state complexity. Master's thesis, University of Hawaii at Manoa, 2013. Accessed: April 20, 2021. URL: http://hdl.handle.net/10125/29507.

16 Kayleigh Hyde and Bjørn Kjos-Hanssen. Nondeterministic automatic complexity of overlap-free and almost square-free words. *Electron. J. Comb.*, 22(3):P3.22, 2015. doi:10.37236/4851.

17 Liam Jordon and Philippe Moser. On the difference between finite-state and pushdown depth. In *46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus*, volume 12011 of *LNCS*, pages 187–198. Springer, 2020. doi:10.1007/978-3-030-38919-2_16.

18 Liam Jordon and Philippe Moser. A normal sequence compressed by PPM* but not by Lempel-Ziv 78. In *47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021, Bolzano-Bozen, Italy*, volume 12607 of *LNCS*, pages 389–399. Springer, 2021. doi:10.1007/978-3-030-67731-2_28.

19 Bjørn Kjos-Hanssen. Automatic complexity of Fibonacci and Tribonacci words. *Discrete Applied Mathematics*, 289:446–454, 2021. doi:10.1016/j.dam.2020.10.014.

20 Bjørn Kjos-Hanssen. Automatic complexity of shift register sequences. *Discrete Mathematics*, 341(9):2409–2417, 2018. doi:10.1016/j.disc.2018.05.015.

21 Alexander Kozachinskiy and Alexander Shen. Automatic Kolmogorov complexity, normality, and finite-state dimension revisited. *J. Comput. Syst. Sci.*, 118:75–107, 2021. doi:10.1016/j.jcss.2020.12.003.

22 James I. Lathrop and Martin Strauss. A universal upper bound on the performance of the Lempel-Ziv algorithm on maliciously-constructed data. In *Compression and Complexity of SEQUENCES 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings*, pages 123–135. IEEE, 1997. doi:10.1109/SEQUEN.1997.666909.

23 M. H. Martin. A problem in arrangements. *Bull. Amer. Math. Soc.*, 40(12):859–864, 1934. doi:10.1090/S0002-9904-1934-05988-3.

24 Satyadev Nandakumar and Santhosh Kumar Vangapelli. Normality and finite-state dimension of Liouville numbers. *Theory Comput. Syst.*, 58(3):392–402, 2016. doi:10.1007/s00224-014-9554-8.

25 Larry A. Pierce and Paul C. Shields. Sequences incompressible by SLZ (LZW), yet fully compressible by ULZ. In *Numbers, Information and Complexity*, pages 385–390. Springer, 2000. doi:10.1007/978-1-4757-6048-4_32.

26 Joseph J. Rotman. *An Introduction to the Theory of Groups*. Graduate Texts in Mathematics. Springer, New York, 1995. ISBN: 978-1-4612-8686-8. doi:10.1007/978-1-4612-4176-8.

27 Camille Flye Sainte-Marie. Solution to question nr. 48. In *L'Intermédiaire des Mathématiciens*, volume 1, pages 107–110, 1894.

28 Claus-Peter Schnorr and H. Stimm. Endliche Automaten und Zufallsfolgen. *Acta Inf.*, 1:345–359, 1972. doi:10.1007/BF00289514.

29 Jeffrey O. Shallit and Ming-wei Wang. Automatic complexity of strings. *J. Autom. Lang. Comb.*, 6(4):537–554, 2001. doi:10.25596/jalc-2001-537.

30 Michael Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 1983, Boston, Massachusetts, USA*, pages 330–335. ACM, 1983. doi:10.1145/800061.808762.

A Appendix

A.1 The other three cases for the proof of Theorem 11

The following is the reasoning for why Equation (4) from the proof of Theorem 11 gives us the upper bound of $S(C)$. We perform similar calculations as in the proof for Cases $1, 2$ and 4 to see this.

Case 1. n is a power of 2.

Suppose we are in Case 1. Let p_{n+2} be such that $C[0..m] = \overline{C_{n+1}}p_{n+2}$. We have a automaton as in Case 1. It requires at most $|\overline{C_{n-1}}| + 1 + 2n + 2^n + 2^{n+1} + |p_{n+2}|$ states. Thus

$$\frac{A(C[0..m])}{m+1} \leq \frac{|\overline{C_{n-1}}| + 1 + 2n + 2^n + 2^{n+1} + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|}.$$

For $|p_{n+2}|$ long enough, it is more beneficial to loop in C_{n+2} instead of C_n and use an automaton in the style of Case 4 since worst case scenario $n+2$ has the form $2^{s'}t'$. Such an automaton has at most $|\overline{C_n}| + 1 + n + 2^{n+1} + 2^{n+2}(\frac{n+2}{2})$ states.

So for $|p_{n+2}| \leq 2^n(n-1) - n + 2^{n+2}(\frac{n+2}{2})$

$$\begin{aligned}\frac{A(C[0..m])}{m+1} &\leq \frac{|\overline{C_{n-1}}| + 1 + 2n + 2^n + 2^{n+1} + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|} \\ &\leq \frac{|\overline{C_{n-1}}| + 1 + 2n - n + 2^n + 2^n(n-1) + 2^{n+1} + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - n + 2^n(n-1) + 2^{n+2}(\frac{n+2}{2})} \\ &= \frac{|\overline{C_n}| + 1 + n + 2^{n+1} + 2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}| - n + 2^n(n-1) + 2^{n+2}(\frac{n+2}{2})}.\end{aligned}$$

For $j \geq 1$ such that $2^n(n-1)-n+2^{n+2}(\frac{n+2}{2}) \leq |p_{n+2}|+j < |C_{n+2}|$ we use an automaton from Case 4 and have that

$$\frac{A(C[0..m])}{m+1} \leq \frac{|\overline{C_n}|+1+n+2^{n+1}+2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}|-n+2^n(n-1)+2^{n+2}(\frac{n+2}{2})+j},$$

i.e. the number of states required remains constant. Furthermore

$$\lim_{n\to\infty} \frac{|\overline{C_n}|+1+n+2^{n+1}+2^{n+2}(\frac{n+2}{2})}{|\overline{C_{n+1}}|-n+2^n(n-1)+2^{n+2}(\frac{n+2}{2})} = \frac{4}{7} < \frac{2}{3}.$$

Case 2. $n+1$ is a power of 2

Suppose we are in Case 2. Let p_{n+2} be such that $C[0..m] = \overline{C_{n+1}}p_{n+2}$. We have a automaton as in Case 2. It requires at most $|\overline{C_{n-1}}|+n+2^n+2^{n+1}+|p_{n+2}|$ states. Thus

$$\frac{A(C[0..m])}{m+1} \leq \frac{|\overline{C_{n-1}}|+n+2^n+2^{n+1}+|p_{n+2}|}{|\overline{C_{n+1}}|+|p_{n+2}|}.$$

For $|p_{n+2}|$ long enough, it is more beneficial to loop in C_{n+2} instead of C_n and use an automaton in the style of Case 1 as $n+2$ is odd and $n+1$ is a power of 2. As the final state will be contained in the second loop, such an automaton requires $|\overline{C_n}|+2+2n+2^{n+1}+2^{n+2}$ states.

So for $|p_{n+2}| \leq 2+n+2^n(n-1)+2^{n+2}$

$$\begin{aligned}\frac{A(C[0..m])}{m+1} &\leq \frac{|\overline{C_{n-1}}|+n+2^n+2^{n+1}+|p_{n+2}|}{|\overline{C_{n+1}}|+|p_{n+2}|} \\ &\leq \frac{|\overline{C_{n-1}}|+2+n+n+2^n+2^n(n-1)+2^{n+1}+2^{n+2}}{|\overline{C_{n+1}}|+2+n+2^n(n-1)+2^{n+2}} \\ &= \frac{|\overline{C_n}|+2+2n+2^{n+1}+2^{n+2}}{|\overline{C_{n+1}}|+2+n+2^n(n-1)+2^{n+2}}.\end{aligned}$$

For $j \geq 1$ such that $2+n+2^n(n-1)+2^{n+2} \leq |p_{n+2}|+j < |C_{n+2}|$ we use an automaton from Case 1 and have that

$$\frac{A(C[0..m])}{m+1} \leq \frac{|\overline{C_n}|+2+2n+2^{n+1}+2^{n+2}}{|\overline{C_{n+1}}|+2+n+2^n(n-1)+2^{n+2}+j},$$

i.e. the number of states required remains constant. Furthermore

$$\lim_{n\to\infty} \frac{|\overline{C_n}|+2+2n+2^{n+1}+2^{n+2}}{|\overline{C_{n+1}}|+2+n+2^n(n-1)+2^{n+2}} = \frac{2}{5} < \frac{2}{3}.$$

Case 4. $n+1$ is even but not a power of 2

Suppose we are in Case 4, i.e. $n+1 = 2^s t$ for some $s \geq 1$ and $t \geq 3$ odd. Let p_{n+2} be such that $C[0..m] = \overline{C_{n+1}}p_{n+2}$. We have a automaton as in Case 4. It requires at most $|\overline{C_{n-1}}|+n+2^n+2^{n+1}t+|p_{n+2}|$ states.

Thus

$$\frac{A(C[0..m])}{m+1} \leq \frac{|\overline{C_{n-1}}|+n+2^n+2^{n+1}t+|p_{n+2}|}{|\overline{C_{n+1}}|+|p_{n+2}|}.$$

For $|p_{n+2}|$ long enough, it is more beneficial to loop in C_{n+2} instead of C_n and use an automaton in the style of Case 3 as $n+2$ is odd and $n+1$ is even but not power of 2. As the final state will be contained in the second loop, such an automaton requires $|\overline{C_n}| + n + 2^{n+1}t + 2^s + 2^{n+2}$ states. Hence, as $2^s \leq n/3$ and we have that the number of states required is bounded above by $|\overline{C_n}| + 4n/3 + 2^{n+1}t + 2^{n+2}$.

So for $|p_{n+2}| \leq n/3 + 2^n(n-1) + 2^{n+2}$

$$\begin{aligned}\frac{A(C[0..m])}{m+1} &\leq \frac{|\overline{C_{n-1}}| + n + 2^n + 2^{n+1}t + |p_{n+2}|}{|\overline{C_{n+1}}| + |p_{n+2}|} \\ &\leq \frac{|\overline{C_{n-1}}| + n + \frac{n}{3} + 2^n + 2^n(n-1) + 2^{n+1}t + 2^{n+2}}{|\overline{C_{n+1}}| + \frac{n}{3} + 2^n(n-1) + 2^{n+2}} \\ &= \frac{|\overline{C_n}| + \frac{4n}{3} + 2^{n+1}t + 2^{n+2}}{|\overline{C_{n+1}}| + \frac{n}{3} + 2^n(n-1) + 2^{n+2}}.\end{aligned}$$

For $j \geq 1$ such that $n/3 + 2^n(n-1) + 2^{n+2} \leq |p_{n+2}| + j < |C_{n+2}|$ we use an automaton from Case 3 and have that

$$\frac{A(C[0..m])}{m+1} \leq \frac{|\overline{C_n}| + \frac{4n}{3} + 2^{n+1}t + 2^{n+2}}{|\overline{C_{n+1}}| + \frac{n}{3} + 2^n(n-1) + 2^{n+2} + j},$$

i.e. the number of states required remains constant. Furthermore

$$\begin{aligned}\lim_{n\to\infty} \frac{|\overline{C_n}| + \frac{4n}{3} + 2^{n+1}t + 2^{n+2}}{|\overline{C_{n+1}}| + \frac{n}{3} + 2^n(n-1) + 2^{n+2}} &\leq \lim_{n\to\infty} \frac{|\overline{C_n}| + \frac{4n}{3} + 2^{n+1}\frac{(n+1)}{2} + 2^{n+2}}{|\overline{C_{n+1}}| + \frac{n}{3} + 2^n(n-1) + 2^{n+2}} \\ &= \frac{3}{5} < \frac{2}{3}.\end{aligned}$$

A.2 Analogous Result to Proposition 12

The following is analogous for Proposition 12 and demonstrates that if reading a zone C_j where $j = 2^k$ for $k \geq 1$, any loop traversed used to read a substring x of C_j, if $|x| \geq j$ then $|x|$ must be a multiple of $2^j - 1$.

▶ **Proposition 14.** *Let $C \in$ PSC. Let $j = 2^k$ for some $k \geq 1$ and let C' be a prefix of C containing the substring C_j. Let $p_0 \to p_1 \to \cdots \to p_{2^j j}$ be the sequence of states an automaton which uniquely accepts C' traverses while reading C_j. If there is some subsequence of states $p_i \to \cdots p_{i+l} \to \cdots p_{i+2l}$ where $l \geq j$ and for all $0 \leq m \leq l$, $p_{i+m} = p_{i+l+m}$, then l must be a multiple of $2^j - 1$.*

Proof. Let C, C', j and k be as above. Recall in such a case that

$$C_j = d_j \cdot d_{j,1} \cdot d_{j,2} \cdots d_{j,2^{j-1}} = 0 \cdot (d_j[1..2^j - 1])^j \cdot 0^{j-1}$$

for some de Bruijn string d_j. First suppose a loop of length l is traversed where $j \leq l \leq 2^j - 2$. Let x be the substring of C_j read during the loop. Hence $x = yz$ where $|y| = j$ and $|z| \leq 2^j - j - 2$. Suppose the loop is traversed twice in a row indicating that $x^2 = yzyz$ is a substring of C_j. Consider yzy which has length at most $2^j + j - 2$. By the construction of C_j, every substring of length $2^j + j - 2$ contains $2^j - 1$ of the strings of length j as a substring exactly once where either 0^j or 10^{j-1} is the remaining string that does not appear. If $y = 0^j$ then 10^{j-1} is missing, otherwise 0^j is missing due to this shift between instances of the de Bruijn strings. However, y is then a substring of length j contained twice withing a substring of length $2^j + j - 2$ giving us a contradiction.

Next suppose a loop of length l is traversed where $d(2^j - 1) < l < (d+1)(2^j - 1)$ for some $d \leq \lfloor j/2 \rfloor$. Let x be the string read while traversing the loop. Hence $x = yz$ where $|y| = d(2^j - 1)$ and $1 \leq |z| < 2^j - 1$. Note that $y = (d_j[i..2^j - 1] \cdot d_j[1..i-1])^d$ for some $i \geq 1$ ($i \neq 0$ as $\mathrm{occ}(0^j, C_j) = 1$). Suppose the loop is traversed twice in a row indicating that $x^2 = yzyz$ is a substring of C_j. By the construction of C_j, this means that z is a prefix of $d_j[i..2^j - 1] \cdot d_j[1..i-1]$. This forces $z = \lambda$ or $z = d_j[i..2^j - 1] \cdot d_j[1..i-1]$ which contradicts the length requirement of z. ◀

Approximate Bisimulation Minimisation

Stefan Kiefer
Department of Computer Science, University of Oxford, UK

Qiyi Tang
Department of Computer Science, University of Oxford, UK

Abstract

We propose polynomial-time algorithms to minimise labelled Markov chains whose transition probabilities are not known exactly, have been perturbed, or can only be obtained by sampling. Our algorithms are based on a new notion of an approximate bisimulation quotient, obtained by lumping together states that are exactly bisimilar in a slightly perturbed system. We present experiments that show that our algorithms are able to recover the structure of the bisimulation quotient of the unperturbed system.

2012 ACM Subject Classification Theory of computation → Program verification; Theory of computation → Models of computation; Mathematics of computing → Probability and statistics

Keywords and phrases Markov chains, Behavioural metrics, Bisimulation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.48

Related Version *Full Version*: https://arxiv.org/abs/2110.00326 [15]

Supplementary Material
Software (Source Code): https://github.com/qiyitang71/approximate-quotienting
archived at swh:1:dir:42b8e694ce904b906e52bac502b425e424412461
Dataset (Experimental Results): https://bit.ly/3vcpblY

Funding *Stefan Kiefer*: Supported by a Royal Society University Fellowship.

Acknowledgements We thank the anonymous reviewers of this paper for their constructive feedback.

1 Introduction

For the algorithmic analysis and verification of system models, computing the bisimulation quotient is a natural preprocessing step: it can make the system much smaller while preserving most properties of interest. This applies equally to probabilistic systems: probabilistic model checkers, e.g., Storm [12], speed up the verification process by "lumping" together states that are equivalent with respect to probabilistic bisimulation. While this is a safe approach, it may not be effective when the probabilities in the system are not known precisely. For example, in the labelled Markov chain shown in Figure 1 the states s_1, t_1 are intuitively "similar", but they are not probabilistically bisimilar even though they carry the same label (here indicated with the colour white) and they both lead, with similar probabilities, to states s_2, t_2, which are again intuitively "similar" but not probabilistically bisimilar.

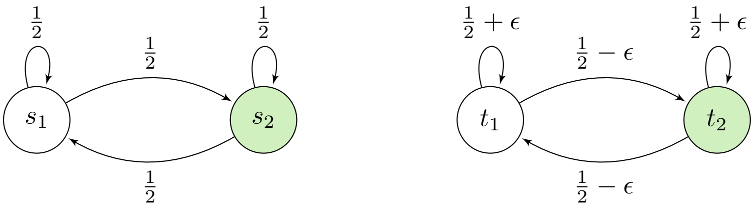


Figure 1 Two intuitively similar labelled Markov chains.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 48; pp. 48:1–48:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

For analysis and verification of a probabilistic system, tackling state space explosion is key. The more symmetry a system has (e.g., due to variables that do not influence the observable behaviour of the model), the greater are the benefits of computing a *quotient* system with respect to probabilistic bisimulation. However, when the transition probabilities in the Markov chain are perturbed or not known exactly, bisimulation minimisation may fail to capture the symmetries in the system and thus fail to achieve the objective of making the probabilistic system amenable to algorithmic analysis. The motivation of this paper is to counter this problem.

A principled approach towards this goal may consider notions of *distance* between probabilistic systems or between states in a probabilistic system. A *probabilistic bisimilarity pseudometric*, based on the Kantorovich metric, was introduced in [8, 9], which assigns to each pair of states s, t a number in the interval $[0, 1]$ measuring a distance between s, t: distance 0 means probabilistic bisimilar, and distance 1 means, in a sense, maximally non-bisimilar. This pseudometric can be computed in polynomial time [5], and, quoting [5], has "been studied in the context of systems biology, games, planning and security, among others". The corresponding distances can be intuitively large though: the pseudometric yields a distance less than 1 only if the two states can reach, with the same label sequence, two states that are *exactly* bisimilar; see [18, Section 4]. As a consequence, for any $\epsilon > 0$, the states s_1, t_1 in Figure 1 have distance 1 in the probabilistic bisimilarity pseudometric of [9] (in the undiscounted version). From a slightly different point of view, a small perturbation of the transition probabilities in the model can change the distance between two states from 0 to 1.

Another pseudometric, *ϵ-bisimulation* $\sim_\epsilon$, was defined in [10], which avoids this issue. It has natural characterisations in terms of games and can be computed in polynomial time using network-flow based algorithms [10]. The runtime of the algorithm from [10] is $O(|S|^7)$, where S is the set of states, thus not practical for large systems. A more fundamental problem lies in the fact that ϵ-bisimulation is not an equivalence: $s \sim_\epsilon t \sim_\epsilon u$ implies $s \sim_{2\epsilon} u$ (by the triangle inequality) but not necessarily $s \sim_\epsilon u$. Therefore, efficient minimisation algorithms via quotienting (such as partition refinement for exact probabilistic bisimilarity) are not available for ϵ-bisimulation.

In this paper we develop algorithms that, given a labelled Markov chain $\mathcal{M}$ with possibly imprecise transition probabilities and a *slightly perturbed* version, say $\mathcal{M}'$, of $\mathcal{M}$, compute a compressed version, $\mathcal{Q}$, of $\mathcal{M}'$. By slightly perturbed we mean that for each state the successor distributions in $\mathcal{M}'$ and $\mathcal{M}$ have small (say, less than ϵ) L_1-distance. We hope that $\mathcal{Q}$ is not much bigger than the exact quotient of $\mathcal{M}$, and we design polynomial-time algorithms that fulfill this hope in practice, but we do not insist on computing the smallest possible $\mathcal{Q}$. Indeed, we show that, given an LMC, $\epsilon > 0$ and a positive integer k, it is NP-complete to decide whether there exists a perturbation of at most ϵ such that the (exact) bisimulation quotient of the perturbed system is of size k. See, e.g., [1] for an exact but non-polynomial approach, where the target number n of states is fixed, and a Markov chain with at most n states is sought that has minimal distance (with respect to the probabilistic bisimilarity pseudometric of [9]) to the given model.

It is not hard to prove (Proposition 2) that if an LMC can be made exactly bisimilar to another LMC by a perturbation of at most ϵ, then these two LMCs are also $\frac{\epsilon}{2}$-bisimilar in the sense of [10]. If, in turn, two states are ϵ-bisimilar, one can show (see [3, Theorem 4]) that any linear-time property that depends only on the first k labels has similar probabilities in the two states, where similar means the difference in probabilities is at most $1 - (1 - \epsilon)^k$. Combining these two results, we obtain a continuity property that says if two LMCs can be made bisimilar by a small perturbation, then any k-bounded linear-time property has similar probabilities in the two LMCs. In other words, our approximate minimisation approximately preserves the probability of bounded linear-time properties.

We apply our approximate minimisation algorithms in a setting of active learning. Here we assume we do not have access to the transition probabilities of the model; rather, for each state we only *sample* the successor distribution. Sampling gives us an approximation of the real Markov chain, and our approximate minimisation algorithms apply naturally. This allows us to lump states that are exactly bisimilar in the real model, but only approximately bisimilar in the sampled model. We give examples where in this way we *recover* the structure (not the precise transition probabilities) of the quotient of the exact model, knowing only the sampled model.

The rest of the paper is organised as follows. In Section 2 we introduce ϵ-quotient, a new notion of approximate bisimulation quotient. In Section 3 given an LMC, ϵ and $k > 0$, we show it is NP-complete to decide whether there exists an ϵ-quotient of size k. In Section 4 we present our approximate minimisation algorithms. We put them in a context of active learning in Section 5. In Section 6 we evaluate these algorithms on slightly perturbed versions of a number of LMCs taken from the probabilistic model checker PRISM [16]. We conclude in Section 7.

2 Preliminaries

We write $\mathbb{N}$ for the set of nonnegative integers and $\mathbb{Z}^+$ for the set of positive integers. We write $\mathbb{R}$ for the set of real numbers. Let S be a finite set. We denote by $\mathsf{Distr}(S)$ the set of probability distributions on S. By default we view vectors, i.e., elements of $\mathbb{R}^S$, as row vectors. For a vector $\mu \in \mathbb{R}^S$ we write $\|\mu\|_1 := \sum_{s \in S} |\mu(s)|$ for its L_1-norm. A vector $\mu \in [0,1]^S$ is a distribution (resp. subdistribution) over S if $\|\mu\|_1 = 1$ (resp. $0 < \|\mu\|_1 \leq 1$). For a (sub)distribution μ we write $\mathsf{support}(\mu) = \{s \in S : \mu(s) > 0\}$ for its support.

A partition of the states S is a set X consisting of pairwise disjoint subsets E of S with $\bigcup_{E \in X} = S$. For an equivalence relation $\mathcal{R} \subseteq S \times S$, $S/_{\mathcal{R}}$ denotes its quotient set and $[s]_{\mathcal{R}}$ denotes the $\mathcal{R}$-equivalence class of $s \in S$.

A *labelled Markov chain* (LMC) is a quadruple $\langle S, L, \tau, \ell \rangle$ consisting of a nonempty finite set S of states, a nonempty finite set L of labels, a transition function $\tau : S \to \mathsf{Distr}(S)$, and a labelling function $\ell : S \to L$.

We denote by $\tau(s)(t)$ the transition probability from s to t. Similarly, we denote by $\tau(s)(E) = \sum_{t \in E} \tau(s)(t)$ the transition probability from s to $E \subseteq S$. For the remainder of the paper, we fix an LMC $\mathcal{M} = \langle S, L, \tau, \ell \rangle$. Let $|\mathcal{M}|$ denote the number of states, $|S|$.

The direct sum $\mathcal{M}_1 \oplus \mathcal{M}_2$ of two LMCs $\mathcal{M}_1 = \langle S_1, L_1, \tau_1, \ell_1 \rangle$ and $\mathcal{M}_2 = \langle S_2, L_2, \tau_2, \ell_2 \rangle$ is the LMC formed by combining the state spaces of $\mathcal{M}_1$ and $\mathcal{M}_2$.

An equivalence relation $\mathcal{R} \subseteq S \times S$ is a *probabilistic bisimulation* if for all $(s,t) \in \mathcal{R}$, $\ell(s) = \ell(t)$ and $\tau(s)(E) = \tau(t)(E)$ for each $\mathcal{R}$-equivalence class E. *Probabilistic bisimilarity*, denoted by $\sim_{\mathcal{M}}$ (or $\sim$ when $\mathcal{M}$ is clear), is the largest probabilistic bisimulation.

Any probabilistic bisimulation $\mathcal{R}$ on $\mathcal{M}$ induces a quotient LMC denoted by $\mathcal{M}/_{\mathcal{R}} = \langle S/_{\mathcal{R}}, L, \tau/_{\mathcal{R}}, \ell/_{\mathcal{R}} \rangle$ where the transition function $\tau/_{\mathcal{R}}([s]_{\mathcal{R}})([t]_{\mathcal{R}}) = \tau(s)([t]_{\mathcal{R}})$ and the labelling function $\ell/_{\mathcal{R}}([s]_{\mathcal{R}}) = \ell(s)$.

We define the notion of an *approximate quotient.* Let $\epsilon \geq 0$. An LMC $\mathcal{Q}$ is an ϵ-*quotient* of $\mathcal{M}$ if and only if there is transition function $\tau' : S \to \mathsf{Distr}(S)$ such that for all $s \in S$ we have $\|\tau'(s) - \tau(s)\|_1 \leq \epsilon$ and $\mathcal{Q}$ is the (exact) bisimulation quotient of the LMC $\mathcal{M}' = \langle S, L, \tau', \ell \rangle$, that is, $\mathcal{Q} = \mathcal{M}'/_{\sim_{\mathcal{M}'}}$. Since the choice of τ' is not unique, there might be multiple ϵ-quotients of $\mathcal{M}$. We are interested in the problem of obtaining an ϵ-quotient of $\mathcal{M}$ with small state space. We retrieve the notion of (exact) quotient when $\epsilon = 0$. For s from $\mathcal{M}$, denote the state in $\mathcal{Q}$ which corresponds to s by $[s]^{\epsilon}_{\mathcal{Q}}$ (or $[s]^{\epsilon}$ when $\mathcal{Q}$ is clear).

The set $\Omega(\mu, \nu)$ of *couplings* of $\mu, \nu \in \mathrm{Distr}(S)$ is defined as $\Omega(\mu, \nu) = \big\{ \omega \in \mathrm{Distr}(S \times S) : \sum_{t \in S} \omega(s,t) = \mu(s) \wedge \sum_{s \in S} \omega(s,t) = \nu(t) \big\}$. Note that a coupling $\omega \in \Omega$ is a joint probability distribution with marginals μ and ν (see, e.g., [4, page 260-262]).

The *ϵ-lifting* of a relation $\mathcal{R} \subseteq S \times S$ proposed by Tracol et al. [19] is the relation $\mathcal{R}{\uparrow_{\epsilon}} \subseteq \mathrm{Distr}(S) \times \mathrm{Distr}(S)$ defined by $(\mu, \nu) \in \mathcal{R}{\uparrow_{\epsilon}}$ if there exists $\omega \in \Omega(\mu, \nu)$ such that $\sum_{(u,v) \in \mathcal{R}} \omega(u, v) \geq 1 - \epsilon$.

The *ϵ-bisimulation* ($\sim_{\epsilon}$) by Desharnais et al. [10] is a relation $\mathcal{R} \subseteq S \times S$ in which for all $(s, t) \in \mathcal{R}$ we have $\ell(s) = \ell(t)$ and $(\tau(s), \tau(t)) \in \mathcal{R}{\uparrow_{\epsilon}}$. The ϵ-bisimulation is reflexive and symmetric, but in general not transitive; hence, it is not an equivalence relation.

3 Properties of Approximate Quotients

$$\begin{tikzcd}
\mathcal{M} \arrow[r, "\text{quotient}"] \arrow[d, "\text{perturbation}"'] & \mathcal{M}/_{\sim_{\mathcal{M}}} \arrow[d, dash, dashed, "\mathcal{Q} \text{ is not much bigger than } \mathcal{M}/_{\sim_{\mathcal{M}}}"] \\
\mathcal{M}' \arrow[r, "\text{approximate quotient}"] & \mathcal{Q}
\end{tikzcd}$$

Figure 2 Problem setup.

Recall from the introduction that we are given an LMC $\mathcal{M}'$, which is a slightly perturbed version of an (unknown) LMC $\mathcal{M}$. By slightly perturbed we mean that for each state the successor distributions in $\mathcal{M}'$ and $\mathcal{M}$ have small (say, less than ϵ) L_1-distance. For example, with sampling we can obtain with high probability a perturbed system that has small distance with $\mathcal{M}$. Assume there are many symmetries, that is, lots of probabilistic bisimilar states in $\mathcal{M}$. The state space of $\mathcal{M}$ can then be compressed a lot by (exact) quotienting. Since the transition probabilities are perturbed in $\mathcal{M}'$, the states that are probabilistic bisimilar in $\mathcal{M}$ might become inequivalent in $\mathcal{M}'$; as a result, the (exact) bisimulation quotient of $\mathcal{M}'$ is much larger than that of $\mathcal{M}$. Given a small compression parameter $\epsilon_2 > 0$, we aim to compute an approximate quotient $\mathcal{Q}$, an ϵ'-quotient of $\mathcal{M}'$ that satisfies two conditions: (1) ϵ' should be small, so that little precision is sacrificed; and (2) the state space of the quotient should be small, to speed up verification algorithms. Our contribution consists of approximate quotienting algorithms with (a) theoretical guarantees on goal (1) in Theorem 7 and Corollary 8, applying to both algorithms: ϵ' is bounded (and can be controlled) by a compression parameter ϵ_2 and the number of iterations i; (b) empirical results on goal (2): the experiments show that our algorithms produce small quotients.

We first show that it is hard to find an ϵ_2-quotient of $\mathcal{M}'$ with minimum number of states: $\mathcal{Q}^* = \arg\min\{|\mathcal{Q}| : \mathcal{Q}$ is an ϵ_2-quotient of $\mathcal{M}'\}$. If there are several ϵ_2-quotients of $\mathcal{M}'$ of minimum size, $\mathcal{Q}^*$ can be taken to be any one of them. Unfortunately, this problem is unlikely to have an efficient (polynomial-time) solution, as we will see from the next theorem that the decision version of this problem is NP-complete.

Given an LMC $\mathcal{M}'$, a compression parameter $\epsilon_2 > 0$ and a constant $k \in \mathbb{Z}^+$, it is NP-complete to decide whether there exists an ϵ_2-quotient of $\mathcal{M}'$ of size k. The hardness result is by reduction from the Subset Sum problem. Given a set $P = \{p_1, \ldots, p_n\}$ and $N \in \mathbb{N}$, Subset Sum asks whether there exists a set $Q \subseteq P$ such that $\sum_{p_i \in Q} p_i = N$. Given an instance of Subset Sum $\langle P, N \rangle$ where $P = \{p_1, \ldots, p_n\}$ and $N \in \mathbb{N}$, we construct an LMC;

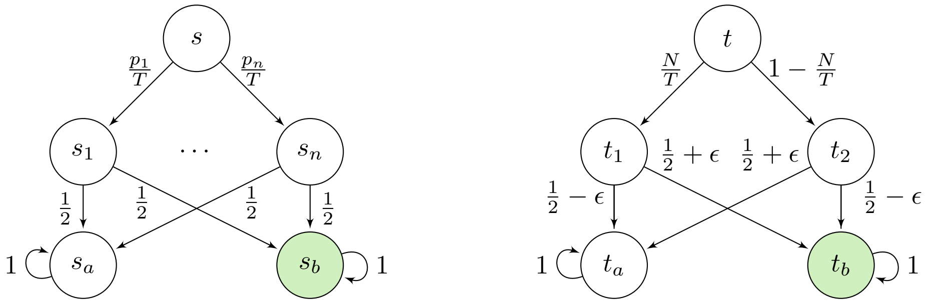


Figure 3 The LMC in the reduction for NP-hardness. All states have the same label a except s_b and t_b which have label b.

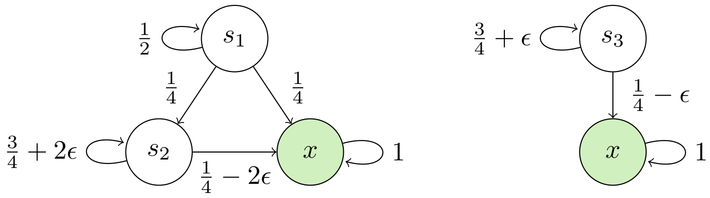


Figure 4 An LMC in which $s_1 \sim_\epsilon s_3 \sim_\epsilon s_2$.

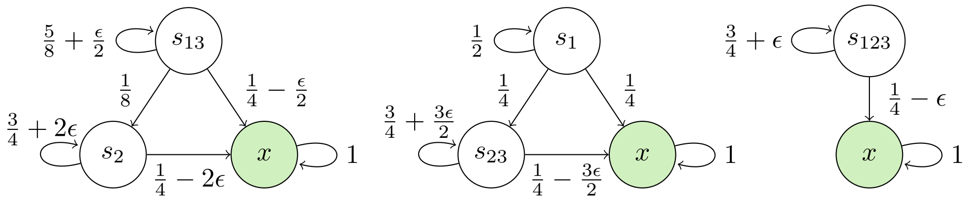
see Figure 3. Let $T = \sum_{p_i \in P} p_i$, $\epsilon = \frac{1}{2T}$ and $k = 5$. In the LMC, state s transitions to state s_i with probability p_i/T for all $1 \le i \le n$. Each state s_i transitions to s_a and s_b with equal probabilities. State t transitions to t_1 and t_2 with probability N/T and $1 - N/T$, respectively. State t_1 (resp. t_2) transitions to t_a (resp. t_b) and t_b (resp. t_a) with probability $\frac{1}{2} - \epsilon$ and $\frac{1}{2} + \epsilon$, respectively. All the remaining states transition to the successor state with probability one. States s_b and t_b have label b and all other states have label a. We can show that $\langle P, N\rangle \in$ Subset Sum $\iff$ there exists an $\frac{1}{2T}$-quotient of $\mathcal{M}'$ of size 5.

▶ **Theorem 1.** *Given an LMC $\mathcal{M}'$, $\epsilon_2 \in (0, 1]$ and $k \in \mathbb{Z}^+$. The problem whether there exists an ϵ_2-quotient of $\mathcal{M}'$ of size k is* NP*-complete. It is* NP*-hard even for (fixed) $k = 5$.*

Due to the NP-hardness result, we hope to develop practical algorithms to compute approximate quotients of $\mathcal{M}'$ that are small but not necessarily of minimum size. To do that, an intuitive idea is to merge "similar" states. As we have discussed in the introduction, merging states with small probabilistic bisimilarity distances might be insufficient. Consider the LMC shown in Figure 1. Assume $\epsilon > 0$. The states s_1 and t_1 (s_2 and t_2) have probabilistic bisimilarity distance one. Thus, to merge s_1, t_1 or s_2, t_2, one needs to merge states with probabilistic bisimilarity distance one. Alternatively, we explore the relation of approximate quotient and ϵ-bisimulation. It is not hard to prove the following proposition:

▶ **Proposition 2.** *Let $\mathcal{Q}$ be an ϵ_2-quotient of $\mathcal{M}'$. Then in the LMC $\mathcal{M}' \oplus \mathcal{Q}$, we have $s \sim_{\frac{\epsilon_2}{2}} [s]_{\mathcal{Q}}^{\epsilon_2}$ for all s from $\mathcal{M}'$.*

Proposition 2 suggests that ϵ_2-quotients and ϵ_2-bisimulation are related. The runtime of the algorithm to compute the ϵ_2-bisimulation in [10] is $O(|S|^7)$ which makes it not practical for large systems. Furthermore, the algorithms based on merging states that are ϵ_2-bisimilar may produce an ϵ'-quotient where ϵ' is large, violating the first condition of a satisfying approximate quotient. Assume the positive number ϵ is much smaller than $\frac{1}{8}$. Let us choose the compression parameter ϵ_2 to be the same as ϵ. We compute the ϵ-bisimulation of the LMC shown in Figure 4 and get $s_1 \sim_\epsilon s_3 \sim_\epsilon s_2$. Since ϵ-bisimulation is not an equivalence



(a) ϵ'-quotient (b) ϵ-quotient (c) 2ϵ-quotient

Figure 5 (a) An ϵ'-quotient obtained by merging s_1 and s_3 where ϵ' is at least $\frac{1}{4}+\epsilon$; (b) An ϵ-quotient obtained by merging s_2 and s_3; (b) A 2ϵ-quotient obtained by merging s_1, s_2 and s_3.

relation, $s_1 \sim_\epsilon s_2$ does not necessarily follow. Indeed, in this LMC, we have $s_1 \sim_{2\epsilon} s_2$ but not $s_1 \sim_\epsilon s_2$. If s_2 and s_3, related by $\sim_\epsilon$, are chosen to be merged, the resulting LMC in Figure 5(b) is an ϵ-quotient. However, if s_1 and s_3 are (unfortunately) chosen to be merged, the resulting LMC, shown in Figure 5(a), is an ϵ'-quotient where ϵ' cannot be smaller than $\frac{1}{4}+\epsilon$. This ϵ', much bigger than ϵ under the assumption that ϵ is much smaller than $\frac{1}{8}$, makes the resulting LMC undesirable. This example shows that arbitrarily merging states that are ϵ-bisimilar may not work. The LMC in Figure 5(c) is obtained by merging s_1, s_2 and s_3, the states that are related by the transitive closure of $\sim_\epsilon$. We show in [15] that for any $n \in \mathbb{Z}^+$ there exists an LMC $\mathcal{M}(n)$ such that merging all states in $\mathcal{M}(n)$ that are related by the transitive closure of $\sim_\epsilon$ results in an ϵ'-quotient where ϵ' is at least $n\epsilon$.

Lemma 3, the additivity lemma, asserts an additivity property of approximate quotients. In Section 4, this lemma will be applied as the two minimisation algorithms successively compute a sequence of approximate quotients.

▶ **Lemma 3.** *Consider three LMCs $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$. Let $\epsilon_1 \geq 0$ and $\mathcal{M}_2$ be an ϵ_1-quotient of $\mathcal{M}_1$. Let $\epsilon_2 \geq 0$ and $\mathcal{M}_3$ be an ϵ_2-quotient of $\mathcal{M}_2$. Then $\mathcal{M}_3$ is an $(\epsilon_1+\epsilon_2)$-quotient of $\mathcal{M}_1$.*

4 Approximate Minimisation Algorithms

$$\begin{tikzcd}
\mathcal{M}' \arrow[r, "\text{exact}", "\text{quotient}"'] & \mathcal{Q}_0 \arrow[r, "\text{approximate}", "\text{quotient}"'] & \mathcal{Q}_1 \arrow[r, "\text{approximate}", "\text{quotient}"'] & \cdots \arrow[r, "\text{approximate}", "\text{quotient}"'] & \mathcal{Q}_i
\end{tikzcd}$$

Figure 6 Overview of the minimisation algorithms. Lemma 3 applies to $\mathcal{M}'$, $\mathcal{Q}_0$, $\mathcal{Q}_1, \cdots, \mathcal{Q}_i$.

In this section, we present two practical minimisation algorithms that compute approximate quotients of $\mathcal{M}'$. Given an LMC $\mathcal{M}' = \langle S, L, \tau_\epsilon, \ell \rangle$ with perturbed transition probabilities and a small compression parameter ϵ_2. Both algorithms start by computing $\mathcal{Q}_0$, the exact quotient of $\mathcal{M}'$. They proceed in iterations and compute a sequence of approximate quotients where the approximate quotient ($\mathcal{Q}_i$) computed at the end of the ith iteration is an ϵ_2-quotient of the quotient ($\mathcal{Q}_{i-1}$) given at the beginning of that iteration. Using the additivity lemma, we can show that the (approximate) quotient $\mathcal{Q}_i$ after the ith iteration is an $i\epsilon_2$-quotient of $\mathcal{M}'$. See Figure 6 for an overview of this approach. Each iteration computes a partition of the state space, lumps the states that are together in the partition and concludes with taking the exact quotient.

4.1 Local Bisimilarity Distance

We define the notion of *local bisimilarity distance*, denoted by $d^{\mathcal{M}}_{local}$ (or d_{local} when $\mathcal{M}$ is clear). Intuitively, two states s and t are at small local bisimilarity distance if they are probabilistic bisimilar in an LMC which is slightly perturbed only at the successor distributions of s and t. We provide a polynomial-time algorithm to compute the local bisimilarity distance. Given an LMC $\mathcal{M}' = \langle S, L, \tau_\epsilon, \ell\rangle$ (with perturbed transition probabilities) and a small compression parameter ϵ_2, we propose an iterative minimisation algorithm to compute approximate quotients of $\mathcal{M}'$ by merging state pairs with small local bisimilarity distances. In each iteration of the algorithm, we select the state pair with the same label and the minimum local bisimilarity distance if such distance is at most ϵ_2. We compute a partition in which this state pair are together and lump together the states that are together in the partition. The algorithm terminates when no pairs can be lumped, that is, all state pairs have local bisimilarity distances greater than ϵ_2.

Computing Local Bisimilarity Distances

Given two different states $s, t \in S$ with the same label. We want to compute a new transition function τ'_ϵ by only changing the successor distributions of s and t ($\tau_\epsilon(s)$ and $\tau_\epsilon(t)$, respectively) such that $\{s, t\}$ belongs to an $\mathcal{R}$-induced partition where $\mathcal{R}$ is a probabilistic bisimulation of the LMC $\mathcal{M}'' = \langle S, L, \tau'_\epsilon, \ell\rangle$. Let T be the set of the all transition functions that satisfy this condition, more precisely, we define $\mathrm{T} = \{\tau'_\epsilon : \tau'_\epsilon(x) = \tau_\epsilon(x)\ \forall x \notin \{s, t\} \wedge \{s, t\} \in S/_{\mathcal{R}}$ where $\mathcal{R}$ is a probabilistic bisimulation of the LMC $\mathcal{M}'' = \langle S, L, \tau'_\epsilon, \ell\rangle\}$. The local bisimilarity distance is defined as $d^{\mathcal{M}'}_{local}(s, t) = \inf_{\tau' \in \mathrm{T}} \max\{\|\tau'(s) - \tau_\epsilon(s)\|_1, \|\tau'(t) - \tau_\epsilon(t)\|_1\}$. It is not immediately clear how to compute it.

By the definition of T, the probabilistic bisimulation $\mathcal{R}$ is the same for any LMC $\langle S, L, \tau'_\epsilon, \ell\rangle$ with $\tau'_\epsilon \in \mathrm{T}$. Let us define the partition $X = S/_{\mathcal{R}}$ where $\mathcal{R}$ is the common probabilistic bisimulation. The local bisimilarity distance can be computed by using X:

▶ **Proposition 4.** *We have* $d^{\mathcal{M}'}_{local}(s, t) = \frac{1}{2}\|(\tau_\epsilon(s)(E))_{E \in X} - (\tau_\epsilon(t)(E))_{E \in X}\|_1$.

It turns out that X can simply be computed by Algorithm 1. As this algorithm is basically taking the (exact) quotient of the LMC constructed on line 1, it runs in polynomial time. It follows from Proposition 4 that the local bisimilarity distance can be computed in polynomial time.

Algorithm 1 Compute Partition for Local Bisimilarity Distances.

```
  Input: An LMC $\mathcal{M}' = \langle S, L, \tau_\epsilon, \ell\rangle$, a state pair $(s, t) \in S \times S$
  Output: A partition $X$ over $S$ containing $\{s, t\}$
1 Construct a new LMC $\mathcal{M}''$ from $\mathcal{M}'$ by introducing a new label, labelling both $s$ and
   $t$ with the new label and making both $s$ and $t$ absorbing[1]
2 $X := S/_{\sim_{\mathcal{M}''}}$
```

▶ **Example 5.** Assume $\epsilon < \frac{1}{2}$. Consider the LMC shown in Figure 1. Let τ_ϵ denote its transition function. To compute the local bisimilarity distance of s_1 and t_1, we first compute the partition containing $\{s_1, t_1\}$: $X = \big\{\{s_1, t_1\}, \{s_2\}, \{t_2\}\big\}$. We have

[1] An absorbing state is a state that, once entered, cannot be left; that is, a state with self-loop.

$(\tau_\epsilon(s_1)(E))_{E\in X} = (\frac{1}{2}, \frac{1}{2}, 0)$ and $(\tau_\epsilon(t_1)(E))_{E\in X} = (\frac{1}{2}+\epsilon, 0, \frac{1}{2}-\epsilon)$. By Proposition 4, the local bisimilarity distance is $d_{local}(s_1,t_1) = \frac{1}{2}\|(\tau_\epsilon(s_1)(E))_{E\in X} - (\tau_\epsilon(t_1)(E))_{E\in X}\|_1 = \frac{1}{2}$. Similarly, we have $d_{local}(s_2,t_2) = \frac{1}{2}$.

Algorithm 2 LMC Minimisation Using Local Bisimilarity Distances.

```
  Input: An LMC $\mathcal{M}' = \langle S, L, \tau_\epsilon, \ell\rangle$, a compression parameter $\epsilon_2$
  Output: An LMC $\mathcal{Q}_i$
1 $i := 0$
2 $\mathcal{Q}_i := \mathcal{M}'/_{\sim_{\mathcal{M}'}}$ and $\mathcal{Q}_i = \langle S^{\mathcal{Q}_i}, L, \tau^{\mathcal{Q}_i}, \ell^{\mathcal{Q}_i}\rangle$
3 while $\exists u, v \in S^{\mathcal{Q}_i}$ such that $u \neq v$ and $\ell^{\mathcal{Q}_i}(u) = \ell^{\mathcal{Q}_i}(v)$ and $d^{\mathcal{Q}_i}_{local}(u,v) \leq \epsilon_2$ do
4     $(s,t) = \arg\min\{d^{\mathcal{Q}_i}_{local}(u,v) : (u,v) \in S^{\mathcal{Q}_i} \times S^{\mathcal{Q}_i} \wedge u \neq v \wedge \ell^{\mathcal{Q}_i}(u) = \ell^{\mathcal{Q}_i}(v)\}$
5     Compute $X_i$ by running Algorithm 1 with input $Q_i$ and $(s,t)$
6     Construct an LMC $\mathcal{M}_{i+1} := \langle X_i, L, \tau^{\mathcal{M}_{i+1}}, \ell^{\mathcal{M}_{i+1}}\rangle$ from $\mathcal{Q}_i$ where
        $\tau^{\mathcal{M}_{i+1}}(E) := \begin{cases} (\tau^{\mathcal{Q}_i}(u)(E'))_{E'\in X_i} & \text{for any } u \in E \text{ if } E \in X_i \text{ and } E \neq \{s,t\} \\ \frac{(\tau^{\mathcal{Q}_i}(s)(E'))_{E'\in X_i} + (\tau^{\mathcal{Q}_i}(t)(E'))_{E'\in X_i}}{2} & \text{if } E = \{s,t\} \end{cases}$
        and $\ell^{\mathcal{M}_{i+1}}(E) := \ell^{\mathcal{Q}_i}(u)$ for $E \in X_i$ and any $u \in E$
7     $\mathcal{Q}_{i+1} := \mathcal{M}_{i+1}/_{\sim_{\mathcal{M}_{i+1}}}$
8     $i := i + 1$
9 end
```

Minimisation Algorithm Using Local Bisimilarity Distances

Algorithm 2 shows the minimisation algorithm using local bisimilarity distances. The input is an LMC $\mathcal{M}'$ and a compression parameter ϵ_2. We start by initializing an index i to 0 and building the quotient LMC $\mathcal{Q}_0 = \mathcal{M}'/_{\sim_{\mathcal{M}'}}$. If there are no states in $\mathcal{Q}_i$ with local bisimilarity distance less than ϵ_2, the algorithm terminates. Otherwise, it steps into the i'th iteration of the loop and computes the local bisimilarity distances for all pairs of states in $\mathcal{Q}_i$ with the same label. It selects the state pair (s,t) which has the smallest local bisimilarity distance on line 4. It then computes the new approximate quotient by merging states s and t on line 5-7. This computation is in three steps where the first step is to compute the partition X_i (line 5) by running Algorithm 1 with input $\mathcal{Q}_i$ and the state pair (s,t). The second step is to construct a new LMC $\mathcal{M}_{i+1}$ by setting X_i as its state space (line 6). The final step is to compute a new approximate quotient $\mathcal{Q}_{i+1}$ by taking the exact quotient of the LMC $\mathcal{M}_{i+1}$ obtained from the previous step. We increment i at the end of the iteration and continue with another iteration if there are states in $\mathcal{Q}_{i+1}$ with local bisimilarity distance at most ϵ_2. Since there are finitely many states and it is polynomial time to compute the local bisimilarity distances, the algorithm always terminates and runs in polynomial time.

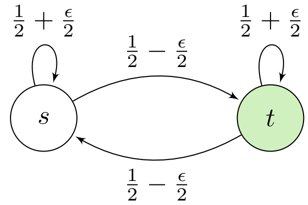
4.2 Minimisation by Approximate Partition Refinement

Consider the LMC in Figure 1. Assume $\epsilon < \frac{1}{2}$ and $\epsilon_2 < \frac{1}{2}$. The minimisation algorithm using local bisimilarity distance (Algorithm 2) cannot merge states s_1, t_1 (or s_2, t_2) as $d_{local}(s_1,t_1) = d_{local}(s_2,t_2) = \frac{1}{2} > \epsilon_2$ as shown by Example 5.

We introduce an approximate partition refinement, a polynomial algorithm similar to the exact partition refinement, which can fix this problem. In the exact partition refinement algorithm, the states will only remain in the same set in an iteration if they have the

$X_0 = \{S\}$
$X_1 = \{\{s_1, t_1\}, \{s_2, t_2\}\}$

(a) The partitions.


$\frac{1}{2}+\frac{\epsilon}{2}$
$\frac{1}{2}-\frac{\epsilon}{2}$
$\frac{1}{2}+\frac{\epsilon}{2}$
s
t
$\frac{1}{2}-\frac{\epsilon}{2}$

(b) The final LMC.

Figure 7 Example of running the minimisation algorithm using approximate partition refinement (Algorithm 3) on the LMC in Figure 1.

same label and their probability distributions over the previous partition are the same. Similarly, we design the approximate partition refinement such that states only remain in the same set in an iteration if they have the same label and the L_1-distance between the probability distributions over the previous partition is small, say, at most ϵ_2. Given an LMC $\mathcal{M}' = \langle S, L, \tau_\epsilon, \ell \rangle$ with perturbed transition probabilities, the minimisation algorithm using the approximate partition refinement also proceeds in iterations. In each iteration, the approximate partition refinement computes a partition X and then the states which are together in X are lumped to form a new LMC. To make sure the new LMC is a quotient, we take the (exact) quotient of this LMC as our new approximate quotient. The algorithm continues when there are states that could be lumped, and it terminates when all sets in the partition computed by the approximate partition refinement are singletons, that is, no states can be lumped.

▶ **Example 6.** Consider again the LMC in Figure 1. Assume $\epsilon < \frac{1}{2}$ and the compression parameter $\epsilon_2 \geq 2\epsilon$. We run the above-mentioned minimisation algorithm using the approximate partition refinement. It will only run for one iteration of approximate partition refinement, as we will see in the following. Figure 7(a) shows the partitions of this iteration. At the beginning of the approximate partition refinement, we have partition X_0 as all states are in the same set. The states are then split by the labels and we get partition X_1. There is no further split since the L_1-distance between the probability distributions over X_1 from s_1 and t_1 (resp. s_2 and t_2) is 2ϵ which is bounded by the compression parameter ϵ_2, that is, $\|(\tau(s_1)(E))_{E\in X_1} - (\tau(t_1)(E))_{E\in X_1}\|_1 = \|(\tau(s_2)(E))_{E\in X_1} - (\tau(t_2)(E))_{E\in X_1}\|_1 = 2\epsilon \leq \epsilon_2$. The states together in X_1 are then lumped to form the new LMC shown in Figure 7(b). The algorithm terminates as no states in the new LMC can be lumped.

Approximate Partition Refinement

Given a compression parameter ϵ_2, the approximate partition refinement is shown in Algorithm 3. At the beginning, an index i is initialized to zero and we have $X_0 = \{S\}$, that is, all states are in the same set. In a refinement step, we increment i and split each set $E \in X_{i-1}$ into one or more sets. We iterate though all $E \in X_{i-1}$ and for each E we construct a set X_E, a partition of E. Starting with $X_E = \emptyset$, we iterate over all $s \in E$ (line 6). After each iteration, the current $s \in E$ appears in one set in X_E: either as a singleton or as an additional state in an already existing set in X_E. We give more details on this loop (lines 6-14) below. After having partitioned E into X_E, we add all sets in X_E to the new

Algorithm 3 Approximate Partition Refinement.

```
   Input: An LMC $\mathcal{M}' = \langle S, L, \tau_\epsilon, \ell \rangle$, a compression parameter $\epsilon_2$
   Output: A partition $X$ over $S$
 1 $i := 0; X_0 := \{S\}$
 2 repeat
 3     $i := i + 1;\ X_i := \emptyset$
 4     foreach $E \in X_{i-1}$ do
 5         $X_E := \emptyset$
 6         for $s \in E$ do
 7             $ESet := \{E' \in X_E :$ for all $t \in E'$ we have $\ell(s) = \ell(t)$ and
                $\|(\tau_\epsilon(s)(E))_{E \in X_{i-1}} - (\tau_\epsilon(t)(E))_{E \in X_{i-1}}\|_1 \leq \epsilon_2\}$
 8             if $ESet = \emptyset$ then $E' := \{s\}$
 9             else
10                 $E' := \arg\min_{E' \in ESet} \{ \frac{\sum_{t \in E'} \|(\tau_\epsilon(s)(E))_{E \in X_{i-1}} - (\tau_\epsilon(t)(E))_{E \in X_{i-1}}\|_1}{|E'|} \}$
11                 remove $E'$ from $X_E$; $E' := E' \cup \{s\}$
12             end
13             add $E'$ to $X_E$
14         end
15         $X_i := X_i \cup X_E$
16     end
17 until $X_i = X_{i-1}$
```

partition X_i. The way we split the sets ensures that for any two states from the same set in X_i the L_1-distance between the successor distributions over X_{i-1} is at most ϵ_2. The algorithm terminates when no splitting can be done. Let X be the final partition produced by the approximate partition refinement. For any two states $s, t \in E$ where $E \in X$, we have $\ell(s) = \ell(t)$ and $\|(\tau_\epsilon(s)(E'))_{E' \in X} - (\tau_\epsilon(t)(E'))_{E' \in X}\|_1 \leq \epsilon_2$.

Let us give more details on the loop (lines 6-14) that partitions an $E \in X_i$. For a state $s \in E$, a candidate set *ESet* is computed such that for all $E' \in ESet$ the state s and all $x \in E'$ have the same label and the L_1-distance between the successor distributions over X_{i-1} of s and any $x \in E'$ is at most ϵ_2 (line 7). If *ESet* is empty, we add the singleton $\{s\}$ into X_E (line 8 and 13). If there is only one set E' in *ESet*, we add s to the set E'. Otherwise, if there are multiple elements in *ESet* that satisfy the condition, we select the one as E' such that the average L_1-distance between the successor distributions of s and $x \in E'$ is the smallest (line 10). We add s to the selected set E' and include E' in X_E (line 10-13).

Minimisation Algorithm Using Approximate Partition Refinement

The minimisation algorithm using approximate partition refinement is shown in Algorithm 4. The input is the same as the first minimisation algorithm: an LMC $\mathcal{M}'$ and a compression parameter ϵ_2. An index i is initialised to 0. Similar to the approximate minimisation algorithm using local bisimilarity distances, we also start by computing the quotient LMC $\mathcal{Q}_0 = \mathcal{M}'/_{\sim_{\mathcal{M}'}}$. It then steps into a loop. We compute the approximate partition X_i of $\mathcal{Q}_i$ on line 4 and construct a new LMC $\mathcal{M}_{i+1}$ by setting X_i as its state space on line 5. For any state $E \in X_i$, we set the probability distribution as the average probability distribution over

Algorithm 4 LMC Minimisation by Approximate Partition Refinement.

```
Input: An LMC $\mathcal{M}' = \langle S, L, \tau_\epsilon, \ell \rangle$, a compression parameter $\epsilon_2$
Output: An LMC $\mathcal{Q}_i$
$i := 0$
$\mathcal{Q}_i := \mathcal{M}_i/_{\sim_{\mathcal{M}_i}}$ where $\mathcal{M}_i = \mathcal{M}'$ and $\mathcal{Q}_i = \langle S^{\mathcal{Q}_i}, L, \tau^{\mathcal{Q}_i}, \ell^{\mathcal{Q}_i} \rangle$
repeat
    Compute $X_i$ by running Algorithm 3 with $\mathcal{Q}_i$ and $\epsilon_2$ as input
    Construct an LMC $\mathcal{M}_{i+1} := \langle X_i, L, \tau^{\mathcal{M}_{i+1}}, \ell^{\mathcal{M}_{i+1}} \rangle$ from $\mathcal{Q}_i$ where
      $\tau^{\mathcal{M}_{i+1}}(E) := \sum_{u \in E} \frac{(\tau^{\mathcal{Q}_i}(u)(E'))_{E' \in X_i}}{|E|}$ and $\ell^{\mathcal{M}_{i+1}}(E) := \ell^{\mathcal{Q}_i}(x)$ for all $E \in X_i$ and
      any $x \in E$
    $\mathcal{Q}_{i+1} := \mathcal{M}_{i+1}/_{\sim_{\mathcal{M}_{i+1}}}$
    $i := i + 1$
until $|S^{\mathcal{Q}_i}| = |S^{\mathcal{Q}_{i-1}}|$
```

X_i from all $u \in E$. The label of any $E \in X$ is set to $\ell^{\mathcal{Q}_i}(u)$ where u can be any state from E. A new approximate quotient $\mathcal{Q}_{i+1}$ is obtained by taking the exact quotient of $\mathcal{M}_{i+1}$. We increment i at the end of the iteration and continue another iteration if the size of the state space of the new approximate quotient decreases. Otherwise, the algorithm terminates as we have no states to merge. As there are finitely many states, the algorithm always terminates.

Let $i \in \mathbb{N}$. The following theorem applies to both the LMCs $\mathcal{Q}_i$ from Algorithm 2 and those from Algorithm 4.

▶ **Theorem 7.** *For all $i \in \mathbb{N}$, we have that $\mathcal{Q}_{i+1}$ is an ϵ_2-quotient of $\mathcal{Q}_i$. Furthermore, by the additivity lemma, we have that $\mathcal{Q}_i$ is an $i\epsilon_2$-quotient of $\mathcal{M}'$.*

In the case that $\mathcal{M}' = \langle S, L, \tau_\epsilon, \ell \rangle$ is a slightly perturbed version of $\mathcal{M} = \langle S, L, \tau, \ell \rangle$, that is, for all $s \in S$ we have $\|\tau(s) - \tau_\epsilon(s)\|_1 \le \epsilon$, the following corollary holds:

▶ **Corollary 8.** *For all $i \in \mathbb{N}$, we have that $\mathcal{Q}_i$ is an $(\epsilon + i\epsilon_2)$-quotient of $\mathcal{M}$.*

5 Active LMC Learning

We apply our approximate minimisation algorithms in a setting of active learning. Before that, we first describe how to obtain a perturbed LMC $\mathcal{M}'$ by sampling. Assume that we want to learn the transition probabilities of an LMC $\mathcal{M}$, that is, the state space, the labelling and the transitions are known. We also assume the system under learning (SUL) $\mathcal{M}$ could answer the query *next* which takes a state s as input and returns a successor state of s according to the transition probability distribution $\tau(s)$.

Given a state s of the LMC. We denote by x_s the number of successor states of s and by n_s the number of times we query the SUL on *next*(s). Let $N_{s,t}$ be the frequency counts of the query result t, that is, the number of times a successor state t appears as the result returned by the queries. We approximate the transition probability distribution by $\tau_\epsilon(s)$ where $\tau_\epsilon(s)(t) = \frac{N_{s,t}}{n_s}$ for all successor states t of s. (Such an estimator is called an empirical estimator in the literature.)

Intuitively, the more queries we ask the SUL, the more accurate the approximate probability distribution $\tau_\epsilon(s)$ would be. In fact, the following theorem holds [2, Section 6.4], [6].

▶ **Theorem 9.** *Let $\epsilon > 0$ be an error parameter and $\delta > 0$ be an error bound. Let $s \in S$. We have $\Pr(\|\tau(s) - \tau_\epsilon(s)\|_1 \le \epsilon) \ge 1 - \delta$ for $n_s \ge \frac{1}{2\epsilon^2} \ln(\frac{2x_s}{\delta})$.*

For each state $s \in S$, we query the SUL on $\mathit{next}(s)$ for $n_s \geq \frac{1}{2\epsilon^2} \ln(\frac{2x_s}{\delta})$ times. We can make δ small since it appears in the logarithmic term. We then approximate the transition function by τ_ϵ and construct a hypothesis LMC $\mathcal{M}' = \langle S, L, \tau_\epsilon, \ell \rangle$. Since the queries $\mathit{next}(s)$ and $\mathit{next}(t)$ for all $s, t \in S$ and $s \neq t$ are mutually independent, by Theorem 9, we have that $\Pr(\forall s \in S : \|\tau(s) - \tau_\epsilon(s)\|_1 \leq \epsilon) \geq (1-\delta)^{|S|}$.

We then apply the minimisation algorithms with compression parameter ϵ_2 on $\mathcal{M}'$ and obtain a minimised system $\mathcal{Q}_i$ which is an $i\epsilon_2$-quotient of $\mathcal{M}'$, the LMC constructed by sampling. Since with high probability the LMC $\mathcal{M}'$ (or its exact quotient $\mathcal{Q}_0$) has small distance ϵ with the SUL $\mathcal{M}$, it follows from Corollary 8 that with high probability the minimised system $\mathcal{Q}_i$ is an ϵ'-quotient of $\mathcal{M}$ where ϵ' is small: for all $i \in \mathbb{N}$, we have $\Pr(\mathcal{Q}_i$ is an ϵ'-quotient of $\mathcal{M}$ with $\epsilon' \leq \epsilon + i\epsilon_2) \geq (1-\delta)^{|S|}$. The probability does not come from our minimisation algorithms and depends solely on the sampling procedure.

6 Experiments

In this section, we evaluate the performance of approximate minimisation algorithms on a number of LMCs. These LMCs model randomised algorithms and probabilistic protocols that are part of the probabilistic model checker PRISM [16]. The LMCs we run experiments on have less than $100,000$ states and model the following protocols or randomised algorithms: Herman's self-stabilisation algorithm [13], the synchronous leader election protocol by Itai and Rodeh [14], the bounded retransmission protocol [7], the Crowds protocol [17] and the contract signing protocol by Even, Goldreich and Lempel [11].

We implemented algorithms to obtain the slightly perturbed LMCs $\mathcal{M}'$. We call LMCs with fewer than 300 states small; otherwise we call them large. For small LMCs, we sample the successor distribution for each state and obtain an approximation of it with error parameter ϵ and error bound δ. For large LMCs, sampling is not practical as the sample size required by Theorem 9 is very large. For these LMCs, we perturb the successor distribution by adding small noise to the successor transition probabilities so that for each state with at least probability $1-\delta$ the L_1-distance of the successor distributions in the perturbed and unperturbed systems is at most ϵ and otherwise the L_1-distance is 2ϵ. We vary the error parameter ϵ in the range of $\{0.00001, 0.0001, 0.001, 0.01\}$ and fix the error bound $\delta = 0.01$. For each unperturbed LMC and a pair of ϵ and δ, we generate 5 perturbed LMCs.

We also implemented the two minimisation algorithms in Java: Algorithm 2 and Algorithm 4. The source code is publicly available[2]. We show some representative results in [15]. The full experimental results are publicly available[3].

For the small LMCs, we apply both approximate minimisation algorithms to the perturbed LMCs with $\epsilon_2 \in \{0.00001, 0.0001, 0.001, 0.01, 0.1\}$. The results for a small LMC which models the Herman's self-stabilisation algorithm is shown on the left of Table 1. For the large LMCs, we only apply the approximate minimisation algorithm using approximate partition refinement to the perturbed LMCs, since the other minimisation algorithm could not finish on the large LMCs with timeout of two hours. The results for a large LMC which models the bounded retransmission protocol is shown on the right of Table 1.

For almost all models, given a perturbed LMC, we are able to recover the structure of the quotient of the unperturbed LMC when ϵ_2 is appropriately chosen, that is, ϵ_2 is no less than ϵ and is not too big; for example, see Table 1 where the rows are highlighted in yellow. However, when ϵ_2 is too big, the approximate minimisation algorithms may aggressively

[2] https://github.com/qiyitang71/approximate-quotienting

[3] https://bit.ly/3vcpblY

Table 1 In the tables, local and apr stand for the minimisation algorithms using local bisimilarity distance and approximate partition refinement, respectively. The tables show the results for the first perturbed LMC (labeled with #1) among the five perturbed LMCs generated by sampling or perturbing with $\epsilon = 0.0001$. (Left) Results of running the two minimisation algorithms on the LMC that models Herman's self-stabilisation algorithm with 5 processes. (Right) Results of running apr on the LMC that models the bounded retransmission protocol with $N = 32$ and $MAX = 2$.

Herman5	# states	# trans	# iter
$\mathcal{M}$ & $\mathcal{M}'$	32	244	
$\mathcal{M}/_{\sim_{\mathcal{M}}}$	4	11	
$\mathcal{M}'/_{\sim_{\mathcal{M}'}}$	23	167	
Perturbed LMC #1			
$\epsilon_2 = 0.00001$			
local & apr	23	167	0
$\epsilon_2 = 0.0001$			
local & apr	22	143	1
$\epsilon_2 \in \{0.001, 0.01, 0.1\}$			
local	22	143	1
apr	4	11	1

BRP32-2	# states	# trans	# iter
$\mathcal{M}$ & $\mathcal{M}'$	1349	1731	
$\mathcal{M}/_{\sim_{\mathcal{M}}}$	647	903	
$\mathcal{M}'/_{\sim_{\mathcal{M}'}}$	961	1343	
Perturbed LMC #1			
$\epsilon_2 = 0.00001$			
apr	879	1230	2
$\epsilon_2 = 0.0001$			
apr	705	986	2
$\epsilon_2 \in \{0.001, 0.01\}$			
apr	647	903	1
$\epsilon_2 = 0.1$			
apr	196	387	1

merge some states in the perturbed LMC and result in a quotient whose size is even smaller than that of the quotient of the unperturbed LMC, as highlighted in red in Table 1. Also, we find that, as expected, the exact partition refinement in general could not recover the structure of quotient of the original LMCs, except for the LMCs which model the synchronous leader election protocol by Itai and Rodeh. Furthermore, compared to the other approximate minimisation algorithm using the local bisimilarity distance, the one using approximate partition refinement performs much better in terms of running time and the ability to recover the structure of the quotient of the original model.

One might ask whether the minimisation algorithm using approximate partition refinement always performs better than the one using the local bisimilarity distances. In general, this is not the case as shown by Example 10.

▶ **Example 10.** Consider the LMC $\mathcal{M} = \langle S, L, \tau, \ell \rangle$ shown in Figure 8. Let $\epsilon_2 = 0.1$. First, we run Algorithm 2. It proceeds in two iterations. In the first iteration, it computes the local bisimilarity distances for all pairs of states with the same label. We have $d_{local}(s_1, s_2) = d_{local}(s_2, s_3) = 0.54$ and $d_{local}(s_1, s_3) = 0.04$. It then selects the pair s_1 and s_3 of which the local bisimilarity distance is less than ϵ_2 and is the smallest. These two states are merged into s_{13} in the LMC shown on the left of Figure 9. In the second iteration, the only pair of states with the same label are s_{13} and s_2. Since $d_{local}(s_{13}, s_2) = 0.06 \leq \epsilon_2$, they are merged and we arrive at the final LMC shown on the right of Figure 9.

Next, we run Algorithm 4 with the same inputs. In the first iteration, we run approximate partition refinement on line 5 (Algorithm 3) and present Table 2 as the possible partitions of the algorithm. At the beginning of the approximate partition refinement, we have partition X_0 as all states are in the same set. The states are then split by the labels and we get partition X_1. Next, we work on the set $\{s_1, s_2, s_3\}$. Suppose that we see s_1 and s_2 before s_3. We have s_1 and s_2 remain together as $\|(\tau(s_1)(E))_{E \in X_1} - (\tau(s_2)(E))_{E \in X_1}\|_1 = 0.08 \leq \epsilon_2$. However, since $\|(\tau(s_3)(E))_{E \in X_1} - (\tau(s_2)(E))_{E \in X_1}\|_1 = 0.16 > \epsilon_2$, we have $ESet = \emptyset$ for s_3 on line 9 of Algorithm 3 and it is split out. In the next iteration, since

$\|(\tau(s_1)(E))_{E\in X_2} - (\tau(s_2)(E))_{E\in X_2}\|_1 = 0.54 > \epsilon_2$, $\{s_1, s_2\}$ is split into two singleton sets. The final partition X_3 in which all sets are singletons suggests no merging can be done and we are left with the original LMC $\mathcal{M}$.

This example also shows that the order of iterating through the states matters for the approximate partition refinement algorithm. Indeed, suppose we iterate though s_1 and s_3 before s_2 after arriving at the partition X_1, we will have Table 3 as the partitions and finally get the LMC on the right of Figure 9 just as the other minimisation algorithm.

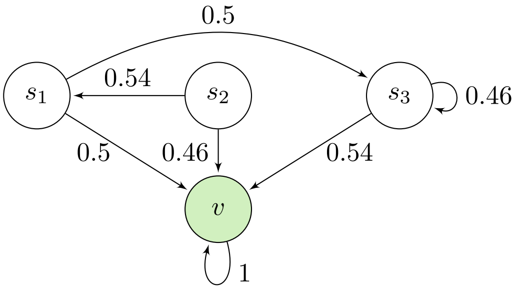


Figure 8 The LMC for which Algorithm 2 may perform better than Algorithm 4.

Table 2 Example of running Algorithm 3 on the LMC in Figure 8. (Suppose we iterate through s_1 and s_2 before s_3.)

$$\begin{aligned} X_0 &= \{S\} \\ X_1 &= \{\{s_1, s_2, s_3\}, \{v\}\} \\ X_2 &= \{\{s_1, s_2\}, \{s_3\}, \{v\}\} \\ X_3 &= \{\{s_1\}, \{s_2\}, \{s_3\}, \{v\}\} \end{aligned}$$

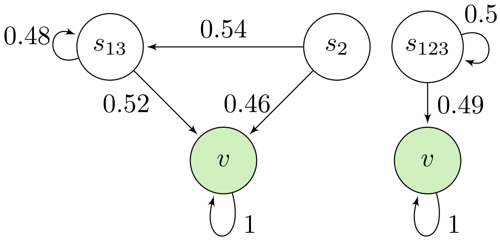


Figure 9 Two Steps of Running Algorithm 2.

Table 3 Example of running Algorithm 3 on the LMC in Figure 8. (Suppose we iterate through s_1 and s_3 before s_2.)

$$\begin{aligned} X_0 &= \{S\} \\ X_1 &= \{\{s_1, s_3, s_2\}, \{v\}\} \\ X_2 &= \{\{s_1, s_3\}, \{s_2\}, \{v\}\} \end{aligned}$$

7 Conclusion

We have developed and analysed algorithms for minimising probabilistic systems via approximate bisimulation. These algorithms are based on ϵ-quotients, a novel yet natural notion of approximate quotients. We have obtained theoretical bounds on the discrepancy between the minimised and the non-minimised systems. In our experiments, approximate partition refinement does well in minimising labelled Markov chains with perturbed transition probabilities, suggesting that approximate partition refinement is a practical approach for "recognising" and exploiting approximate bisimulation.

Future work might consider the following questions: Does approximate minimisation allow for further forms of active learning? Can our techniques be transferred to Markov decision processes?

References

1 Giovanni Bacci, Giorgio Bacci, Kim G. Larsen, and Radu Mardare. On the metric-based approximate minimization of Markov chains. *J. Log. Algebraic Methods Program.*, 100:36–56, 2018. doi:10.1016/j.jlamp.2018.05.006.

2 Hugo Bazille, Blaise Genest, Cyrille Jégourel, and Jun Sun. Global PAC bounds for learning discrete time Markov chains. 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 304–326. Springer, 2020. doi:10.1007/978-3-030-53291-8_17.

3 Gaoang Bian and Alessandro Abate. On the relationship between bisimulation and trace equivalence in an approximate probabilistic context. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 321–337, 2017. doi:10.1007/978-3-662-54458-7_19.

4 Patrick Billingsley. *Probability and measure.* Wiley Series in Probability and Statistics. Wiley, New York, NY, USA, 3rd edition, 1995.

5 Di Chen, Franck van Breugel, and James Worrell. On the complexity of computing probabilistic bisimilarity. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7213 of *Lecture Notes in Computer Science*, pages 437–451. Springer, 2012. doi:10.1007/978-3-642-28729-9_29.

6 Jianhua Chen. Properties of a new adaptive sampling method with applications to scalable learning. *Web Intelligence*, 13(4):215–227, 2015. doi:10.3233/WEB-150322.

7 P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In L. de Alfaro and S. Gilmore, editors, *Proc. 1st Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modelling and Verification (PAPM/PROBMIV'01)*, volume 2165 of *LNCS*, pages 39–56. Springer, 2001.

8 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labeled Markov systems. In Jos Baeten and Sjouke Mauw, editors, *Proceedings of the 10th International Conference on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 258–273, Eindhoven, The Netherlands, August 1999. Springer-Verlag.

9 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theor. Comput. Sci.*, 318(3):323–354, 2004. doi:10.1016/j.tcs.2003.09.013.

10 Josée Desharnais, François Laviolette, and Mathieu Tracol. Approximate analysis of probabilistic processes: Logic, simulation and games. In *Fifth International Conference on the Quantitative Evaluaiton of Systems (QEST 2008), 14-17 September 2008, Saint-Malo, France*, pages 264–273. IEEE Computer Society, 2008. doi:10.1109/QEST.2008.42.

11 S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

12 Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. The probabilistic model checker Storm. *CoRR*, abs/2002.07080, 2020. arXiv:2002.07080.

13 T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35(2):63–67, 1990.

14 Aron Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, September 1990.

15 Stefan Kiefer and Qiyi Tang. Approximate bisimulation minimisation, 2021. arXiv:2110.00326.

16 Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591, Snowbird, UT, USA, July 2011. Springer-Verlag. doi:10.1007/978-3-642-22110-1_47.

17 M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC*, 1(1):66–92, 1998.

18 Qiyi Tang and Franck van Breugel. Deciding probabilistic bisimilarity distance one for labelled markov chains. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 681–699. Springer, 2018.

19 Mathieu Tracol, Josée Desharnais, and Abir Zhioua. Computing distances between probabilistic automata. In Mieke Massink and Gethin Norman, editors, *Proceedings Ninth Workshop on Quantitative Aspects of Programming Languages, QAPL 2011, Saarbrücken, Germany, April 1-3, 2011*, volume 57 of *EPTCS*, pages 148–162, 2011. doi:10.4204/EPTCS.57.11.

Simple Derivation Systems for Proving Sufficient Completeness of Non-Terminating Term Rewriting Systems

Kentaro Kikuchi
Tohoku University, Sendai, Japan

Takahito Aoto
Niigata University, Niigata, Japan

Abstract

A term rewriting system (TRS) is said to be sufficiently complete when each function yields some value for any input. Proof methods for sufficient completeness of terminating TRSs have been well studied. In this paper, we introduce a simple derivation system for proving sufficient completeness of possibly non-terminating TRSs. The derivation system consists of rules to manipulate a set of guarded terms, and sufficient completeness of a TRS holds if there exists a successful derivation for each function symbol. We also show that variations of the derivation system are useful for proving special cases of local sufficient completeness of TRSs, which is a generalised notion of sufficient completeness.

2012 ACM Subject Classification Theory of computation → Rewrite systems; Theory of computation → Equational logic and rewriting

Keywords and phrases Term rewriting, Sufficient completeness, Local sufficient completeness, Non-termination, Derivation rule, Well-founded induction schema

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.49

Funding This work was partially supported by JSPS KAKENHI Grant Numbers JP17K00005, JP19K11891, JP20H04164 and JP21K11750.

Acknowledgements We are grateful to the anonymous reviewers for valuable comments.

1 Introduction

This paper addresses a kind of reachability problem in transformation of labelled trees (i.e. terms) by rules schematised as term rewriting systems (TRSs). The main concern is whether all ground terms (i.e. terms without variables) can be transformed into terms consisting only of special labels called constructors. When the problem is solved positively, the TRS is said to be *sufficiently complete*. This property is useful in automated inductive theorem proving of TRSs, and has largely been studied. One of the sufficient conditions for sufficient completeness of a TRS is that it is terminating (strongly normalising) and quasi-reducible. For terminating TRSs, various decision procedures of sufficient completeness have been proposed [2, 9, 11].

On the other hand, only a few results [3, 4, 17] have been known about proof methods for sufficient completeness of non-terminating TRSs. In recent work [10], the authors proposed a framework for proving inductive theorems of possibly non-terminating TRSs. It is based on a generalised notion of sufficient completeness, called *local sufficient completeness*, where the problem concerns not all ground terms but only terms of specific form, specific sort, etc. In [10], the authors introduced a derivation system for proving local sufficient completeness, but it involves complicated notations and rules with complicated side conditions. In later work [15], the authors gave a proof method based on a sufficient condition for local sufficient

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 49; pp. 49:1–49:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

completeness. The method is applicable to some non-terminating TRSs for which the property is difficult to show by the derivation system of [10]. However, the method of [15] works only for TRSs that consist of functions on natural numbers and lists of natural numbers.

In the present paper, we introduce a simple derivation system for proving sufficient completeness of possibly non-terminating TRSs. The derivation system has rules to manipulate a set of guarded terms, which are pairs of a term and a set of terms. Although the meaning of a guarded term is not easy to grasp, we give its interpretation by introducing a notion of *well-founded induction schema*. This notion plays an important role in the proof of the correctness of our method based on the derivation system. It is easy to apply the method to various non-terminating TRSs, and we give some examples of application of it.

We also introduce variations of the derivation system for proving two particular cases of local sufficient completeness: local sufficient completeness with signature restriction and local sufficient completeness with sort partition. Apart from the simplicity of the systems, our method is different from the method of [10] in that a group of function symbols are simultaneously tested for existence of successful derivations. In this respect, our proof method can be seen as a natural extension of checking quasi-reducibility in the case of terminating TRSs, i.e., it not only checks reducibility of $f(t_1, \ldots, t_n)$ for each function symbol f but also traces results of reduction to terms to which the induction hypothesis can be applied.

Related work. The notion of sufficient completeness was originally introduced in [6, 7]. Since then, numerous works have treated the property in the fields of algebraic specification and term rewriting. In the literature, sufficient completeness has often been defined not w.r.t. reduction but w.r.t. conversion. Sufficient completeness w.r.t. reduction, as in the present paper, was introduced in [11]. In most cases, efforts have been devoted to TRSs consisting of those functions for which transformations by reduction rules are always terminating.

In [17], Toyama studied sufficient completeness w.r.t. reduction in the light of a more general notion of "reachability". He also gave a proof method for reachability, and applied it to some examples of left-linear non-terminating TRSs. Some of those examples are related to local sufficient completeness with signature restriction in terms of the present paper.

In [3, 4], Gnaedig and Kirchner studied sufficient completeness w.r.t. reduction (called $\mathcal{C}$-reducibility) of possibly non-terminating TRSs. They treated only usual sufficient completeness, and did not address any kind of local sufficient completeness. Although their system involves notions of abstract variables and narrowing, the process of proving sufficient completeness has some similarity with ours. We give an example of a TRS for which the method of [3, 4] does not work but our method works.

Organisation of the paper. The paper is organised as follows. In Section 2, we explain basic notions and notations of term rewriting. In Section 3, we introduce a derivation system for proving sufficient completeness of TRSs. In Section 4, we discuss local sufficient completeness with signature restriction. In Section 5, we discuss local sufficient completeness with sort partition. In Section 6, we conclude with suggestions for further work.

2 Preliminaries

In this section, we introduce some notations and notions from the field of term rewriting. For detailed information about term rewriting, see, e.g. [1, 14, 16].

A *many-sorted signature* is given by a non-empty finite set $\mathcal{S}$ of *sorts* and a finite set $\mathcal{F}$ of *function symbols*; each $f \in \mathcal{F}$ is equipped with its *sort declaration* $f : \alpha_1 \times \cdots \times \alpha_n \to \alpha_0$ where $\alpha_0, \ldots, \alpha_n \in \mathcal{S}$ ($n \geq 0$). We also use $f : \alpha_1 \times \cdots \times \alpha_n \to \alpha_0$ to mean that f is equipped with the sort declaration, or to denote such a function symbol f itself. We use $\mathcal{V}$ to denote the set of *variables* where $\mathcal{F} \cap \mathcal{V} = \emptyset$ and each $x \in \mathcal{V}$ has a unique sort $\alpha \in \mathcal{S}$. The set of variables with sort α is denoted by $\mathcal{V}^\alpha$. Then the set $T^\alpha(\mathcal{F}, \mathcal{V})$ of *terms of sort* α is defined inductively as follows:

1. If $x \in \mathcal{V}^\alpha$ then $x \in T^\alpha(\mathcal{F}, \mathcal{V})$.
2. If $f \in \mathcal{F}$, $f : \alpha_1 \times \cdots \times \alpha_n \to \alpha$ and $t_i \in T^{\alpha_i}(\mathcal{F}, \mathcal{V})$ for each i $(1 \leq i \leq n)$ then $f(t_1, \ldots, t_n) \in T^\alpha(\mathcal{F}, \mathcal{V})$.

We define $T(\mathcal{F}, \mathcal{V}) = \bigcup_{\alpha \in \mathcal{S}} T^\alpha(\mathcal{F}, \mathcal{V})$, and $sort(t) = \alpha$ for each $t \in T^\alpha(\mathcal{F}, \mathcal{V})$.

For a term $t = f(t_1, \ldots, t_n)$, its *root symbol* f is denoted by $root(t)$. The set of variables in a term t is denoted by $\mathcal{V}(t)$. A term t is *ground* if $\mathcal{V}(t) = \emptyset$; the set of ground terms is denoted by $T(\mathcal{F})$. We write $f(\vec{x})$ for a term $f(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are distinct variables.

A *context* is a term $C \in T(\mathcal{F} \cup \{\Box^\alpha \mid \alpha \in \mathcal{S}\}, \mathcal{V})$ where $\mathcal{F} \cap \{\Box^\alpha \mid \alpha \in \mathcal{S}\} = \emptyset$ and the special symbol $\Box^\alpha$, called a *hole*, is a term of sort α. A context C with only one hole is denoted by $C[\,]$, and $C[t]$ denotes the term obtained by filling the hole with a term t of the same sort. If $s = C[t]$ for some context $C[\,]$, then t is a *subterm* of s, denoted by $t \trianglelefteq s$.

A *substitution* is a mapping $\theta : \mathcal{V} \to T(\mathcal{F}, \mathcal{V})$ such that $sort(x) = sort(\theta(x))$ for every $x \in \mathcal{V}$, and $dom(\theta) = \{x \in \mathcal{V} \mid \theta(x) \neq x\}$ is finite. A substitution θ is *ground* if $\theta(x) \in T(\mathcal{F})$ for every $x \in dom(\theta)$. The term obtained by applying a substitution θ to a term t is written as $t\theta$. If θ_g is a ground substitution and $\mathcal{V}(t) \subseteq dom(\theta_g)$, the ground term $t\theta_g$ is called a *ground instance* of t. We sometimes write $f(\vec{t})$ for a term $f(\vec{x})\theta$ where $\vec{t}$ is a sequence of terms $x_1\theta, \ldots, x_n\theta$.

A *rewrite rule*, written as $l \to r$, is an ordered pair of terms l and r such that $l \notin \mathcal{V}$, $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ and $sort(l) = sort(r)$. A *term rewriting system* (*TRS*, for short) is a finite set of rewrite rules. For a TRS $\mathcal{R}$, the binary relation $\to_\mathcal{R}$ on $T(\mathcal{F}, \mathcal{V})$ is defined by $s \to_\mathcal{R} t$ iff $s = C[l\theta]$ and $t = C[r\theta]$ for some $l \to r \in \mathcal{R}$, some context $C[\,]$ and some substitution θ. The reflexive transitive closure of $\to_\mathcal{R}$ is denoted by $\xrightarrow{*}_\mathcal{R}$. A term s is in *normal form* if $s \to_\mathcal{R} t$ for no term t. The set of terms in normal form is denoted by $NF(\mathcal{R})$.

Let $\mathcal{R}$ be a TRS. The set $\mathcal{D}$ of *defined symbols* is given by $\mathcal{D} = \{root(l) \mid l \to r \in \mathcal{R}\}$, and the set $\mathcal{C}$ of *constructors* is given by $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. Terms in $T(\mathcal{C}, \mathcal{V})$ are called *constructor terms*, and terms in $T(\mathcal{C})$ are called *ground constructor terms*.

Now we define the notion of sufficient completeness w.r.t. reduction.

▶ **Definition 1** (Sufficient completeness)**.** A TRS $\mathcal{R}$ is *sufficiently complete* for a ground term $t_g \in T(\mathcal{F})$, denoted by $SC(t_g)$, if there exists a ground constructor term $s_g \in T(\mathcal{C})$ such that $t_g \xrightarrow{*}_\mathcal{R} s_g$. $\mathcal{R}$ is *(globally) sufficiently complete* if $SC(t_g)$ for every ground term $t_g \in T(\mathcal{F})$.

Let $\mathcal{R}$ be a TRS. $\mathcal{R}$ is *terminating* if there exists no infinite sequence $t_0 \to_\mathcal{R} t_1 \to_\mathcal{R} \cdots$. $\mathcal{R}$ is *quasi-reducible* if $f(t_1, \ldots, t_n) \notin NF(\mathcal{R})$ for every $f(t_1, \ldots, t_n) \in T(\mathcal{F})$ with $f \in \mathcal{D}$ and $t_1, \ldots, t_n \in T(\mathcal{C})$. The next proposition provides a criterion of sufficient completeness of $\mathcal{R}$. (For its proof, see, e.g. Proposition 2.4 of [10].)

▶ **Proposition 2.** *Let $\mathcal{R}$ be a terminating TRS. Then, $\mathcal{R}$ is sufficiently complete if and only if $\mathcal{R}$ is quasi-reducible.*

In this paper we do not use the above proposition, but the proof method by applying our derivation system can be seen as an extension of checking quasi-reducibility.

Next we introduce some notions on orders. A (*strict*) *partial order* is a binary relation that is irreflexive and transitive. A partial order $\succ$ on terms is *monotonic* if it is closed under context, i.e. $s \succ t$ implies $C[s] \succ C[t]$ for every context $C[\,]$; it is *stable* if it is closed under substitution, i.e. $s \succ t$ implies $s\theta \succ t\theta$ for every substitution θ; it is *well-founded* if there exists no infinite descending chain $t_0 \succ t_1 \succ \cdots$; it has *the subterm property* if $s \trianglelefteq t$ and $s \neq t$ imply $t \succ s$.

A well-founded monotonic stable partial order with the subterm property is called a *simplification order*, and many methods for constructing such an order are known (cf. [1, 16]).

3 A Simple Derivation System for Sufficient Completeness

In this section, we present a derivation system for proving sufficient completeness of TRSs. We illustrate the proof method by applying it to some non-terminating TRSs.

In the following, we use a lexicographic path order as an order that is required to define the derivation system.

▶ **Definition 3** (Lexicographic path order). Let $>$ be a partial order, called a *precedence*, on the set $\mathcal{F}$ of function symbols. The *lexicographic path order* $>_{lpo}$ on $T(\mathcal{F}, \mathcal{V})$ induced by the precedence $>$ is defined inductively as follows: $s >_{lpo} t$ iff

1. $t \in \mathcal{V}(s)$ and $s \neq t$, or
2. $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and
   a. there exists i $(1 \leq i \leq m)$ such that $s_i >_{lpo} t$ or $s_i = t$, or
   b. $f > g$ and $s >_{lpo} t_j$ for every j $(1 \leq j \leq n)$, or
   c. $f = g$, $s >_{lpo} t_j$ for every j $(1 \leq j \leq n)$, and there exists i $(1 \leq i \leq m,\ i \leq n)$ such that $s_1 = t_1$, …, $s_{i-1} = t_{i-1}$ and $s_i >_{lpo} t_i$.

Lexicographic path orders have the following properties.

▶ **Proposition 4.** *Every lexicographic path order $>_{lpo}$ induced by any precedence $>$ on $\mathcal{F}$ is a simplification order. Furthermore, $s >_{lpo} t$ implies $\mathcal{V}(t) \subseteq \mathcal{V}(s)$ for every $s, t \in T(\mathcal{F}, \mathcal{V})$.*

▶ **Lemma 5.** *Let $>$ be a precedence on $\mathcal{F}$ such that $f > g$ for every $f \in \mathcal{D}$ and $g \in \mathcal{C}$. Then the lexicographic path order $>_{lpo}$ induced by $>$ satisfies $s_g >_{lpo} t_g$ for every $s_g \in T(\mathcal{F}) \setminus T(\mathcal{C})$ and $t_g \in T(\mathcal{C})$.*

Next we introduce the derivation system, which acts on a set of guarded terms.

▶ **Definition 6** (Guarded term). A *guarded term*, denoted by $t|H$, consists of a term t and a set H of terms. We write $H\theta$ for the set $\{u\theta \mid u \in H\}$.

A derivation starts from a singleton set consisting of a guarded term of the form $\{t|\{t\}\}$. Intuitively, it means the premise of well-founded induction for ground instances of t with respect to the order $>_{lpo}$. Derivation rules subsequently transform the set of guarded terms[1], preserving the meaning of the well-founded induction schema (Definition 10).

▶ **Definition 7** (Derivation). Let $\mathcal{R}$ be a TRS, and let $>_{lpo}$ be a lexicographic path order induced by some precedence $>$ on $\mathcal{F}$ such that $f > g$ for every $f \in \mathcal{D}$ and $g \in \mathcal{C}$.

- The derivation rules of the system are listed in Figure 1. It derives from a set of guarded terms (given at the upper side) a set of guarded terms (given at the lower side) if the side condition is satisfied.
- For sets Γ, Γ' of guarded terms, we write $\Gamma \rightsquigarrow \Gamma'$ if Γ' is derived from Γ by one of the derivation rules. The reflexive transitive closure of $\rightsquigarrow$ is written as $\overset{*}{\rightsquigarrow}$.

Decompose

$$\frac{\Gamma \cup \{f(t_1,\dots,t_n)|H\}}{\Gamma \cup \{t_1|H,\dots,t_n|H\}}\ f \in \mathcal{C}$$

Expand

$$\frac{\Gamma \cup \{t|H\}}{\Gamma \cup \{t\sigma_i|H\sigma_i\}_i}\quad \begin{array}{l}\{\sigma_i\}_i = \{\{x \mapsto f(\vec{x})\} \mid f \in \mathcal{C},\ sort(x) = sort(f(\vec{x}))\} \\ \text{where } x \in \mathcal{V}(t) \text{ and } \vec{x} \text{ is a sequence of fresh variables}\end{array}$$

Simplify

$$\frac{\Gamma \cup \{t|H\}}{\Gamma \cup \{s|H\}}\ t \to_{\mathcal{R}} s$$

Delete

$$\frac{\Gamma \cup \{t|H\}}{\Gamma}\ \exists u \in H.\ t <_{lpo} u$$

Figure 1 Derivation rules for proving sufficient completeness.

The *Expand* rule substitutes a variable in t by each pattern with a constructor as its root symbol, and yields the same number of guarded terms as the constructors. (The index i ranges over a set isomorphic to $\{f \in \mathcal{C} \mid sort(x) = sort(f(\vec{x}))\}$.) The *Delete* rule removes the guarded term $t|H$ if t is less than some $u \in H$ with respect to $>_{lpo}$.

We have a lemma on preservation of variables occurring in guarded terms.

▶ **Definition 8** (VP). *For a set Γ of guarded terms, $VP(\Gamma)$ means that for every $t|H \in \Gamma$ and every $x \in \mathcal{V}(t)$, there exists $u \in H \setminus \mathcal{V}$ such that $x \in \mathcal{V}(u)$.*

▶ **Lemma 9.** *If $\Gamma \rightsquigarrow \Gamma'$ and $VP(\Gamma)$ then $VP(\Gamma')$.*

The predicate about the well-founded induction schema is defined as follows.

▶ **Definition 10** (WIS). *For a set Γ of guarded terms, $WIS(\Gamma)$ means that for every $t|H \in \Gamma$ and every ground substitution σ_g, the following holds:*

$$(\forall u \in H.\ \forall w_g <_{lpo} u\sigma_g.\ SC(w_g)) \Rightarrow SC(t\sigma_g). \tag{WIS 1}$$

Now we prove a key lemma on the well-founded induction schema.

▶ **Lemma 11.** *Let $\Gamma \rightsquigarrow \Gamma'$ and $VP(\Gamma)$. Then, $WIS(\Gamma')$ implies $WIS(\Gamma)$.*

Proof. Let $\Gamma \rightsquigarrow \Gamma'$ and $VP(\Gamma)$. We prove that if

$$(\forall u \in H'.\ \forall w_g <_{lpo} u\sigma_g.\ SC(w_g)) \Rightarrow SC(t'\sigma_g)$$

for every $t'|H' \in \Gamma'$ and every ground substitution σ_g, then

$$(\forall u \in H.\ \forall w_g <_{lpo} u\sigma_g.\ SC(w_g)) \Rightarrow SC(t\sigma_g)$$

for every $t|H \in \Gamma$ and every ground substitution σ_g. The proof is by case analysis depending on the rule used in the derivation step $\Gamma \rightsquigarrow \Gamma'$.

[1] Actually, for any guarded term $t|H$ in a derivation starting with the form $\{t|\{t\}\}$, a singleton set H is sufficient, but we prove lemmas in the general setting for future developments.

(*Decompose*) Then $\Gamma = \Sigma \cup \{f(t_1,\ldots,t_n)|H'\}$ and $\Gamma' = \Sigma \cup \{t_1|H',\ldots,t_n|H'\}$, where $f \in \mathcal{C}$. The case $t|H \in \Sigma$ follows directly from the hypothesis for Γ'. Thus, it remains to show the case $t|H = f(t_1,\ldots,t_n)|H'$. Let σ_g be a ground substitution such that $\forall u \in H.\ \forall w_g <_{lpo} u\sigma_g.\ SC(w_g)$. Then, by the hypothesis for Γ', we have $SC(t_i\sigma_g)$ for every i $(1 \le i \le n)$. Thus, since $f \in \mathcal{C}$, we have $SC(f(t_1,\ldots,t_n)\sigma_g)$.

(*Expand*) Then $\Gamma = \Sigma \cup \{t'|H'\}$ and $\Gamma' = \Sigma \cup \{t'\sigma_i|H'\sigma_i\}_i$, where $x \in \mathcal{V}(t')$, $\vec{x}$ is a sequence of fresh variables, and $\{\sigma_i\}_i = \{\{x \mapsto f(\vec{x})\} \mid f \in \mathcal{C},\ \mathit{sort}(x) = \mathit{sort}(f(\vec{x}))\}$. The case $t|H \in \Sigma$ follows directly from the hypothesis for Γ'. Thus, it remains to show the case $t|H = t'|H'$. Let σ_g be a ground substitution such that (β): $\forall u \in H.\ \forall w_g <_{lpo} u\sigma_g.\ SC(w_g)$. Our aim is to show $SC(t\sigma_g)$. For this, we distinguish two cases.

1. Suppose that there exists an index i such that $t\sigma_g = (t\sigma_i)\sigma'_g$ for some σ'_g. Then, by $t\sigma_i|H\sigma_i \in \Gamma'$, we know from the hypothesis for Γ' that if $\forall u \in H\sigma_i.\ \forall w_g <_{lpo} u\sigma'_g.\ SC(w_g)$ then $SC((t\sigma_i)\sigma'_g)$. Since $t\sigma_g = (t\sigma_i)\sigma'_g$, it remains to show $\forall u \in H\sigma_i.\ \forall w_g <_{lpo} u\sigma'_g.\ SC(w_g)$. Suppose $u \in H\sigma_i$ and $w_g <_{lpo} u\sigma'_g$. Then, there exists $\hat{u} \in H$ such that $u = \hat{u}\sigma_i$, and we have $w_g <_{lpo} u\sigma'_g = (\hat{u}\sigma_i)\sigma'_g = \hat{u}\sigma_g$. Thus, $SC(w_g)$ holds by our assumption (β).
2. Otherwise. Then we have $t\sigma_g = (t\theta)\sigma'_g$ for some $\theta = \{x \mapsto f(\vec{y})\}$ with $f \in \mathcal{D}$. By $VP(\Gamma)$ and the subterm property of $>_{lpo}$, we have $x <_{lpo} u$ for some $u \in H$, and so by the stability of $>_{lpo}$, we have $x\sigma_g <_{lpo} u\sigma_g$ for some $u \in H$. Hence by our assumption (β), we obtain $SC(x\sigma_g)$. Then by the definition, we know there exists a ground constructor term $g(\vec{w}_g) \in T(\mathcal{C})$ such that $x\sigma_g \xrightarrow{*}_{\mathcal{R}} g(\vec{w}_g)$. From $g \in \mathcal{C}$, there is an index i such that $\sigma_i = \{x \mapsto g(\vec{x})\}$. As $\vec{x}$ is fresh, we may assume $x\sigma_g \xrightarrow{*}_{\mathcal{R}} g(\vec{w}_g) = (x\sigma_i)\sigma'_g$. Hence $t\sigma_g \xrightarrow{*}_{\mathcal{R}} (t\sigma_i)\sigma'_g$. Also, we have $x\sigma_g >_{lpo} (x\sigma_i)\sigma'_g$ by Lemma 5.

   The reminder of the proof proceeds in a similar way to 1 except that $t\sigma_g \xrightarrow{*}_{\mathcal{R}} (t\sigma_i)\sigma'_g$ instead of $t\sigma_g = (t\sigma_i)\sigma'_g$. By $t\sigma_i|H\sigma_i \in \Gamma'$, we know from the hypothesis for Γ' that if $\forall u \in H\sigma_i.\ \forall w_g <_{lpo} u\sigma'_g.\ SC(w_g)$ then $SC((t\sigma_i)\sigma'_g)$, which implies $SC(t\sigma_g)$ since $t\sigma_g \xrightarrow{*}_{\mathcal{R}} (t\sigma_i)\sigma'_g$. Thus, it remains to show $\forall u \in H\sigma_i.\ \forall w_g <_{lpo} u\sigma'_g.\ SC(w_g)$. Suppose $u \in H\sigma_i$ and $w_g <_{lpo} u\sigma'_g$. Then, there exists $\hat{u} \in H$ such that $u = \hat{u}\sigma_i$, and we have $w_g <_{lpo} u\sigma'_g = (\hat{u}\sigma_i)\sigma'_g \le_{lpo} \hat{u}\sigma_g$, where the last part follows from $x\sigma_g >_{lpo} (x\sigma_i)\sigma'_g$ by the monotonicity of $>_{lpo}$ (or $(\hat{u}\sigma_i)\sigma'_g = \hat{u}\sigma_g$ if $x \notin \mathcal{V}(\hat{u})$). Hence $SC(w_g)$ holds by our assumption (β).

(*Simplify*) Then $\Gamma = \Sigma \cup \{t'|H'\}$ and $\Gamma' = \Sigma \cup \{s'|H'\}$, where $t' \to_{\mathcal{R}} s'$. The case $t|H \in \Sigma$ follows directly from the hypothesis for Γ'. Thus, it remains to show the case $t|H = t'|H'$. Let σ_g be a ground substitution such that $\forall u \in H.\ \forall w_g <_{lpo} u\sigma_g.\ SC(w_g)$. Then, by the hypothesis for Γ', we have $SC(s'\sigma_g)$. Since $t\sigma_g = t'\sigma_g \to_{\mathcal{R}} s'\sigma_g$, $SC(t\sigma_g)$ clearly holds.

(*Delete*) Then $\Gamma = \Gamma' \cup \{t'|H'\}$, where $t' <_{lpo} u$ for some $u \in H'$. The case $t|H \in \Gamma'$ follows directly from the hypothesis for Γ'. Thus, it remains to show the case $t|H = t'|H'$. Let σ_g be a ground substitution such that $\forall u \in H.\ \forall w_g <_{lpo} u\sigma_g.\ SC(w_g)$. Since $t = t' <_{lpo} u$ for some $u \in H' = H$, we have $t\sigma_g <_{lpo} u\sigma_g$ for some $u \in H$. Hence $SC(t\sigma_g)$ holds. ◀

Now we are ready to show the theorem on global sufficient completeness of a TRS. In the following proof of the theorem, we use the fact that every ground term t_g has the form $h(\vec{x})\theta_g$ for some $h \in \mathcal{F}$ and some ground substitution θ_g.

▶ **Theorem 12.** *Let $\mathcal{R}$ be a TRS, and let $>_{lpo}$ be a lexicographic path order induced by some precedence $>$ on $\mathcal{F}$ such that $f > g$ for every $f \in \mathcal{D}$ and $g \in \mathcal{C}$. If $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow} \{\,\}$ for every $h \in \mathcal{F}$, then $\mathcal{R}$ is sufficiently complete.*

Proof. Let $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow} \{\,\}$ for every $h \in \mathcal{F}$. We prove $SC(h(\vec{x})\theta_g)$ for every $h \in \mathcal{F}$ and every ground instance $h(\vec{x})\theta_g$ by induction on $T(\mathcal{F})$ with the lexicographic order $>_{lpo}$. Let $h \in \mathcal{F}$ and consider the derivation $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow} \{\,\}$. Then by Lemma 9, $VP(\Gamma)$ holds for every Γ appearing in the derivation. Since $WIS(\{\,\})$ vacuously holds, we have by Lemma 11 $WIS(\{h(\vec{x})|\{h(\vec{x})\}\})$, i.e.,

$$(\forall w_g <_{lpo} h(\vec{x})\theta_g.\ SC(w_g)) \Rightarrow SC(h(\vec{x})\theta_g) \tag{WIS 2}$$

for each ground instance $h(\vec{x})\theta_g$. To prove $SC(h(\vec{x})\theta_g)$, it suffices to show $\forall w_g <_{lpo} h(\vec{x})\theta_g.\ SC(w_g)$. Let $w_g <_{lpo} h(\vec{x})\theta_g$. Since $w_g = f(\vec{y})\rho_g$ for some $f \in \mathcal{F}$ and some ground substitution ρ_g, we have $SC(w_g)$ by the induction hypothesis. Hence $\forall w_g <_{lpo} h(\vec{x})\theta_g.\ SC(w_g)$, and we obtain $SC(h(\vec{x})\theta_g)$. ◀

We give some examples of application of the theorem.

▶ **Example 13** ([4, Example 8.1])**.** Consider a signature with $\mathcal{S} = \{B\}$ and

$$\mathcal{F} = \left\{ \begin{array}{lll} \mathsf{and} : B \times B \to B, & \mathsf{or} : B \times B \to B, & \\ \mathsf{not} : B \to B, & \mathsf{0} : B, & \mathsf{1} : B \end{array} \right\}$$

where $\mathcal{C} = \{\mathsf{0} : B,\ \mathsf{1} : B\}$. Let $\mathcal{R}_1$ be the following TRS:

$$\mathcal{R}_1 = \left\{ \begin{array}{llll} (1) & \mathsf{and}(\mathsf{1}, x) & \to & x \\ (2) & \mathsf{and}(\mathsf{0}, x) & \to & \mathsf{0} \\ (3) & \mathsf{or}(\mathsf{1}, x) & \to & \mathsf{1} \\ (4) & \mathsf{or}(\mathsf{0}, x) & \to & x \\ (5) & \mathsf{and}(\mathsf{1}, x) & \to & \mathsf{not}(\mathsf{not}(\mathsf{and}(\mathsf{1}, x))) \\ (6) & \mathsf{not}(\mathsf{1}) & \to & \mathsf{0} \\ (7) & \mathsf{not}(\mathsf{0}) & \to & \mathsf{1} \\ (8) & \mathsf{not}(\mathsf{and}(x, y)) & \to & \mathsf{or}(\mathsf{not}(x), \mathsf{not}(y)) \end{array} \right\}.$$

Note that $\mathcal{R}_1$ is not terminating since $\mathsf{and}(\mathsf{1}, x) \to_{\mathcal{R}_1} \mathsf{not}(\mathsf{not}(\mathsf{and}(\mathsf{1}, x))) \to_{\mathcal{R}_1} \mathsf{not}(\mathsf{not}(\mathsf{not}(\mathsf{not}(\mathsf{and}(\mathsf{1}, x))))) \to_{\mathcal{R}_1} \cdots$. We show that $\mathcal{R}_1$ is sufficiently complete, using Theorem 12. For this, take a lexicographic path ordering $>_{lpo}$ induced by any precedence $>$ such that $f > g$ for every $f \in \mathcal{D}$ and $g \in \mathcal{C}$. In Figure 2, we give derivations of $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow} \{\,\}$ for $h \in \mathcal{D}$. Also, we have $\{\mathsf{0}|\{\mathsf{0}\}\} \rightsquigarrow \{\,\}$ and $\{\mathsf{1}|\{\mathsf{1}\}\} \rightsquigarrow \{\,\}$ using *Decompose*. Thus by Theorem 12, $\mathcal{R}_1$ is sufficiently complete. ◀

The next example shows that there exists a TRS for which the method of [3, 4] does not work but our method works.

▶ **Example 14.** Let $\mathcal{R}_2$ be the TRS obtained from $\mathcal{R}_1$ of Example 13 by deleting the rule (1). Then $\mathcal{R}_2$ is still sufficiently complete. Indeed, derivations for defined symbols except and are the same as those of Figure 2. For and, we have a derivation of $\{\mathsf{and}(x_1, x_2)|\{\mathsf{and}(x_1, x_2)\}\} \overset{*}{\rightsquigarrow} \{\,\}$ as shown in Figure 3. Note that the *Simplify* step using the rule (8) (the fifth row in the figure) cannot be made by the abstract-narrow-based process of [3, 4] (cf. Appendix A). ◀

$$
\begin{array}{ll}
 & \{\ \mathsf{not}(x)|\{\mathsf{not}(x)\}\ \} \\
\rightsquigarrow_{\mathit{Expand}} & \{\ \mathsf{not}(0)|\{\mathsf{not}(0)\},\ \mathsf{not}(1)|\{\mathsf{not}(1)\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \{\ 1|\{\mathsf{not}(0)\},\ 0|\{\mathsf{not}(1)\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Decompose}} & \{\quad\}
\end{array}
$$

$$
\begin{array}{ll}
 & \{\ \mathsf{or}(x_1,x_2)|\{\mathsf{or}(x_1,x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Expand}} & \{\ \mathsf{or}(0,x_2)|\{\mathsf{or}(0,x_2)\},\ \mathsf{or}(1,x_2)|\{\mathsf{or}(1,x_2)\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \{\ x_2|\{\mathsf{or}(0,x_2)\},\ 1|\{\mathsf{or}(1,x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Decompose}} & \{\ x_2|\{\mathsf{or}(0,x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Delete}} & \{\quad\}
\end{array}
$$

$$
\begin{array}{ll}
 & \{\ \mathsf{and}(x_1,x_2)|\{\mathsf{and}(x_1,x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Expand}} & \{\ \mathsf{and}(0,x_2)|\{\mathsf{and}(0,x_2)\},\ \mathsf{and}(1,x_2)|\{\mathsf{and}(1,x_2)\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \{\ 0|\{\mathsf{and}(0,x_2)\},\ x_2|\{\mathsf{and}(1,x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Decompose}} & \{\ x_2|\{\mathsf{and}(1,x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Delete}} & \{\quad\}
\end{array}
$$

Figure 2 Derivations for proving sufficient completeness of $\mathcal{R}_1$ in Example 13.

$$
\begin{array}{ll}
 & \{\ \mathsf{and}(x_1,x_2)|\{\mathsf{and}(x_1,x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Expand}} & \{\ \mathsf{and}(0,x_2)|\{\mathsf{and}(0,x_2)\},\ \mathsf{and}(1,x_2)|\{\mathsf{and}(1,x_2)\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \{\ 0|\{\mathsf{and}(0,x_2)\},\ \mathsf{not}(\mathsf{not}(\mathsf{and}(1,x_2)))|\{\mathsf{and}(1,x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Decompose}} & \{\ \mathsf{not}(\mathsf{not}(\mathsf{and}(1,x_2)))|\{\mathsf{and}(1,x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Simplify}} & \{\ \mathsf{not}(\mathsf{or}(\mathsf{not}(1),\mathsf{not}(x_2)))|\{\mathsf{and}(1,x_2)\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \{\ \mathsf{not}(\mathsf{not}(x_2))|\{\mathsf{and}(1,x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Expand}} & \{\ \mathsf{not}(\mathsf{not}(0))|\{\mathsf{and}(1,0)\},\ \mathsf{not}(\mathsf{not}(1))|\{\mathsf{and}(1,1)\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \{\ 0|\{\mathsf{and}(1,0)\},\ 1|\{\mathsf{and}(1,1)\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Decompose}} & \{\quad\}
\end{array}
$$

Figure 3 A derivation for proving sufficient completeness of $\mathcal{R}_2$ in Example 14.

The next example is a modification of [17, Example A.2].

▶ **Example 15.** Consider a signature with $\mathcal{S} = \{N\}$ and

$$
\mathcal{F} = \left\{ \begin{array}{lll} \mathsf{d} : N \to N, & \mathsf{if} : N \times N \times N \to N, & \\ - : N \times N \to N, & 0 : N, & \mathsf{s} : N \to N \end{array} \right\}
$$

where $\mathcal{C} = \{0 : N,\ \mathsf{s} : N \to N\}$. Let $\mathcal{R}_3$ be the following TRS where $\mathsf{d}(n)$ computes the double of a given natural number n:

$$
\mathcal{R}_3 = \left\{ \begin{array}{llll}
(1) & \mathsf{d}(x) & \to & \mathsf{if}(x, 0, \mathsf{s}(\mathsf{s}(\mathsf{d}(-(x, \mathsf{s}(0)))))) \\
(2) & \mathsf{if}(0, y, z) & \to & y \\
(3) & \mathsf{if}(\mathsf{s}(x), y, z) & \to & z \\
(4) & -(0, y) & \to & 0 \\
(5) & -(x, 0) & \to & x \\
(6) & -(\mathsf{s}(x), \mathsf{s}(y)) & \to & -(x, y)
\end{array} \right\}.
$$

Note that $\mathcal{R}_3$ is not terminating by repeated application of the rule (1). We show that $\mathcal{R}_3$ is sufficiently complete, using Theorem 12. For this, take a lexicographic path ordering $>_{lpo}$ induced by any precedence $>$ such that $f > g$ for every $f \in \mathcal{D}$ and $g \in \mathcal{C}$. In Figure 4,

$$
\begin{array}{ll}
 & \{\ \mathsf{d}(x)|\{\mathsf{d}(x)\}\ \} \\
\rightsquigarrow_{\mathit{Expand}} & \{\ \mathsf{d}(\mathsf{0})|\{\mathsf{d}(\mathsf{0})\},\ \mathsf{d}(\mathsf{s}(x_1))|\{\mathsf{d}(\mathsf{s}(x_1))\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \{\ \mathsf{0}|\{\mathsf{d}(\mathsf{0})\},\ \mathsf{s}(\mathsf{s}(\mathsf{d}(-(\mathsf{s}(x_1),\mathsf{s}(\mathsf{0})))))|\{\mathsf{d}(\mathsf{s}(x_1))\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Decompose}} & \{\ \mathsf{d}(-(\mathsf{s}(x_1),\mathsf{s}(\mathsf{0})))|\{\mathsf{d}(\mathsf{s}(x_1))\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \{\ \mathsf{d}(x_1)|\{\mathsf{d}(\mathsf{s}(x_1))\}\ \} \\
\rightsquigarrow_{\mathit{Delete}} & \{\quad\}
\end{array}
$$

$$
\begin{array}{ll}
 & \{\ \mathsf{if}(x_1,x_2,x_3)|\{\mathsf{if}(x_1,x_2,x_3)\}\ \} \\
\rightsquigarrow_{\mathit{Expand}} & \{\ \mathsf{if}(\mathsf{0},x_2,x_3)|\{\mathsf{if}(\mathsf{0},x_2,x_3)\},\ \mathsf{if}(\mathsf{s}(x_4),x_2,x_3)|\{\mathsf{if}(\mathsf{s}(x_4),x_2,x_3)\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \{\ x_2|\{\mathsf{if}(\mathsf{0},x_2,x_3)\},\ x_3|\{\mathsf{if}(\mathsf{s}(x_4),x_2,x_3)\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Delete}} & \{\quad\}
\end{array}
$$

$$
\begin{array}{ll}
 & \{\ -(x_1,x_2)|\{-(x_1,x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Expand}} & \{\ -(\mathsf{0},x_2)|\{-(\mathsf{0},x_2)\},\ -(\mathsf{s}(x_3),x_2)|\{-(\mathsf{s}(x_3),x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Simplify}} & \{\ \mathsf{0}|\{-(\mathsf{0},x_2)\},\ -(\mathsf{s}(x_3),x_2)|\{-(\mathsf{s}(x_3),x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Decompose}} & \{\ -(\mathsf{s}(x_3),x_2)|\{-(\mathsf{s}(x_3),x_2)\}\ \} \\
\rightsquigarrow_{\mathit{Expand}} & \{\ -(\mathsf{s}(x_3),\mathsf{0})|\{-(\mathsf{s}(x_3),\mathsf{0})\},\ -(\mathsf{s}(x_3),\mathsf{s}(x_4))|\{-(\mathsf{s}(x_3),\mathsf{s}(x_4))\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \{\ \mathsf{s}(x_3)|\{-(\mathsf{s}(x_3),\mathsf{0})\},\ -(x_3,x_4)|\{-(\mathsf{s}(x_3),\mathsf{s}(x_4))\}\ \} \\
\overset{*}{\rightsquigarrow}_{\mathit{Delete}} & \{\quad\}
\end{array}
$$

Figure 4 Derivations for proving sufficient completeness of $\mathcal{R}_3$ in Example 15.

we give derivations of $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow} \{\,\}$ for $h \in \mathcal{D}$. Also, we have $\{\mathsf{0}|\{\mathsf{0}\}\} \rightsquigarrow \{\,\}$ and $\{\mathsf{s}(x)|\{\mathsf{s}(x)\}\} \rightsquigarrow \{\,\}$ using *Decompose* and *Delete*. Thus by Theorem 12, $\mathcal{R}_3$ is sufficiently complete. ◀

Without the rule (4), the above TRS $\mathcal{R}_3$ still has some kind of sufficient completeness, which we discuss in the next section.

4 A Simple Derivation System for Local Sufficient Completeness with Signature Restriction

In the remainder of the paper, we are concerned with local sufficient completeness [10], which is a generalised notion of sufficient completeness. In this section, we consider local sufficient completeness on the set of ground terms consisting of particular function symbols.

▶ **Definition 16** (Local sufficient completeness with signature restriction)**.** Let $\mathcal{R}$ be a TRS, and let $\mathcal{F}' \subseteq \mathcal{F}$. Then $\mathcal{R}$ is *locally sufficiently complete* on $T(\mathcal{F}')$ if $SC(t_g)$ for every $t_g \in T(\mathcal{F}')$.

The standard notion of sufficient completeness, as given in Definition 1, is the case of local sufficient completeness on $T(\mathcal{F}')$ where $\mathcal{F}' = \mathcal{F}$. By the restriction to $\mathcal{F}'$, we can talk about sufficient completeness locally, e.g. on $T(\{\mathsf{d},\mathsf{s},\mathsf{0}\})$ in Example 15 (cf. Example 25).

The notion of a derivation of the system in this section is defined as follows.

▶ **Definition 17** (Derivation)**.** Let $\mathcal{R}$ be a TRS, and let $\mathcal{F}' \subseteq \mathcal{F}$. Suppose that $>_{lpo}$ is a lexicographic path order induced by some precedence $>$ on $\mathcal{F}$ such that $f > f' > g$ for every $f \in \mathcal{D} \setminus \mathcal{F}'$, $f' \in \mathcal{F}' \setminus \mathcal{C}$ and $g \in \mathcal{C}$. The derivation rules of the system are the same as those listed in Figure 1. We write $\Gamma \rightsquigarrow_{\ell} \Gamma'$ if Γ' is derived from Γ by one of the derivation rules.

▶ **Lemma 18.** *Let $>_{lpo}$ be a lexicographic path order induced by a precedence $>$ as above. If $s_g \in T(\mathcal{F}' \cup \mathcal{C})$ and $s_g >_{lpo} t_g$ then $t_g \in T(\mathcal{F}' \cup \mathcal{C})$.*

As before, we have a lemma on preservation of variables occurring in guarded terms.

▶ **Lemma 19.** *If $\Gamma \rightsquigarrow_\ell \Gamma'$ and $VP(\Gamma)$ then $VP(\Gamma')$.*

In addition, we have a lemma on preservation of function symbols of guarded terms.

▶ **Definition 20** (*SigP*)**.** *For a set Γ of guarded terms, $SigP(\Gamma)$ means that for every $t|H \in \Gamma$ and every $u \in H$, $u \in T(\mathcal{F}' \cup \mathcal{C}, \mathcal{V})$.*

▶ **Lemma 21.** *If $\Gamma \rightsquigarrow_\ell \Gamma'$ and $SigP(\Gamma)$ then $SigP(\Gamma')$.*

The predicate about the well-founded induction schema is defined as follows.

▶ **Definition 22** (*WIS*$_\ell$)**.** *For a set Γ of guarded terms, $WIS_\ell(\Gamma)$ means that for every $t|H \in \Gamma$ and every ground substitution $\sigma_g : \mathcal{V} \to T(\mathcal{F}' \cup \mathcal{C})$, the following holds:*

$$(\forall u \in H.\ \forall w_g <_{lpo} u\sigma_g.\ SC(w_g)) \Rightarrow SC(t\sigma_g). \tag{WIS 3}$$

▶ **Lemma 23.** *Let $\Gamma \rightsquigarrow_\ell \Gamma'$, $VP(\Gamma)$ and $SigP(\Gamma)$. Then, $WIS_\ell(\Gamma')$ implies $WIS_\ell(\Gamma)$.*

Proof. The proof proceeds by case analysis in the same way as that of Lemma 11, except that $\sigma : \mathcal{V} \to T(\mathcal{F}' \cup \mathcal{C})$ is enforced on every ground substitution σ appearing in the proof. ◀

We are now ready to show the theorem on local sufficient completeness on $T(\mathcal{F}')$.

▶ **Theorem 24.** *Let $\mathcal{R}$ be a TRS, $\mathcal{F}' \subseteq \mathcal{F}$ and $>_{lpo}$ be a lexicographic path order induced by some precedence $>$ on $\mathcal{F}$ such that $f > f' > g$ for every $f \in \mathcal{D} \setminus \mathcal{F}'$, $f' \in \mathcal{F}' \setminus \mathcal{C}$ and $g \in \mathcal{C}$. If $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow}_\ell \{\,\}$ for every $h \in \mathcal{F}'$, then $\mathcal{R}$ is locally sufficiently complete on $T(\mathcal{F}')$.*

Proof. Let $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow}_\ell \{\,\}$ for every $h \in \mathcal{F}'$. (Using *Decompose* and *Delete*, we can automatically have $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow}_\ell \{\,\}$ for every $h \in \mathcal{C}$.) It suffices to prove $SC(h(\vec{x})\theta_g)$ for every $h \in \mathcal{F}' \cup \mathcal{C}$ and every ground instance $h(\vec{x})\theta_g$ with $\theta_g : \mathcal{V} \to T(\mathcal{F}' \cup \mathcal{C})$ by induction on $T(\mathcal{F}' \cup \mathcal{C})$ with respect to $>_{lpo}$. This is shown by a similar argument to that of Theorem 12, and we have that $\mathcal{R}$ is locally sufficiently complete on $T(\mathcal{F}' \cup \mathcal{C})$ and thus on $T(\mathcal{F}')$. ◀

Now we consider the example mentioned at the end of the previous section.

▶ **Example 25.** Let $\mathcal{R}_4$ be the TRS obtained from $\mathcal{R}_3$ of Example 15 by deleting the rule (4). Then $\mathcal{R}_4$ is not globally sufficiently complete any more, since $-(0, \mathsf{s}(0)) \in NF(\mathcal{R}_4)$ but $-(0, \mathsf{s}(0)) \notin T(\mathcal{C})$. However, it can be shown that $\mathcal{R}_4$ is locally sufficiently complete on $T(\mathcal{F}')$ where $\mathcal{F}' = \{\mathsf{d}, \mathsf{s}, 0\}$. Indeed, the derivation for the function symbol $\mathsf{d} \in \mathcal{D}$ shown in Figure 4 works where the rule (4) is not used in the *Simplify* steps. The precedence can be given as $\mathsf{if}, - > \mathsf{d} > \mathsf{s}, 0$. Hence by Theorem 24, $\mathcal{R}_4$ is locally sufficiently complete on $T(\mathcal{F}')$. ◀

5 A Simple Derivation System for Local Sufficient Completeness with Sort Partition

In this section, we treat another type of local sufficient completeness than that discussed in the previous section. Specifically, we consider local sufficient completeness on the set of ground terms of particular sorts.

▶ **Definition 26** (Conditions on the signature)**.** We assume the following conditions S1–S4 on the signature of $\mathcal{R}$.

S1. $\mathcal{S} = \mathcal{S}_0 \uplus \mathcal{S}_1$. ($\uplus$ stands for the disjoint union.)

S2. The sets $\mathcal{F}_i$ $(i = 0, 1)$ of function symbols are defined by
$\mathcal{F}_i = \{f \in \mathcal{F} \mid f : \alpha_1 \times \cdots \times \alpha_n \to \alpha,\ \alpha \in \mathcal{S}_i\}$.

S3. The sets T_i $(i = 0, 1)$ of ground terms are defined by
$T_i = \{t_g \in T(\mathcal{F}) \mid \mathit{sort}(t_g) \in \mathcal{S}_i\}$.

S4. For every $g \in \mathcal{F}_0 \cap \mathcal{C}$, if $g : \alpha_1 \times \cdots \times \alpha_n \to \alpha$ then $\alpha_1, \ldots, \alpha_n \in \mathcal{S}_0$.

Our aim in this section is to show that the following holds under certain conditions.

▶ **Definition 27** (Local sufficient completeness with sort partition)**.** Let $\mathcal{R}$ be a TRS with a signature satisfying the conditions S1–S4 of Definition 26. Then $\mathcal{R}$ is said to be *locally sufficiently complete* on T_0 if $SC(t_g)$ for every $t_g \in T_0$.

The next example illustrates the difference between global sufficient completeness and local sufficient completeness on T_0.

▶ **Example 28** ([15, Example 8])**.** Consider a signature with $\mathcal{S}_0 = \{N\}$, $\mathcal{S}_1 = \{L\}$ and

$$\mathcal{F} = \left\{ \begin{array}{llll} \mathsf{sum} : N \times L \to N, & + : N \times N \to N, & \mathsf{from} : N \to L, & \\ \mathsf{0} : N, & \mathsf{s} : N \to N, & [\,] : L, & :: : N \times L \to L \end{array} \right\}$$

where $\mathcal{C} = \{\mathsf{0} : N,\ \mathsf{s} : N \to N,\ [\,] : L,\ :: : N \times L \to L\}$. Let $\mathcal{R}_5$ be the following TRS where $\mathsf{sum}(n, ts)$ computes the summation of the first n elements of a (possibly infinite) list ts of natural numbers:

$$\mathcal{R}_5 = \left\{ \begin{array}{lll} (1) & \mathsf{sum}(\mathsf{0}, xs) & \to \mathsf{0} \\ (2) & \mathsf{sum}(\mathsf{s}(x), [\,]) & \to \mathsf{0} \\ (3) & \mathsf{sum}(\mathsf{s}(x), y :: ys) & \to +(y, \mathsf{sum}(x, ys)) \\ (4) & +(\mathsf{0}, y) & \to y \\ (5) & +(\mathsf{s}(x), y) & \to \mathsf{s}(+(x, y)) \\ (6) & \mathsf{from}(x) & \to x :: \mathsf{from}(\mathsf{s}(x)) \end{array} \right\}.$$

$\mathcal{R}_5$ is not terminating since $\mathsf{from}(\mathsf{0}) \to_{\mathcal{R}} \mathsf{0} :: \mathsf{from}(\mathsf{s}(\mathsf{0})) \to_{\mathcal{R}} \mathsf{0} :: \mathsf{s}(\mathsf{0}) :: \mathsf{from}(\mathsf{s}(\mathsf{s}(\mathsf{0}))) \to_{\mathcal{R}} \cdots$. $\mathcal{R}_5$ is not globally sufficiently complete either since $u \notin T(\mathcal{C})$ for any u with $\mathsf{from}(\mathsf{0}) \xrightarrow{*}_{\mathcal{R}} u$. However, it can be shown that $\mathcal{R}_5$ is locally sufficiently complete on T_0 (cf. Example 35). ◀

The notion of a derivation of the system in this section is defined similarly to Definition 7 except that the *Expand* rule is replaced by

Expand

$$\frac{\Gamma \cup \{t|H\}}{\Gamma \cup \{t\sigma_i|H\sigma_i\}_i} \quad \begin{array}{l} \{\sigma_i\}_i = \{\{x \mapsto f(\vec{x})\} \mid f \in \mathcal{C} \cup \mathcal{F}_1,\ \mathit{sort}(x) = \mathit{sort}(f(\vec{x}))\} \\ \text{where } x \in \mathcal{V}(t) \text{ and } \vec{x} \text{ is a sequence of fresh variables} \end{array}$$

We write $\Gamma \rightsquigarrow_S \Gamma'$ if Γ' is derived from Γ by one of the derivation rules.

▶ **Lemma 29.** *If* $\Gamma \rightsquigarrow_S \Gamma'$ *and* $VP(\Gamma)$ *then* $VP(\Gamma')$*.*

In addition, we have a lemma on preservation of sorts of guarded terms.

▶ **Definition 30** (SrtP)**.** *For a set* Γ *of guarded terms,* $\mathit{SrtP}(\Gamma)$ *means that for every* $t|H \in \Gamma$*,* $\mathit{sort}(t) \in \mathcal{S}_0$ *and* $\mathit{sort}(u) \in \mathcal{S}_0$ *for every* $u \in H$*.*

▶ **Lemma 31.** *If* $\Gamma \rightsquigarrow_S \Gamma'$ *and* $\mathit{SrtP}(\Gamma)$ *then* $\mathit{SrtP}(\Gamma')$*.*

Proof. By case analysis depending on the rule used in the derivation step $\Gamma \rightsquigarrow_S \Gamma'$. In the case of the *Decompose* rule, we use the condition S4 of Definition 26. ◀

The predicate about the well-founded induction schema is defined as follows.

▶ **Definition 32** (*WIS_S*). *For a set Γ of guarded terms, $WIS_S(\Gamma)$ means that for every $t|H \in \Gamma$ and every ground substitution σ_g, the following holds:*

$$(\forall u \in H.\ \forall w_g \in T_0.\ w_g <_{lpo} u\sigma_g \Rightarrow SC(w_g)) \Rightarrow SC(t\sigma_g). \tag{WIS 4}$$

▶ **Lemma 33.** *Let $\Gamma \rightsquigarrow_S \Gamma'$, $VP(\Gamma)$ and $SrtP(\Gamma)$. Then, $WIS_S(\Gamma')$ implies $WIS_S(\Gamma)$.*

Proof. We prove that if

$$(\forall u \in H'.\ \forall w_g \in T_0.\ w_g <_{lpo} u\sigma_g \Rightarrow SC(w_g)) \Rightarrow SC(t'\sigma_g)$$

for every $t'|H' \in \Gamma'$ and every ground substitution σ_g, then

$$(\forall u \in H.\ \forall w_g \in T_0.\ w_g <_{lpo} u\sigma_g \Rightarrow SC(w_g)) \Rightarrow SC(t\sigma_g)$$

for every $t|H \in \Gamma$ and every ground substitution σ_g. The proof proceeds by case analysis depending on the rule used in the derivation step $\Gamma \rightsquigarrow_S \Gamma'$. Here we only consider the cases of *Expand* and *Delete*. The other cases are proved in the same way as those of Lemma 11.

(*Expand*) Then $\Gamma = \Sigma \cup \{t'|H'\}$ and $\Gamma' = \Sigma \cup \{t'\sigma_i|H'\sigma_i\}_i$, where $x \in \mathcal{V}(t')$, $\vec{x}$ is a sequence of fresh variables, and $\{\sigma_i\}_i = \{\{x \mapsto f(\vec{x})\} \mid f \in \mathcal{C} \cup \mathcal{F}_1,\ sort(x) = sort(f(\vec{x}))\}$. The case $t|H \in \Sigma$ follows directly from the hypothesis for Γ'. Thus, it remains to show the case $t|H = t'|H'$. Let σ_g be a ground substitution such that (β): $\forall u \in H.\ \forall w_g \in T_0.\ w_g <_{lpo} u\sigma_g \Rightarrow SC(w_g)$. Our aim is to show $SC(t\sigma_g)$. For this, we distinguish two cases.

1. Suppose that there exists an index i such that $t\sigma_g = (t\sigma_i)\sigma'_g$ for some σ'_g. This case is proved similarly to the same case of the proof of Lemma 11.
2. Otherwise. Then we have $t\sigma_g = (t\theta)\sigma'_g$ for some $\theta = \{x \mapsto f(\vec{y})\}$ with $f \in \mathcal{F}_0 \cap \mathcal{D}$. This case is proved similarly to the case 2 of the proof of Lemma 11.

(*Delete*) Then $\Gamma = \Gamma' \cup \{t'|H'\}$, where $t' <_{lpo} u$ for some $u \in H'$. The case $t|H \in \Gamma'$ follows directly from the hypothesis for Γ'. Thus, it remains to show the case $t|H = t'|H'$. Let σ_g be a ground substitution such that $\forall u \in H.\ \forall w_g \in T_0.\ w_g <_{lpo} u\sigma_g \Rightarrow SC(w_g)$. Since $t = t' <_{lpo} u$ for some $u \in H' = H$, we have $t\sigma_g <_{lpo} u\sigma_g$ for some $u \in H$, and by $SrtP(\Gamma)$, we have $t\sigma_g \in T_0$. Hence $SC(t\sigma_g)$ holds. ◀

We are now ready to show the theorem on local sufficient completeness on T_0.

▶ **Theorem 34.** *Let $\mathcal{R}$ be a TRS with a signature satisfying the conditions S1–S4 of Definition 26, and let $>_{lpo}$ be a lexicographic path order induced by some precedence $>$ on $\mathcal{F}$ such that $f > g$ for every $f \in \mathcal{D}$ and $g \in \mathcal{C}$. If $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow}_S \{\,\}$ for every $h \in \mathcal{F}_0$, then $\mathcal{R}$ is locally sufficiently complete on T_0.*

Proof. Let $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow}_S \{\,\}$ for every $h \in \mathcal{F}_0$. We prove $SC(h(\vec{x})\theta_g)$ for every $h \in \mathcal{F}_0$ and every ground instance $h(\vec{x})\theta_g$ by induction on $T(\mathcal{F})$ with respect to the order $>_{lpo}$. Let $h \in \mathcal{F}_0$ and consider the derivation $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow}_S \{\,\}$. Then by Lemmas 9 and 31, $VP(\Gamma)$ and $SrtP(\Gamma)$ hold for every Γ appearing in the derivation. Since $WIS_S(\{\,\})$ vacuously holds, we have by Lemma 33 $WIS_S(\{h(\vec{x})|\{h(\vec{x})\}\})$, i.e.,

$$(\forall w_g \in T_0.\ w_g <_{lpo} h(\vec{x})\theta_g \Rightarrow SC(w_g)) \Rightarrow SC(h(\vec{x})\theta_g) \tag{WIS 5}$$

for each ground instance $h(\vec{x})\theta_g$. For every $w_g \in T_0$, we have $w_g = f(\vec{y})\rho_g$ for some $f \in \mathcal{F}_0$ and some ground substitution ρ_g. Hence, for every $w_g \in T_0$, $w_g <_{lpo} h(\vec{x})\theta_g$ implies $SC(w_g)$ by the induction hypothesis. Thus, by (WIS 5), we obtain $SC(h(\vec{x})\theta_g)$. ◀

$$
\begin{array}{ll}
 & \{\ \mathsf{sum}(x_1,x_2)|\{\mathsf{sum}(x_1,x_2)\}\ \} \\
\rightsquigarrow_{S\,\mathit{Expand}} & \{\ \mathsf{sum}(\mathsf{0},x_2)|\{\mathsf{sum}(\mathsf{0},x_2)\},\ \mathsf{sum}(\mathsf{s}(x_3),x_2)|\{\mathsf{sum}(\mathsf{s}(x_3),x_2)\}\ \} \\
\rightsquigarrow_{S\,\mathit{Simplify}} & \{\ \mathsf{0}|\{\mathsf{sum}(\mathsf{0},x_2)\},\ \mathsf{sum}(\mathsf{s}(x_3),x_2)|\{\mathsf{sum}(\mathsf{s}(x_3),x_2)\}\ \} \\
\rightsquigarrow_{S\,\mathit{Decompose}} & \{\ \mathsf{sum}(\mathsf{s}(x_3),x_2)|\{\mathsf{sum}(\mathsf{s}(x_3),x_2)\}\ \} \\
\rightsquigarrow_{S\,\mathit{Expand}} & \left\{ \begin{array}{l} \mathsf{sum}(\mathsf{s}(x_3),[\,])|\{\mathsf{sum}(\mathsf{s}(x_3),[\,])\}, \\ \mathsf{sum}(\mathsf{s}(x_3),x_4::x_5)|\{\mathsf{sum}(\mathsf{s}(x_3),x_4::x_5)\}, \\ \mathsf{sum}(\mathsf{s}(x_3),\mathsf{from}(x_6))|\{\mathsf{sum}(\mathsf{s}(x_3),\mathsf{from}(x_6))\} \end{array} \right\} \\
\overset{*}{\rightsquigarrow}_{S\,\mathit{Simplify}} & \left\{ \begin{array}{l} \mathsf{0}|\{\mathsf{sum}(\mathsf{s}(x_3),[\,])\}, \\ +(x_4,\mathsf{sum}(x_3,x_5))|\{\mathsf{sum}(\mathsf{s}(x_3),x_4::x_5)\}, \\ +(x_6,\mathsf{sum}(x_3,\mathsf{from}(\mathsf{s}(x_6))))|\{\mathsf{sum}(\mathsf{s}(x_3),\mathsf{from}(x_6))\} \end{array} \right\} \\
\rightsquigarrow_{S\,\mathit{Decompose}} & \left\{ \begin{array}{l} +(x_4,\mathsf{sum}(x_3,x_5))|\{\mathsf{sum}(\mathsf{s}(x_3),x_4::x_5)\}, \\ +(x_6,\mathsf{sum}(x_3,\mathsf{from}(\mathsf{s}(x_6))))|\{\mathsf{sum}(\mathsf{s}(x_3),\mathsf{from}(x_6))\} \end{array} \right\} \\
\overset{*}{\rightsquigarrow}_{S\,\mathit{Delete}} & \{\quad\} \\
 & \\
 & \{\ +(x_1,x_2)|\{+(x_1,x_2)\}\ \} \\
\rightsquigarrow_{S\,\mathit{Expand}} & \{\ +(\mathsf{0},x_2)|\{+(\mathsf{0},x_2)\},\ +(\mathsf{s}(x_3),x_2)|\{+(\mathsf{s}(x_3),x_2)\}\ \} \\
\overset{*}{\rightsquigarrow}_{S\,\mathit{Simplify}} & \{\ x_2|\{+(\mathsf{0},x_2)\},\ \mathsf{s}(+(x_3,x_2))|\{+(\mathsf{s}(x_3),x_2)\}\ \} \\
\rightsquigarrow_{S\,\mathit{Decompose}} & \{\ x_2|\{+(\mathsf{0},x_2)\},\ +(x_3,x_2)|\{+(\mathsf{s}(x_3),x_2)\}\ \} \\
\overset{*}{\rightsquigarrow}_{S\,\mathit{Delete}} & \{\quad\}
\end{array}
$$

Figure 5 Derivations for proving local sufficient completeness of $\mathcal{R}_5$ in Example 28.

Now we apply the theorem to the TRS of Example 28.

▶ **Example 35.** Consider the TRS $\mathcal{R}_5$ of Example 28. We show that $\mathcal{R}_5$ is locally sufficiently complete on T_0, using Theorem 34. It is easily seen that the signature of $\mathcal{R}_5$ satisfies the conditions S1–S4 of Definition 26. Let $>_{lpo}$ be the lexicographic path order induced by the precedence $>$ such that $\mathsf{sum} > + > \mathsf{from} > \mathsf{0}, \mathsf{s}, ::, [\,]$. In Figure 5, we give derivations of $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow}_S \{\,\}$ for $h \in \mathcal{F}_0 \cap \mathcal{D}$. Also, using *Decompose* and *Delete*, we have derivations $\{h(\vec{x})|\{h(\vec{x})\}\} \overset{*}{\rightsquigarrow}_S \{\,\}$ for $h \in \mathcal{F}_0 \cap \mathcal{C}$. Thus, by Theorem 34, we conclude that $\mathcal{R}_5$ is locally sufficiently complete on T_0. ◀

6 Conclusion

We have presented simple derivation systems for proving sufficient completeness and two types of local sufficient completeness. We have given transparent correctness proofs of the derivation systems by introducing some suitable notions like well-founded induction schema. This is in contrast to the approach and correctness proofs of [3, 4] which are involved. The transparency allows us to provide derivation systems that deal with global and (two types of) local sufficient completeness in a uniform manner. Our proof methods using the derivation systems have been illustrated by applying them to some non-terminating TRSs.

The methods of [10] and our new methods are orthogonal in the following sense. In our methods, a group of function symbols are simultaneously tested, while an individual term pattern is tested in [10]. Then local sufficient completeness with signature restriction discussed in Section 4 cannot be inspected by the methods of [10], since one cannot substitute for a variable each term on a restricted signature (one can only substitute each term of the same sort as the variable; for problems in Section 5, it might be possible that the proof abilities of the two approaches are equivalent). On the other hand, any linear term pattern that is not necessarily of the form $f(x_1, \ldots, x_n)$ can be tested in [10]. This is not possible by the methods in the present paper.

The methods of [15] and our methods are also orthogonal. Example 28 of the present paper (i.e. Example 8 of [15]) is one of the examples that cannot be handled by the methods of [15]. On the other hand, the running example of [15] is difficult to handle by the current derivation systems (in [10] and the present paper), since they cannot help inducing a failing derivation with a divergent sequence as seen in Figure 4 of [15].

Găină et al. [5] have recently discussed a kind of local sufficient completeness with sort partition, where the set $\mathcal{S}_0$ in our terminology (Definition 26) contains every sort that is the codomain of some constructor. Earlier works [8, 13] discussed sufficient completeness relative to a set of constructors, where non-terminating systems are transformed into terminating ones using replacement restrictions of context-sensitive rewriting [12]. Detailed comparisons between these approaches and ours are left as future work.

In the present paper, we employed lexicographic path orders to define the derivation systems, but other simplification orders can be used if necessary. More generally, there would be a way to give abstract conditions on orders and generate constraints for derivations to be successful.

As a direction of further work, it is interesting to integrate the methods proposed in this paper and those in the previous work [10, 15] into a unified framework for proving various kinds of local sufficient completeness. Implementation of the methods and experiments to examine to what extent they work are also left as future work.

References

1 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

2 H. Comon and F. Jacquemard. Ground reducibility is EXPTIME-complete. *Inf. Comput.*, 187(1):123–153, 2003. doi:10.1016/S0890-5401(03)00134-2.

3 I. Gnaedig and H. Kirchner. Computing constructor forms with non terminating rewrite programs. In *Proceedings of the 8th PPDP*, pages 121–132. ACM, 2006. doi:10.1145/1140335.1140351.

4 I. Gnaedig and H. Kirchner. Proving weak properties of rewriting. *Theor. Comput. Sci.*, 412(34):4405–4438, 2011. doi:10.1016/j.tcs.2011.04.028.

5 D. Găină, M. Nakamura, K. Ogata, and K. Futatsugi. Stability of termination and sufficient-completeness under pushouts via amalgamation. *Theor. Comput. Sci.*, 848:82–105, 2020. doi:10.1016/j.tcs.2020.09.024.

6 J. V. Guttag. *The Specification and Application to Programming of Abstract Data Types.* PhD thesis, University of Toronto, 1975.

7 J. V. Guttag and J. J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10(1):27–52, 1978. doi:10.1007/BF00260922.

8 J. Hendrix and J. Meseguer. On the completeness of context-sensitive order-sorted specifications. In *Proceedings of the 18th RTA*, volume 4533 of *Lecture Notes in Computer Science*, pages 229–245. Springer, 2007. doi:10.1007/978-3-540-73449-9_18.

9 D. Kapur, P. Narendran, and H. Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987. doi:10.1007/BF00292110.

10 K. Kikuchi, T. Aoto, and I. Sasano. Inductive theorem proving in non-terminating rewriting systems and its application to program transformation. In *Proceedings of the 21st PPDP*, pages 13:1–13:14. ACM, 2019. doi:10.1145/3354166.3354178.

11 A. Lazrek, P. Lescanne, and J. J. Thiel. Tools for proving inductive equalities, relative completeness, and ω-completeness. *Inf. Comput.*, 84(1):47–70, 1990. doi:10.1016/0890-5401(90)90033-E.

12 S. Lucas. Context-sensitive computations in functional and functional logic programs. *J. Funct. Log. Program.*, 1998(1), 1998. URL: http://danae.uni-muenster.de/lehre/kuchen/JFLP/articles/1998/A98-01/A98-01.html.

13 S. Lucas. Completeness of context-sensitive rewriting. *Inf. Process. Lett.*, 115(2):87–92, 2015. doi:10.1016/j.ipl.2014.07.004.

14 E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002.

15 T. Shiraishi, K. Kikuchi, and T. Aoto. A proof method for local sufficient completeness of term rewriting systems. In *Proceedings of the 18th ICTAC*, volume 12819 of *Lecture Notes in Computer Science*, pages 386–404. Springer, 2021. doi:10.1007/978-3-030-85315-0_22.

16 Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

17 Y. Toyama. How to prove equivalence of term rewriting systems without induction. *Theor. Comput. Sci.*, 90(2):369–390, 1991.

A Proof Ability of the Method in Section 3

In this section, we compare the proof abilities of the method in [3, 4] and our method by the derivation system in Section 3.

The next example is a modification of a TRS described in [3, page 125 (the right column)]. ([3] discusses TRSs with possibly non-free constructors whereas we discuss TRSs with free constructors in this paper.)

▶ **Example 36.** Consider a signature with $\mathcal{S} = \{A\}$ and

$$\mathcal{F} = \{ \ \mathsf{f} : A \to A, \quad \mathsf{a} : A, \quad \mathsf{b} : A, \quad \mathsf{c} : A \ \}$$

where $\mathcal{C} = \{\mathsf{c} : A\}$. Let $\mathcal{R}_6$ be the following TRS:

$$\mathcal{R}_6 = \left\{ \begin{array}{lllll} (1) & \mathsf{a} & \to & \mathsf{b} \\ (2) & \mathsf{f}(\mathsf{f}(\mathsf{b})) & \to & \mathsf{c} \\ (3) & \mathsf{b} & \to & \mathsf{f}(\mathsf{b}) \\ (4) & \mathsf{f}(\mathsf{c}) & \to & \mathsf{c} \end{array} \right\}.$$

Then, using Theorem 12, we can show that $\mathcal{R}_6$ is sufficiently complete. Required derivations are, e.g., $\{\mathsf{b}|\{\mathsf{b}\}\} \rightsquigarrow_{\mathit{Simplify}} \{\mathsf{f}(\mathsf{b})|\{\mathsf{b}\}\} \rightsquigarrow_{\mathit{Simplify}} \{\mathsf{f}(\mathsf{f}(\mathsf{b}))|\{\mathsf{b}\}\} \rightsquigarrow_{\mathit{Simplify}} \{\mathsf{c}|\{\mathsf{b}\}\} \rightsquigarrow_{\mathit{Decompose}} \{\,\}$. However, as remarked in [3, page 125 (the right column)], these *Simplify* steps are not represented by the abstract-narrow-based process of [3, 4]. ◀

Parikh Images of Register Automata

Sławomir Lasota
University of Warsaw, Poland

Mohnish Pattathurajan
University of Warsaw, Poland

Abstract

As it has been recently shown, Parikh images of languages of nondeterministic one-register automata are rational (but not semilinear in general), but it is still open if the property extends to all register automata. We identify a subclass of nondeterministic register automata, called *hierarchical register automata* (HRA), with the following two properties: every rational language is recognised by a HRA; and Parikh image of the language of every HRA is rational. In consequence, these two properties make HRA an automata-theoretic characterisation of languages of nondeterministic register automata with rational Parikh images.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Sets with atoms, register automata, Parikh images, rational sets, hierarchical register automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.50

Funding This work was partially supported by NCN grant 2019/35/B/ST6/02322 and by the ERC grant INFSYS, agreement no. 950398.

Acknowledgements We are grateful to Piotrek Hofman and Arka Ghosh for fruitful discussions.

1 Introduction

Register automata, also know as finite-memory automata, introduced over 25 years ago by Francez and Kaminski [14], are nondeterministic finite-state devices recognising languages over infinite alphabets. They are equipped with a finite number of registers that can store data values (atoms) from an infinite data domain. A register automaton inputs a string of data values (a data word) and compares each consecutive input to its registers; based on this comparison and on the current control state, it chooses a next control state and possibly stores the input value in one of its registers. The only allowed comparisons of data values considered in this paper are equality and inequaltiy tests. An automaton can also guess a fresh data value different from the ones seen currently in the input or stored in registers, and store it in some register (we thus consider nondeterministic register automata *with guessing* [24]). Likewise one may define register context-free grammars [6], [1, Sect.5].

Register automata lack most of the good properties of finite automata, like determinisation or closure properties. In particular, no satisfactory characterisation in terms of rational (regular) expressions is known. Indeed, all known generalisations of Kleene's theorem for register automata either apply to a restricted subclass of the model [17], or introduce an involved syntax significantly extending the concept of rational expressions [19, 18], or rely on a richer algebraic structure than the free monoid of data words [3].

Register automata are expressively equivalent to *orbit-finite automata* [5, 6], a natural extension of finite automata where input alphabets and state spaces are possibly infinite, but finite up to permutation of the data domain (such sets are called *orbit-finite*). Along these lines, we focus on a natural extension of rational expressions, which differ from the classical ones just by allowing for *orbit-finite unions* instead of only finite ones. In other words, we

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 50; pp. 50:1–50:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

consider the class of *rational languages*, defined as the smallest class of languages containing all single-word languages, and closed under concatenation, star, and orbit-finite unions. In particular, the class contains the empty language, all finite and all orbit-finite languages.

Languages of register automata are not rational in general, even in case of deterministic one-register automata. Kleene theorem may be however recovered, at least in case of automata with one register, when commutative images (Parikh images) are considered: the language of every one-register automaton is Parikh-equivalent to (i.e., has the same Parikh image as) a rational language [16]. An analogous result holds for one-register context-free grammars [16].

▶ **Example 1.** Fix the data domain $\mathrm{ATOMS} = \{0, 1, 2, \ldots\}$. As a working example we will use the language L_1 consisting of all nonempty words over ATOMS of length divisible by 3, where every three consecutive letters are pairwise different (we write $\neq(a, b, c)$ as a shorthand for $a \neq b \neq c \neq a$, to concisely express pairwise inequality of three atoms):

$$L_1 \;=\; \{a_1 a_2 \ldots a_{3n} \in \mathrm{ATOMS}^* \,:\, n \geq 0,\; \neq(a_1, a_2, a_3),\; \neq(a_2, a_3, a_4),\; \neq(a_3, a_4, a_5),\; \ldots\}.$$

The language is recognised by a deterministic two-register automaton but it is not rational (cf. Section 3). It is however Parikh-equivalent to a larger language L_2, where the pairwise inequality constraint is imposed at consecutive disjoint triples of positions only:

$$L_2 \;=\; \{a_1 a_2 \ldots a_{3n} \in \mathrm{ATOMS}^* \,:\, n \geq 0,\; \neq(a_1, a_2, a_3),\; \neq(a_4, a_5, a_6),\; \ldots\},$$

which is defined by the following rational (regular) expression

$$L_2 \;=\; \Big(\bigcup_{a,b,c \in \mathrm{ATOMS},\; \neq(a,b,c)} abc \Big)^* \tag{1}$$

and is thus rational. The formal definition of rational languages will be given in Section 3; here we note that the union is indexed by the set $\mathrm{ATOMS}^{(3)} = \{\langle abc \rangle \in \mathrm{ATOMS}^3 \,:\, \neq(a, b, c)\}$ of all triples of pairwise-distinct atoms, which is infinite but orbit-finite, i.e., finite up to permutations of ATOMS (in fact, it is one orbit).

The language L_1 is Parikh-equivalent to L_2 as every $w = a_1 a_2 \ldots a_{3n} \in L_2$ can be transformed, by swapping letters, to a word in $w' \in L_1$.

Indeed, consider the first two triples (a_1, a_2, a_3) and (a_4, a_5, a_6) in w. We keep the first triple in w'. For the fourth position of w', we choose a letter from $\{a_4, a_5, a_6\} - \{a_2, a_3\}$, say a_6. For the fifth position we choose $\{a_4, a_5\} - \{a_3\}$, say a_4. We note that both the choices are possible due to pigeon-hole principle. Finally, at the sixth position of w' we place the remaining letter a_5. Then we consider next two triples, (a_5, a_4, a_6) and (a_7, a_8, a_9), and treat them analogously by swapping a_7, a_8 and a_9 accordingly. Continuing in this way we finally arrive at a word in $w' \in L_1$. ⌟

Contribution. We contribute to understanding of expressive power of nondeterministic register automata (NRA), by investigating sets of *data vectors* obtainable as commutative images (Parikh images) of their languages. Parikh images of rational languages we call rational as well. Here are our contributions:

(1) We identify a syntactic subclass of NRA, called *hierarchical register automata* (HRA).
(2) We show that every rational language is recognised by a HRA.
(3) We show that Parikh images of HRA languages are rational (as a set of data vectors).
(4) As a corollary, we deduce that an NRA has rational Parikh image if, and only if it is Parikh-equivalent to some HRA (with, possibly, more registers).

These results are a step towards the ultimate (but still unreachable) goal: generalise the main result of [16], namely rationality of Parikh images of nondeterministic 1-register automata, to all NRA. Point (3) is an extension from 1-NRA to all HRA. In consequence of (4), the ultimate goal can be equivalently achieved by proving that every nondeterministic register automaton is Parikh-equivalent to a hierarchical one. Finally, we believe that the subclass of HRA (1) is interesting on its own, as it seems to be equally well-behaved as one-register automata.

Related research. Register automata have been intensively studied with respect to their foundational properties [14, 23, 17, 21]. Following the seminal paper of Francez and Kaminski [14], subsequent extensions of the model allow for comparing data values with respect to some fixed relations such as a total order, or introduce alternation, variations on the allowed form of nondeterminism, etc. The model is well known to satisfy almost no semantic equivalences that hold for classical finite automata, in particular register automata admit no satisfactory characterizations in terms of regular expressions [19, 18] or logic [21, 10]. There just are few positive results: simulation of two-way nondeterministic automata by one-way alternating automata with guessing [1]; Myhill-Nerode-style characterisation of languages of deterministic automata [15, 5, 6]; and the well-behaved class of languages definable by orbit-finite monoids [2], characterised in terms of logic [9] and a syntactic subclass of deterministic register automata [8]. Register automata have been also intensively studied with respect to their applications to XML databases and logics [12, 21, 10, 24].

Other extensions of finite-state machines to infinite alphabets include: abstract reformulation or register automata, known as orbit-finite automata, or nominal automata, or automata over atoms) [5, 6, 1]; symbolic automata [11]; pebble automata [20]; and data automata [4, 7] (the list is illustrative).

2 Orbit-finite sets

Sets with atoms. Our definitions rely on basic notions and results of the theory of *sets with atoms* [1], also known as nominal sets [22]. In this section we recall, following [16], what is necessary for understanding of our arguments. This paper is a part of a uniform abstract approach to register automata in the realm of orbit-finite sets with atoms, developed in [5, 6, 1].

Fix a countably infinite set $\textsc{Atoms}$, whose elements we call *atoms*. Informally speaking, a set with atoms is a set that can have atoms, or other sets with atoms, as elements. Formally, we define the universe of sets with atoms by a suitably adapted cumulative hierarchy of sets, by transfinite induction: the only set of *rank* 0 is the empty set; and for a cardinal γ, a set of rank γ may contain, as elements, sets of rank smaller than γ as well as atoms. In particular, nonempty subsets $X \subseteq \textsc{Atoms}$ have rank 1. Sets containing no atoms, whose elements contain no atoms, and so on, we call *pure* (or *atomless*).

Denote by $\textsc{Perm}$ the group of all permutations of $\textsc{Atoms}$. Atom permutations $\pi : \textsc{Atoms} \to \textsc{Atoms}$ act on sets with atoms by consistently renaming all atoms in a given set. Formally, by another transfinite induction we define $\pi(X) = \{\pi(x) : x \in X\}$. Via standard set-theoretic encodings of pairs or finite sequences we obtain, in particular, the pointwise action on pairs $\pi(x, y) = (\pi(x), \pi(y))$, and likewise on finite sequences. For pure sets X, $\pi(X) = X$ for every $\pi \in \textsc{Perm}$.

We restrict to sets with atoms that only depend on finitely many atoms, in the following sense. A *support* of x is any set $S \subseteq \textsc{Atoms}$ such that the following implication holds for all $\pi \in \textsc{Perm}$: if $\pi(s) = s$ for all $s \in S$, then $\pi(x) = x$. An element (or set) x is *finitely* *supported* if it has some finite support; in this case x has *the least support*, denoted $\textsc{supp}(x)$, called *the support* of x (cf. [1, Sect. 6]), [22, Prop. 2.3], [6, Cor. 9.4]). Sets supported by $\emptyset$ we call *equivariant*. For instance, given $a, b \in \textsc{Atoms}$, the support of the set

$$L_{ab} \;=\; \{a_1 a_2 \ldots a_n \in \textsc{Atoms}^* \;:\; n \geq 2,\; a_1 \neq a,\; a_n = b\}$$

is $\{a, b\}$; every pure set is equivariant; the support of a sequence $\langle a_1 \ldots a_n \rangle \in \textsc{Atoms}^*$, encoded as a set in a standard way, is the set of atoms $\{a_1, \ldots, a_n\}$ appearing in it; and the support of a function $f : \textsc{Atoms} \to \mathbb{N}$ such that $\textsc{dom}(f) = \{a \in \textsc{Atoms} \;:\; f(a) > 0\}$ is finite, is exactly $\textsc{dom}(f)$.

From now on, we shall only consider sets with atoms that are hereditarily finitely supported (called briefly *legal*), i.e., ones that are finitely supported, whose every element is finitely supported, and so on.

Orbit-finite sets. Two (elements of) sets with atoms x, y are *in the same orbit* if $\pi(x) = y$ for some $\pi \in \textsc{Perm}$. This equivalence relation splits every set with atoms X into equivalence classes, which we call *orbits in* X. A (legal) set is *orbit-finite* if it splits into finitely many orbits. Examples of orbit-finite sets are: $\textsc{Atoms}$ (1 orbit); $\textsc{Atoms} - \{a\}$ for some $a \in \textsc{Atoms}$ (1 orbit); $\textsc{Atoms}^2$ (2 orbits: diagonal $\{\langle a, b\rangle \;:\; a = b\}$ and non-diagonal $\{\langle a, b\rangle \;:\; a \neq b\}$); $\textsc{Atoms}^3$ (5 orbits, corresponding to equality types of triples); $\{1, \ldots, n\} \times \textsc{Atoms}$ (n orbits, as $\pi(i) = i$ for every $i \in \mathbb{N}$ and $\pi \in \textsc{Perm}$, since $\{1, \ldots, n\}$ is pure); the set of non-repeating n-tuples of atoms $\textsc{Atoms}^{(n)} = \{a_1 \ldots a_n \in \textsc{Atoms}^n \;:\; a_i \neq a_j \text{ for every } 1 \leq i < j \leq n\}$ (1 orbit). On the other hand, the set $\textsc{Atoms}^*$ is an example of an orbit-infinite set.

A finer equivalence relation is defined using *S-atom permutations*, i.e., permutations that fix a finite set $S \subseteq \textsc{Atoms}$. Each orbit splits into finitely many *S-orbits* (cf. [1, Sect. 3.2]). For instance, for every $a \in \textsc{Atoms}$, the set $\textsc{Atoms}^2$ splits into four $\{a\}$-orbits: $\{\langle a, a\rangle\}$, $\{\langle a, b\rangle \;:\; b \neq a\}$, $\{\langle b, a\rangle \;:\; b \neq a\}$, $\{\langle b, c\rangle \;:\; b, c \neq a\}$.

Given a family $(X_i)_{i \in I}$ of sets indexed by an orbit-finite set I, the union $\bigcup_{i \in I} X_i$ we call *orbit-finite union* of sets X_i. (Formally, not only each set X_i is assumed to be legal, but also the indexing function $i \mapsto X_i$.) As an example, consider $(L_{ab})_{b \in \textsc{Atoms}}$. The indexing function $b \mapsto L_{ab}$ is supported by $\{a\}$, and so is the union:

$$\bigcup_{b \in \textsc{Atoms}} L_{ab} \;=\; \{a_1 a_2 \ldots a_n \in \textsc{Atoms}^* \;:\; n \geq 2,\; a_1 \neq a\}.$$

Orbit-finite sets are closed under Cartesian products, subsets, and orbit-finite unions: if each of X_i is orbit-finite, their union $\bigcup_{i \in I} X_i$ is orbit-finite too [1, Sect. 3].

3 Rational sets

In this section we recall the definition of rational sets of data words and data vectors introduced in [16], and state and prove its useful closure properties.

Data words and vectors. By a finite multiset over a set (an alphabet) Σ we mean any function $v : \Sigma \to \mathbb{N}$ such that $v(\alpha) = 0$ for all $\alpha \in \Sigma$ except finitely many. We define the *domain* of v as $\textsc{dom}(v) = \{\alpha \in \Sigma \;:\; v(\alpha) > 0\}$, and its *size* as $|v| = \sum_{\alpha \in \textsc{dom}(v)} v(\alpha)$ (the same notation is used for the size of a set, and for the length of a word). The *Parikh image* (commutative image) of a word $w \in \Sigma^*$ is the multiset $\textsc{Par}(w) : \Sigma \to \mathbb{N}$, where $\textsc{Par}(w)(\alpha)$ is the number of appearances of a letter $\alpha \in \Sigma$ in w. For a language $L \subseteq \Sigma^*$, its Parikh image is $\textsc{Par}(L) = \{\textsc{Par}(w) \;:\; w \in L\}$. Two languages $L, L' \subseteq \Sigma^*$ are *Parikh-equivalent* if they have the same Parikh images: $\textsc{Par}(L) = \textsc{Par}(L')$. Overloading the notation, we write $|w|$ for the *length* of a word w, and hence $|\textsc{Par}(w)| = |w|$. We order multisets pointwise: $v \sqsubseteq v'$ if $v(\alpha) \leq v'(\alpha)$ for all $\alpha \in \Sigma$. The zero (empty) multiset $\mathbf{0}$ satisfies $\mathbf{0}(\alpha) = 0$ for every $\alpha \in \Sigma$. Thus $\mathbf{0} = \textsc{Par}(\varepsilon)$. A singleton $\{\alpha\}$ that maps α to 1 and all other letters to 0, is written as α, omitting brackets $\{\}$. Addition of multisets is pointwise: $(v + v')(\alpha) = v(\alpha) + v'(\alpha)$ for every $\alpha \in \Sigma$; likewise subtraction $v - v'$, for $v' \sqsubseteq v$.

When Σ is an orbit-finite alphabet, words $w \in \Sigma^*$ are traditionally called *data words*, languages $L \subseteq \Sigma^*$ are called *data languages*, and finite multisets $v : \Sigma \to \mathbb{N}$ are called *data vectors*.

Orbit-finite unions. Consider a family of sets $\mathcal{X}$. We say that $\mathcal{X}$ is *closed under orbit-finite unions* if for every family $(X_i)_{i\in I}$ of sets $X_i \in \mathcal{X}$ indexed by an orbit-finite set I, the union $\bigcup_{i\in I} X_i$ belongs to $\mathcal{X}$. We instantiate below this abstract definition to families $\mathcal{X}$ of sets of data words and data vectors.

Rational data languages. We consider data languages over a fixed orbit-finite alphabet Σ. As usual, we define concatenation of two data languages $LL' = \{ww' : w \in L, w' \in L'\}$, and the Kleene star (iteration): $L^* = \{w_1 \ldots w_n : n \geq 0, w_1, \ldots, w_n \in L\}$. Let *rational data languages* be the smallest class of data languages that contains that contains $\{\varepsilon\}$, all singleton languages $\{\sigma\}$ containing a single one-letter word $\sigma \in \Sigma$, and is closed under concatenation, iteration, and orbit-finite unions. In particular the empty language, all finite languages and all orbit-finite ones are rational. For finite Σ we obtain the classical rational (regular) sets. As expected, without the Kleene star we obtain exactly sets of words of bounded length, or equivalently (cf. [16, Lemma 1]) orbit-finite languages.

When convenient, we may speak of a *rational expression*, by which we mean a formal derivation of a rational language according to the closure rules listed above, in the form of well-founded tree. Concretely, a derivation of $\bigcup_{i\in I} L_i$ is the function mapping every $i \in I$ to a derivation of L_i (a node in a tree whose children are labeled by I), a derivation of LL' is a pair of derivations of L and L' (a binary node), a derivation of L^* is just a derivation of L (a unary node), and a derivation of $\{\varepsilon\}$ or $\{\sigma\}$ is a leaf node.

▶ **Example 2.** Continuing Example 1, the language L_2 is rational, as it can be presented by a rational expression:

$$L_2 = \Big(\bigcup_{a,b,c \in \textsc{Atoms},\ \neq(a,b,c)} \{a\}\{b\}\{c\} \Big)^*.$$

For readability, in the sequel we omit brackets $\{\}$ when denoting singletons, as in (1). On the other hand, one easily shows that the language L_1 is not rational (e.g., using Proposition 12 from Section 4 and Theorem 13 from Section 5).

Rational sets of data vectors. We consider sets of data vectors over a fixed orbit-finite alphabet Σ. Let addition of two sets X, Y of data vectors be defined by Minkowski sum

$$X + Y = \{x + y : x \in X, y \in Y\},$$

and let the additive star X^* contain all finite sums of elements of X:

$$X^* = \{x_1 + \ldots + x_n : n \geq 0, x_1, \ldots, x_n \in X\}.$$

We define *rational sets* of data vectors as the smallest class of sets of data vectors that contains $\{\mathbf{0}\}$, all singletons $\{\sigma\}$ where σ stands for the 'unit' data vector over Σ that maps σ to 1 and all other letters to 0, and is closed under addition, additive star, and orbit-finite unions. In particular, the empty set, all finite sets and all orbit-finite sets of data vectors are rational.

▶ **Example 3.** Continuing Example 2, the Parikh image of L_1 (and L_2) is rational (for readability we keep omitting brackets $\{\}$):

$$\textsc{Par}(L_1) \ = \ \Big(\bigcup_{a,b,c \in \textsc{Atoms}, \ \neq(a,b,c)} a+b+c \Big)^*.$$

▷ Claim 4. (1) Rational sets of data vectors are exactly Parikh images of rational data languages. (2) $\textsc{Par}(L)$ is rational if, and only if, L is Parikh-equivalent to a rational data language.

▶ Remark 5. The classical notion of rational sets in an arbitrary monoid ([13, Chapter VII]) can be generalised along the same lines as above to sets with atoms, by considering orbit-finite unions instead of finite ones. In this paper we stick to monoids of data words and data vectors, over an orbit-finite alphabet.

Closure properties. As tools to be used later, we prove that rationality of a language is preserved by the restriction to a subset of its alphabet, as well as by substitution by rational languages. The same preservation property holds for languages with rational Parikh images.

▶ **Lemma 6.** *If a language $L \subseteq \Sigma^*$ has rational Parikh image (resp. is rational) and $\Gamma \subseteq \Sigma$ then the restriction $L \cap \Gamma^*$ has also rational Parikh image (resp. is rational).*

Proof. Intuitively speaking, it is enough to syntactically remove, in the rational expression defining $\textsc{Par}(L)$, every appearance of a letter $\sigma \in \Sigma - \Gamma$.

Formally, we proceed by induction on a derivation of L. By Claim 4(2) we assume, w.l.o.g., that the language L is rational.

The induction base: when $L = \{\sigma\}$ is a singleton, $\sigma \in \Sigma$, then

$$L \cap \Gamma^* \ = \ \begin{cases} L & \text{if } \sigma \in \Gamma, \\ \emptyset & \text{otherwise,} \end{cases}$$

and in each case $L \cap \Gamma^*$ is rational. The induction step follows immediately as restriction commutes with all the operations involved:

$$(LK)\cap\Gamma^* \ = \ (L\cap\Gamma^*)(K\cap\Gamma^*) \qquad L^*\cap\Gamma^* \ = \ (L\cap\Gamma^*)^* \qquad \big(\bigcup_{i\in I} L_i\big)\cap\Gamma^* \ = \ \big(\bigcup_{i\in I} L_i\cap\Gamma^*\big). ◀$$

Consider a language L over an orbit-finite alphabet Σ and a (legal) family of languages $K = (K_\sigma)_{\sigma\in\Sigma}$ over an alphabet Γ, indexed by Σ. We use the anonymous function notation

$$\sigma \quad \mapsto \quad K_\sigma.$$

The *substitution* $L(K)$ is the language over Γ containing all words obtained from some word $\sigma_1\sigma_2\ldots\sigma_n \in L$, by replacing every letter σ_i by some word from K_{σ_i}:

$$L(K) \ = \ \bigcup_{\sigma_1\sigma_2\ldots\sigma_n \in L} K_{\sigma_1}K_{\sigma_2}\ldots K_{\sigma_n}.$$

▶ **Example 7.** As usual, let $L^+ = L^*L$. Consider the language L_1 from Example 1 and $\Sigma = \Gamma = \textsc{Atoms}$. By the equivariant substitution $K_a = a^+$, or $a \mapsto a^+$, we obtain the language $L_1(K) \subseteq \textsc{Atoms}^*$ containing words, where each three consecutive maximal constant infixes use three distinct letters (each two consecutive maximal constant infixes use two distinct letters by the very definition), and the total number of these infixes is divisible by 3.

▶ **Lemma 8** ([16], Lemma 5). *If L and all languages K_σ have rational Parikh images (resp. are rational) then the substitution $L(K)$ has also rational Parikh image (resp. is rational).*

4 Register automata

We define the model of nondeterministic register automata, and its syntactic subclass of hierarchical automata.

Nondeterministic register automata (NRA). From now on we mostly consider input alphabets of the form $\Sigma = H \times \textsc{Atoms}$, where H is a finite pure (atomless) set.

Let $k \geq 1$. In the sequel we consistently use variables x_i, x'_i, for $1 \leq i \leq k$, to represent the value of ith register at the start (pre-value) and at the end (post-value) of a transition, respectively. We also consistently use the variable y to represent an input atom. A *nondeterministic k-register automaton* (k-NRA) $\mathcal{A}$ consists of: a finite set H (finite component of the alphabet), a finite set of control locations Q, subsets $I, F \subseteq Q$ of initial resp. accepting locations, and a finite set Δ of transition rules of the form

$$(q(x_1, x_2 \ldots x_k), \langle h, y\rangle, \varphi, q'(x'_1, x'_2 \ldots x'_k)) \tag{2}$$

where $q, q' \in Q$, $h \in H$, and the transition constraint $\varphi(x_1, x_2 \ldots x_k, y, x'_1, x'_2 \ldots x'_k)$ is a Boolean combination of equalities involving the variables $x_1, x_2 \ldots x_k, y, x'_1, x'_2 \ldots x'_k$. The constraint specifies possible relation between the register pre-values $(x_1, x_2 \ldots x_k)$, input atom (y), and register post-values $(x'_1, x'_2 \ldots x'_k)$ resulting from a transition. If φ entails the equality $x_i = x'_i$, we say that the ith register is *preserved* by the transition rule.

A configuration $\langle q, (a_1 a_2 \ldots a_k)\rangle \in Q \times \textsc{Atoms}^{(k)}$ of $\mathcal{A}$, written briefly $q(a_1 a_2 \ldots a_k)$, consists of a control location $q \in Q$ and (pairwise distinct[1]) register values $a_i \in \textsc{Atoms}$, for $1 \leq i \leq k$. We note that different registers can not store the same value. For each tuple $\mathbf{r} = a_1 a_2 \ldots a_k \in \textsc{Atoms}^{(k)}$, atom $b \in \textsc{Atoms}$, and tuple $\mathbf{r}' = a'_1 a'_2 \ldots a'_k \in \textsc{Atoms}^{(k)}$ that satisfy the transition constraint, i.e., $(a_1 a_2 \ldots a_k, b, a'_1 a'_2 \ldots a'_k) \models \varphi$, a rule (2) induces a transition

$$q(a_1 a_2 \ldots a_k) \xrightarrow{\langle h,b\rangle} q'(a'_1 a'_2 \ldots a'_k)$$

labeled by $\langle h, b\rangle$ from the configuration $q(a_1 a_2 \ldots a_k)$ to the configuration $q'(a'_1 a'_2 \ldots a'_k)$. The semantics of k-NRA is defined as in case of classical NFA, with configurations considered as states and $\Sigma = H \times \textsc{Atoms}$ as an alphabet. A *run* of $\mathcal{A}$ over a data word $w = \langle h_1, b_1\rangle\langle h_2, b_2\rangle \ldots \langle h_n, b_n\rangle \in \Sigma^*$ is any sequence of configurations $q_0(\mathbf{r}_0), q_1(\mathbf{r}_1), \ldots, q_n(\mathbf{r}_n)$, related by transitions labeled by consecutive letters of w:

$$q_0(\mathbf{r}_0) \xrightarrow{\langle h_1,b_1\rangle} q_1(\mathbf{r}_1) \xrightarrow{\langle h_2,b_2\rangle} \ldots \xrightarrow{\langle h_n,b_n\rangle} q_n(\mathbf{r}_n), \tag{3}$$

[1] Distinctness of register values is not relevant for expressiveness of register automata.

where $q_0(\mathbf{r}_0)$ is an initial configuration (i.e., $q_0 \in I$). A run is *accepting* if the ending configuration $q_n(\mathbf{r}_n)$ is accepting (i.e., $q_n \in F$). A data word w is *accepted* by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run over w.

Let $L_{q(\mathbf{r})\, q'(\mathbf{r}')}(\mathcal{A})$ be the set of data words having an accepting run (3) that starts in $q_0(\mathbf{r}_0) = q(\mathbf{r})$ and ends in $q_n(\mathbf{r}_n) = q'(\mathbf{r}')$. The *language* $L(\mathcal{A})$ recognised by $\mathcal{A}$ is defined as:

$$L(\mathcal{A}) \;=\; \bigcup_{q \in I, q' \in F, \mathbf{r}, \mathbf{r}' \in \textsc{Atoms}^{(k)}} L_{q(\mathbf{r})\, q'(\mathbf{r}')}(\mathcal{A}). \tag{4}$$

▶ Remark 9. The above definition allows for *guessing*, i.e., an automaton may nondeterministically choose, and store in its register, an atom not yet seen in the input (cf. [24]). In particular, the initial register values are guessed nondeterministically.

▶ Remark 10. An alphabet $H \times \textsc{Atoms}$ and configurations $Q \times \textsc{Atoms}^{(k)}$ are orbit-finite. The model of NRA is a special case of the abstract notion of orbit-finite automata (cf. [1, Sect. 5.2]), where alphabets and state spaces may be arbitrary orbit-finite sets. For alphabet of the form $\Sigma = H \times \textsc{Atoms}$, where H is pure and finite, NRA are expressively equivalent to orbit-finite automata [1, Sect. 5.2].

Hierarchical register automata (HRA). We define a syntactical subclass of NRA by restricting transition constraints. The idea is to update registers in a hierarchical manner: if a transition rule does not preserve ith register, pre- and post-values of every larger register (jth register, for $j > i$) are unspecified. Formally, a HRA is a NRA where each transition constraint φ has the following form:

$$\varphi \quad \equiv \quad \psi(x_1, x_2, \ldots, x_i, y, x_i') \;\wedge \bigwedge_{1 \le j < i} x_j = x_j', \tag{5}$$

for some $i \in \{1, \ldots, k\}$. The sub-formula ψ describes how the post-value of ith register (x_i') depends on the relation between the input atom (y) and the pre-values of ith register and smaller ones ($x_1, x_2, \ldots, x_i$). Note that all smaller registers are preserved, and larger ones are not mentioned in φ (and hence their pre- and post-values are unspecified, which means that any pre- and post-values are allowed). Note also that the constraint φ allows for updating ith register (according to the sub-constraint ψ) as well as every larger register (arbitrarily); the former we call *specified* update, and the latter one we call *unspecified* one. The number i we call the *level* of the transition constraint, or of the transition (rule) it appears in. As extreme examples, the following all-registers-preserving constraint

$$\bigwedge_{1 \le j \le k} x_j = x_j' \ne y, \tag{6}$$

as well as the most liberal constraint **true** satisfied by any pre- and post-values of registers and any input atom, both are in the syntactic form (5), at level k and 1, respectively.

Intuitively speaking a HRA, when restricted to transition rules of some fixed level i, resembles a NRA with just one (ith) register, with all larger registers removed, and all smaller registers *frozen* to some fixed values. For $i \le k$ and a tuple of atoms $\mathbf{r} \in \textsc{Atoms}^{(i)}$, we may define a refined semantics of a k-HRA $\mathcal{A}$ as the language of words accepted by a run where the values of the first (smallest) i registers are continuously $\mathbf{r}$ and hence never change. We denote the so defined language by $L_{\mathbf{r}}(\mathcal{A})$.

W.l.o.g. we may assume that a HRA is *orbitized*, i.e., its every transition constraint $\varphi(x_1, \ldots, x_i, y, x_1', \ldots, x_i')$ at level i defines one orbit (one equality type) in $\textsc{Atoms}^{2i+1}$. For instance, the constraint (6) defines one orbit, while **true** does not.

▶ **Example 11.** Let H be a singleton, omitted below; we thus consider ATOMS as an alphabet. The following 2-HRA recognises the language L_2 from Example 1. The control locations are $Q = \{q_1, q_2, q_3\}$, with single initial and accepting one $I = F = \{q_3\}$. The automaton has the following three transition rules:

$$(q_3(x_1, x_2),\ y,\ x_1 = x_1' \neq y \wedge x_2 = x_2' \neq y,\ q_2(x_1', x_2')),$$
$$(q_2(x_1, x_2),\ y,\ x_1 = x_1' \wedge x_2 = x_2' = y,\ q_1(x_1', x_2')),$$
$$(q_1(x_1, x_2),\ y,\ x_1 = y,\ q_3(x_1', x_2')).$$

the first two at level 2 and the last one at level 1. The post-value x_2' of the second register is unspecified in the last two rules. Moreover, the post-value x_1' of the first register is also unspecified in the last rule, and therefore the automaton is not orbitized. It can be easily made orbitized by replacing this last rule with the following ones:

$$(q_1(x_1, x_2),\ y,\ x_1 = y = x_1',\ q_0(x_1', x_2')),$$
$$(q_1(x_1, x_2),\ y,\ x_1 = y \neq x_1',\ q_0(x_1', x_2')).$$

It is not difficult to show that in terms of expressiveness HRA are a strict subclass of NRA:

▶ **Proposition 12.** *The language L_1 from Example 1 is not recognised by any* HRA*.*

Proof. Towards contradiction, suppose L_1 is recognised by a k-HRA $\mathcal{A}$. Consider a word $w = a_1 a_2 \ldots a_{k+2} \in \text{ATOMS}^*$ of length $k+2$ in which all letters are pairwise different ($a_i \neq a_j$ for $i \neq j$) and an accepting run π of $\mathcal{A}$ over w. Let $\mathbf{r}_i$ be the valuation of registers in π after reading a_i.

We observe that each letter a_i, for $i < k+2$, must be stored in a register in the considered run π: a_i it is the value of some register in $\mathbf{r}_i$. Indeed, suppose contrarily that a_i is not the value of any register in $\mathbf{r}_i$. By replacing this letter in w with a_{i+1} we obtain a word w' where two consecutive letters are equal, and hence $w' \notin L_1$. On the other hand the run π is also an accepting run over w', and hence $w' \in L(\mathcal{A})$ – a contradiction.

Therefore we know that a_i is the value of some ℓ_ith register in $\mathbf{r}_i$, for every $i = 1, \ldots, k+1$. Note that this register with value a_i is unique, and that it gets its value either by the specified or unspecified update. We claim that $\ell_i < \ell_{i+1}$ for every $i = 1, \ldots, k$. Indeed, suppose $\ell_i \geq \ell_{i+1}$ for some i. The inequality implies that either the value a_i stored in ℓ_ith register is overwritten by the specified update (when $\ell_i = \ell_{i+1}$), or *may* be overwritten by an unspecified one (when $\ell_i > \ell_{i+1}$). By replacing a_i in w with a_{i+2} we obtain a word $w'' \notin L_1$. On the other hand the run π is easily modified into an accepting run over w'' by replacing a_i with a_{i+2} in $\mathbf{r}_i$. In consequence, $w'' \in L(\mathcal{A})$ – a contradiction, similarly as before.

We have thus an increasing sequence $1 \leq \ell_1 < \ell_2 < \ldots < \ell_{k+1} \leq k$, thus yielding a contradiction. ◀

As an intermediate corollary of Proposition 12 and Theorem 13 (cf. Section 5) we deduce that L_1 is not rational either.

5 Parikh-equivalence of HRA and rational languages

As our main contribution, we prove that Parikh images of rational languages (rational sets of data vectors) coincide with Parikh images of HRA (cf. Corollary 22). This is split into two parts: on one side we prove that rational data languages are recognised by HRA, and on the other side Parikh images of HRA languages are rational (as sets of data vectors):

▶ **Theorem 13.** *Rational data languages are recognised by* HRA.

▶ **Theorem 14.** *Parikh images of* HRA *languages are rational.*

Proof of Theorem 13. We proceed by induction on derivation of a rational language. For convenience we assume, w.l.o.g., that each orbit-finite sum is indexed by a subset of $I \subseteq \text{ATOMS}^{(n)}$ of non-repeating n-tuples of atoms, for some $n \in \mathbb{N}$. Indeed, every orbit-finite union can be split into a finite union of single-orbit unions, and every single-orbit set J is the image of an equivariant function f from such a set I (cf. [1, Sect. 3.2]), $J = f(I)$, hence

$$\bigcup_{j \in J} L_j \quad = \quad \bigcup_{i \in I} L_{f(i)} \quad = \quad \bigcup_{i \in I} K_i$$

where $K_i = L_{f(i)}$. Under this simplifying assumption we prove, by induction on derivation of a rational language, the following claim (we say that a tuple $\mathbf{s} \in \text{ATOMS}^{(n)}$ supports x if the set of n atoms appearing in $\mathbf{s}$ does so):

▷ Claim 15. For every rational language L over an alphabet of the form $\Sigma = H \times \text{ATOMS}$, and every tuple $\mathbf{s}$ supporting its derivation, there is a HRA $\mathcal{A}$ such that $L_{\mathbf{s}}(\mathcal{A}) = L$.

We emphasise that we consider supports of *derivations* of rational languages, defined as well-founded trees (cf. Section 3), instead of supports of languages themselves. Clearly, a tuple supporting a derivation of a language also support the language itself.

The induction base, for $L = \{\varepsilon\}$ or $L = \{\sigma\}$ where $\sigma \in \Sigma$, is straightforward. The induction step splits into three cases.

Case 1: $L = L_1 L_2$ Let $\mathbf{s}$ be a tuple of atoms supporting the derivation of L, and hence also the derivations of L_1 and L_2. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the HRA which, due to the induction assumption, recognize $L_{\mathbf{s}}(\mathcal{A}_1) = L_1$ and $L_{\mathbf{s}}(\mathcal{A}_2) = L_2$. Let the automaton $\mathcal{A}$ initially run $\mathcal{A}_1$, and from each accepting location of $\mathcal{A}_1$ nondeterministically choose either to continue inside $\mathcal{A}_1$, or to run $\mathcal{A}_2$. We have $L_{\mathbf{s}}(\mathcal{A}) = L$, as required.

Case 2: $L = K^*$ This case is dealt with similarly to the previous one.

Case 3: $L = \bigcup_{i \in I} L_i$ Let $\mathbf{s}$ be a tuple of atoms supporting the derivation of L, and hence also the set I and the mapping $i \mapsto L_i$. Thus the concatenated tuple $\mathbf{s}i$ supports L_i (recall that i is assumed for convenience to be a tuple of atoms). For an $\mathbf{s}$-orbit J in I, let

$$L_J \;=\; \bigcup_{j \in J} L_j \;\subseteq\; L.$$

Consider an arbitrary $\mathbf{s}$-orbit J in I (each orbit is treated separately). Fix an arbitrary element $i \in J$ and an automaton $\mathcal{B}$ such that, due to the induction assumption, recognizes $L_{\mathbf{s}i}(\mathcal{B}) = L_i$. Therefore, for every $j = \pi(i) \in J$, where π is an $\mathbf{s}$-automorphism, the same automaton $\mathcal{B}$ recognizes $L_{\mathbf{s}j}(\mathcal{B}) = L_j$. Let the automaton $\mathcal{A}_J$ initially guess $i \in J$ and put it into the smallest registers not occupied by $\mathbf{s}$, and then run $\mathcal{B}$. We have $L_{\mathbf{s}}(\mathcal{A}_J) = L_J$. The language L is the union of finitely many languages L_J, and hence L is recognized by a HRA that initially chooses an $\mathbf{s}$-orbit J in I and then runs $\mathcal{A}_J$. ◀

Proof of Theorem 14. We now focus on showing that Parikh images of languages of HRA are rational. The proof proceeds by induction on the number of registers.

Induction base. The induction base, i.e., rationality of Parikh images of 1-HRA languages, follows immediately by the following result of [16]:

▶ **Lemma 16** ([16], Theorem 6)**.** *Parikh images of* 1*-NRA languages are rational.*

Altering paths. Before proceeding to the induction step we recall an immediate corollary of another results of [16] (cf. Lemma 17 below). Given a k-HRA $\mathcal{A} = \langle H, Q, I, F, \Delta\rangle$, we define the language $P_\mathcal{A}$ over the alphabet[2] $(Q \times \text{Atoms} \times Q) \cup (H \times \text{Atoms})$ containing words of the form:

$$\langle q_1, a_1, p_1\rangle\langle h_1, b_1\rangle\langle q_2, a_2, p_2\rangle\langle h_2, b_2\rangle \ \dots \ \langle q_{n-1}, a_{n-1}, p_{n-1}\rangle\langle h_{n-1}, b_{n-1}\rangle\langle q_n, a_n, p_n\rangle \tag{7}$$

$(n \geq 1)$ such that, for $i = 1, \dots, n-1$, it holds $a_i \neq a_{i+1}$ and

$$p_i(a_i\mathbf{r}) \xrightarrow{\langle h_i, b_i\rangle} q_{i+1}(a_{i+1}\mathbf{r}') \tag{8}$$

is a transition of $\mathcal{A}$ at level 1 for some tuples $\mathbf{r}, \mathbf{r}' \in \text{Atoms}^{(k-1)}$, and such that $q_1 \in I$ and $p_n \in F$. The atoms a_i and a_{i+1} are here pre- and post-values of the first register, and $\mathbf{r}, \mathbf{r}'$ are pre- and post-values of the remaining $k-1$ registers. Words in P are called *altering paths.* Intutively, a letter $\langle q, a, p\rangle$ represents a run of $\mathcal{A}$ starting from a configuration $q(a\mathbf{r}')$ and ending in $p(a\mathbf{r})$, for some $\mathbf{r}, \mathbf{r}' \in \text{Atoms}^{(k-1)}$, such that the first register contains a and is preserved along the run until the automaton reaches the configuration $p(a\mathbf{r})$, from which the automaton finally updates the first register. Along this run other registers may be updated. As an immediate consequence[3] of [16, Lemma 17] we get:

▶ **Lemma 17.** *The altering path language $P_\mathcal{A}$ of a* 1*-HRA $\mathcal{A}$ has rational Parikh image.*

We observe that the altering path language of a k-HRA $\mathcal{A}$ is the same as the altering path language of a 1-HRA $\mathcal{A}'$ obtained from $\mathcal{A}$ by removing all registers except the first (smallest) one, and all transition rules of level greater than 1. Therefore, as an immediate corollary of Lemma 17 we get:

▷ Claim 18. For every $k \geq 1$, the altering path language $P_\mathcal{A}$ of k-HRA $\mathcal{A}$ has rational Parikh image.

Induction step. We now proceed to the induction step. To this aim we fix $k > 1$ and assume that languages of HRA with less than k registers have rational Parikh images. We consider a fixed k-HRA $\mathcal{A} = \langle H, Q, I, F, \Delta\rangle$ and aim at showing that Parikh image of $L(A)$ is rational. W.l.o.g. we assume that $\mathcal{A}$ is orbitized. Let $\Sigma = H \times \text{Atoms}$ denote the input alphabet.

We construct a k-HRA $\mathcal{A}_{qp}$ by removing from $\mathcal{A}$ all transition rules that update (i.e., do not preserve) the first register, and by taking q as the only initial location and p as the only accepting one. Intuitively speaking, the first register is *frozen* in $\mathcal{A}_{qp}$, in the sense that it is never updated and thus keeps its initial value a along the whole run. For $a \in \text{Atoms}$, we denote by

$$L_a(\mathcal{A}_{qp}) \ = \bigcup_{\mathbf{r},\mathbf{s} \in \text{Atoms}^{(k-1)}} L_{q(a\mathbf{r})\,p(a\mathbf{s})}(\mathcal{A}_{qp}) \ \subseteq \ L(\mathcal{A}_{qp})$$

[2] This is the unique place where we consider reacher alphabets than $H \times \text{Atoms}$, for finite H.

[3] Altering path languages considered in Lemma 17 in [16] start and end in fixed locations. The language $P_\mathcal{A}$ is thus a finite union of these languages.

the subset of $L(\mathcal{A}_{qp})$ consisting of words accepted by $\mathcal{A}_{qp}$ by a run where the value of the first register is (continuously) a. We need to deduce from the induction assumption the following claim:

▷ Claim 19. The languages $L_a(\mathcal{A}_{qp})$ have rational Parikh images.

Before proving the above claim we use it to complete the proof Theorem 14. Consider the language $K = P_{\mathcal{A}}(S)$ obtained by applying the following substitution S to the language $P_{\mathcal{A}}$:

$$\langle q, a, p\rangle \quad \mapsto \quad L_a(\mathcal{A}_{qp}) \qquad\qquad \langle h, b\rangle \quad \mapsto \quad \{\langle h, b\rangle\}.$$

In words, triples $\langle q, a, p\rangle$ are replaced by any word accepted by $\mathcal{A}_{qp}$ by a run where the value of the first register is continuously a, while pairs $\langle h, b\rangle$ are preserved.

▷ Claim 20. $L(A) = K$.

We argue that both inclusions hold. The inclusion $L(A) \subseteq K$ is shown by factorising each accepting run of $\mathcal{A}$ by transitions that update the first register, of the form (8), so that each word $w \in L(\mathcal{A})$ factorizes into:

$$w \;=\; w_1 \,\langle h_1, b_1\rangle\, w_2 \,\langle h_2, b_2\rangle\, \ldots\, w_{n-1} \,\langle h_{n-1}, b_{n-1}\rangle\, w_n, \tag{9}$$

for $w_i \in L_{a_i}(\mathcal{A}_{q_i p_i})$ for some atom a_i and control locations q_i, p_i, and therefore $w \in K$. For the reverse inclusion $K \subseteq L(A)$ consider a word $w \in K$, necessarily of the form (9), due to an altering path as in (7) and accepting runs π_i of $\mathcal{A}_{q_i p_i}$ over words w_i, where the first register is continuously equal a_i along π_i. By concatenating these runs (considered as sequences of configurations) one gets an accepting run $\pi = \pi_1 \pi_2 \ldots \pi_n$ of $\mathcal{A}$ over the word w, as required. The transitions (8) confirm that π is a run since $\mathcal{A}$ is hierarchical: all these transitions are all at level 1 and may perform (unspecified) updates of all other registers.

Having Claims 18, 19 and 20 one easily completes the proof of Theorem 14. Indeed, Parikh image of $K = P_{\mathcal{A}}(S)$ is rational due to Lemma 8, as Parikh images of $P_{\mathcal{A}}$ and all languages $L_a(\mathcal{A}_{qp})$ are so due to Claim 18 and 19, respectively, and therefore the same holds for $L(A)$, due to Claim 20.

Proof of Claim 19. For every $q, p \in Q$ we define a new $(k-1)$-HRA $\mathcal{A}'_{qp}$ that behaves exactly as $\mathcal{A}_{qp}$ except that the first register is removed. The removal of the register is compensated by an additional bit in the finite component of the alphabet of $\mathcal{A}'_{qp}$ that informs the automaton whether the input atom is equal to the (removed) first register or not.

Formally, the new automaton is $\mathcal{A}'_{qp} = \langle\{=, \neq\} \times H, Q, \{q\}, \{p\}, \Delta'\rangle$, where the transition rules Δ' are defined as follows. Due to the assumption that $\mathcal{A}$ is orbitized (and hence so are all automata $\mathcal{A}_{qp}$), its every transition constraint (5) at level i, say, either entails the equality $y = x_1$, or the inequality $y \neq x_1$. The transition rules Δ' are obtained from the transition rules of $\mathcal{A}_{qp}$ (i.e., from transition rules of $\mathcal{A}$ at level greater than 1) by transforming each transition rule

$$(q(x_1, x_2 \ldots x_k), \langle h, y\rangle, \varphi, q'(x'_1, x'_2 \ldots x'_k))$$

of $\mathcal{A}_{qp}$ to the following one:

$$(q(x_1, x_2 \ldots x_k), \langle(\sim, h), y\rangle, \varphi', q'(x'_1, x'_2 \ldots x'_k))$$

where $\sim \in \{=, \neq\}$ is chosen so that φ entails $y \sim x_1$, and φ' is obtained from φ by removing all (in)equalities referring to the first register.

By induction assumption we know that Parikh image of $\mathcal{A}'_{qp}$ is rational, for every $q, p \in Q$. For $a \in \textsc{Atoms}$, consider the following sub-alphabet (that fixes, intuitively, the value of the first register to be a):

$$\Sigma_a \;=\; \{\langle(=,h),a\rangle \;:\; h \in H\} \;\cup\; \{\langle(\neq,h),b\rangle \;:\; h \in H,\; b \in \textsc{Atoms} - \{a\}\} \;\subseteq\; \Sigma,$$

and define the languages L_{qap} as the restriction of $L(\mathcal{A}'_{qp})$ to the sub-alphabet Σ_a:

$$L_{qap} \;:=\; L(\mathcal{A}'_{qp}) \cap (\Sigma_a)^*.$$

By Lemma 6 we have:

▷ Claim 21. Parikh images of the languages L_{qap} are rational.

Finally, we observe that $L_a(\mathcal{A}_{qp})$ is obtained from L_{qap} by applying the substitution (actually, the projection):

$$\langle(\sim,h),b\rangle \quad\mapsto\quad \{\langle h,b\rangle\}$$

and therefore also has rational Parikh image, as required. This completes the proof of Claim 19, and hence also the proof of Theorem 14. ◁

◀

▶ **Corollary 22.** *Parikh images of* HRA *languages and of rational languages coincide.*

▶ **Corollary 23.** *An* NRA *has rational Parikh image if, and only if, it is Parikh-equivalent to some* HRA.

References

1 Mikołaj Bojańczyk. Slightly infinite sets. A draft of a book. URL: `https://www.mimuw.edu.pl/~bojan/paper/atom-book`.

2 Mikołaj Bojańczyk. Data monoids. In *Proc. STACS 2011*, volume 9 of *LIPIcs*, pages 105–116. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

3 Mikołaj Bojańczyk. Regular expressions for data words. Personal communication, 2020.

4 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011.

5 Mikołaj Bojańczyk, Bartek Klin, and Slawomir Lasota. Automata with group actions. In *Proc. LICS 2011*, pages 355–364, 2011.

6 Mikołaj Bojańczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3), 2014.

7 Mikołaj Bojańczyk and Sławomir Lasota. An extension of data automata that captures XPath. *Log. Methods Comput. Sci.*, 8(1), 2012.

8 Mikołaj Bojańczyk and Rafał Stefański. Single-use automata and transducers for infinite alphabets. In *Proc. ICALP 2020*, volume 168 of *LIPIcs*, pages 113:1–113:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

9 Thomas Colcombet, Clemens Ley, and Gabriele Puppis. Logics with rigidly guarded data tests. *Log. Methods Comput. Sci.*, 11(3), 2015.

10 Thomas Colcombet and Amaldev Manuel. Generalized data automata and fixpoint logic. In *Proc. FSTTCS 2014*, volume 29 of *LIPIcs*, pages 267–278. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.

11 Loris D'Antoni and Margus Veanes. Minimization of symbolic automata. In *Proc. POPL '14*, pages 541–554. ACM, 2014.

12 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009.

13 Samuel Eilenberg. *Automata, languages, and machines. A*. Pure and applied mathematics. Academic Press, 1974. URL: `https://www.worldcat.org/oclc/310535248`.

14 Nissim Francez and Michael Kaminski. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

15 Nissim Francez and Michael Kaminski. An algebraic characterization of deterministic regular languages over infinite alphabets. *Theor. Comput. Sci.*, 306(1-3):155–175, 2003.

16 Piotr Hofman, Marta Juzepczuk, Slawomir Lasota, and Mohnish Pattathurajan. Parikh's theorem for infinite alphabets. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021.

17 Michael Kaminski and Tony Tan. Regular expressions for languages over infinite alphabets. *Fundam. Informaticae*, 69(3):301–318, 2006.

18 Alexander Kurz, Tomoyuki Suzuki, and Emilio Tuosto. On nominal regular languages with binders. In Lars Birkedal, editor, *Proc. FOSSACS 2012*, volume 7213 of *Lecture Notes in Computer Science*, pages 255–269. Springer, 2012.

19 Leonid Libkin, Tony Tan, and Domagoj Vrgoc. Regular expressions for data words. *J. Comput. Syst. Sci.*, 81(7):1278–1297, 2015.

20 Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for XML transformers. *J. Comput. Syst. Sci.*, 66(1):66–97, 2003.

21 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

22 A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013.

23 Hiroshi Sakamoto and Daisuke Ikeda. Intractability of decision problems for finite-memory automata. *Theor. Comput. Sci.*, 231(2):297–308, 2000.

24 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Proc. CSL 2006*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006.

Concrete Categorical Model of a Quantum Circuit Description Language with Measurement

Dongho Lee
Université Paris-Saclay, CentraleSupélec, LMF, France & CEA, List, France

Valentin Perrelle
Université Paris-Saclay, CEA, List, France

Benoît Valiron
Université Paris-Saclay, CentraleSupélec, LMF, France

Zhaowei Xu
Université Paris-Saclay, LMF, France

Abstract

In this paper, we introduce dynamic lifting to a quantum circuit-description language, following the Proto-Quipper language approach. Dynamic lifting allows programs to transfer the result of measuring quantum data – qubits – into classical data – booleans – . We propose a type system and an operational semantics for the language and we state safety properties. Next, we introduce a concrete categorical semantics for the proposed language, basing our approach on a recent model from Rios&Selinger for Proto-Quipper-M. Our approach is to construct on top of a concrete category of circuits with measurements a Kleisli category, capturing as a side effect the action of retrieving classical content out of a quantum memory. We then show a soundness result for this semantics.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases Categorical semantics, Operational semantics, Quantum circuit description language

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.51

Funding This work was supported in part by the French National Research Agency (ANR) under the research projects SoftQPRO ANR-17-CE25-0009-02 and PPS ANR-19-CE48-0014, by the DGE of the French Ministry of Industry under the research project PIA-GDN/QuantEx P163746-484124, and by STICAMSUD 21-SITC-20 Qapla'.

Acknowledgements The authors want to thank Christophe Chareton and Sébastien Bardin for enlightening discussions.

1 Introduction

In quantum computation, one considers a special kind of memory where data is encoded on the state of objects governed by the laws of quantum mechanics. The basic unit for quantum data is the quantum bit, or qubit, and in general, a quantum memory is understood as consisting in individually addressable qubits. As derived in the no-cloning theorem [23], qubits are *non-duplicable* objects. The state of a quantum memory can be represented by a unit vector in a complex Hilbert space. Elementary operations on qubits consist in unitary operations on the state space, called *quantum gates*, and *measurements*, which are probabilistic operations returning a classical boolean.

The usual model for quantum computation is the notion of quantum circuits. Quantum circuits consist of quantum gates and wires. A wire represents a qubit, and each gate, attached to one or several wires, is a unitary operation acting on the corresponding qubits. In this model, a computation consists in allocating a quantum register, applying a circuit (i.e. the list of gates, in order), followed with a measurement to get back classical data.

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 51; pp. 51:1–51:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The *QRAM model* [8] generalizes circuits: in this model, quantum computation is performed under the control of a classical host. It emits a stream of interleaved (pieces of) quantum circuits and measurements to the quantum co-processor. The quantum co-processor executes the instructions while returning the results of measurements on the fly to the classical host. In this model, the computation is not a fixed linear list of quantum gates: the quantum gates emitted to the quantum co-processor might depend on the results of intermediate measurements. Although quantum circuits and QRAM models are equivalent in terms of expressive power, practical quantum computation is more likely to be based on the QRAM model. For this reason, many programming languages and their semantics are based on the QRAM model [17, 11, 7, 22, 24, 20, 18].

An interesting implication of this model is that the quantum circuit construction in the classical host can be dependent on the result of a measurement: there is a transfer of information from the quantum co-processor to the classical host. This feature is implemented for example in Quipper [7, 1] and QWire [11, 12]. Following Quipper's convention, we call this transfer *dynamic lifting*: classical information is *lifted* from the quantum co-processor to the classical host. Some use-cases for dynamic lifting are as follows. First, quantum error correction typically interleaves unitaries and measurements. Other examples include subroutines with repeat-until-success, where the result of the measurement on one wire says whether the computation succeeded or not [4], and measurement-based quantum computation.

The classical control over the circuit construction imposed by dynamic lifting has not been explicitly formalized in the semantics of the circuit construction languages using it. To illustrate this problem, let us look at the program in Eq. (1). The syntax we use is presented in Section 2, so we explain what the program does here: The program measures the qubit v_c and obtains the updated state of the qubit together with the resulting boolean b. Based on b, it then either allocates a new qubit initialized by true, then free the qubit v_c, or simply returns ${v_c}^1$. Despite this simple structure, the program does not correspond to a circuit because of the classical control.

$$\mathrm{exp} \quad ::= \quad \mathbf{let}\ \langle b, v_c\rangle = \mathrm{meas}(v_c)\ \mathbf{in} \quad \mathbf{if}\ b\ \mathbf{then}\ \langle \mathrm{init(tt)}, \mathrm{free}(v_c)\rangle\ \mathbf{else}\ \langle v_c, *\rangle \tag{1}$$

In QWire, the operational semantics performs normalization for composition and unbox operations but the classical control by dynamic lifting is hidden in the host term within the unbox. In Quipper, the operational semantics is encoded in Haskell's monadic type system and captures a notion of dynamic circuit including measurements. However, this semantics has never been fully formalized in the context of higher-order, functional quantum languages.

Besides operational semantics, programming languages for quantum circuits have been formalized using denotational semantics based on density matrices [11] and categorical semantics based on symmetric monoidal categories [16, 14, 9, 14, 5], or on the category of C^*-algebras [19, 13]. However, these examples of formalization do not solve the problem in that they either ignore the structure of circuit or keep the term with dynamic lifting abstract. In particular, in [14, 5], the authors construct expressive categorical models for the family of circuits – or parameterized circuits – and linear dependent type theory, respectively, while they do not provide semantics of dynamic lifting explicitly.

Our goal in this paper is to find a model and formalize a semantics for interleaved quantum circuits and dynamic lifting. The problem rests in how to analyze the structure of the computation without requiring the quantum co-processor to decide on the value of

[1] The program actually returns a pair consisting of a qubit and the unit term $*$ so that it is well-typed; we assume that the return type of free is the unit-type.

measurement. The interest of such a model and semantics is that it can serve as a test-bed to explore properties of the language. Mixing non-duplicable data, higher-order and circuits in a language yields a non-trivial system, and dependent-types were for instance only added recently for Proto-Quipper [5], with the use of such tools.

Contributions. In this paper, we propose both a small step operational semantics and a categorical semantics for a typed language – called Proto-Quipper-L – extending quantum lambda calculus [17] with circuit construction operators (box and unbox) and circuit constants. The formalization extends the one of Proto-Quipper [15]: circuits are generalized to *quantum channels* enabling the formalization of the semantics of the dynamic lifting. A quantum computation that only consists of unitary gates deterministically reduces to only one possible value. On the other hand, a quantum computation with dynamic lifting might reduce to distinct values depending on the results of the measurements. We support this by making circuits not only lists but trees branching over the results of measurements: we call such objects *quantum channels*. The language is then extended with a notion of branching terms, representing the possible choices along the computation. We prove the usual safety properties for the language: subject reduction, progress and termination (Lemmas 7, 8 and 9).

Next, we propose a sound categorical model for Proto-Quipper-L. The model is based on Proto-Quipper-M, the work of Rios&Selinger [14]. It consists of two categories: a symmetric monoidal category abstracting the notion of circuit, and an extension capturing classical computation and circuit manipulation. A morphism in the former category becomes a circuit-element in the extension. If this construction captures a sound notion of circuit, in its abstract formulation it is not *a priori* amenable to dynamic lifting. To answer the issue, we propose a concrete instantiation of the model in which dynamic lifting can be represented: we define a concrete, symmetric monoidal closed category for representing quantum channels, and, based on the construction proposed in [14], a linear category admitting a strong monad F representing the branching side-effect associated with the measurement. Following [10, 21], we use the Kleisli category $\overline{\overline{M}}_F$ to represent terms of Proto-Quipper-L.

In fact, branching monad in our categorical model corresponds to the `Circ` monad in Quipper which models non-deterministic branching in the level of type system. Although it is standard to use monad to model non-deterministic side effects, it was not clear whether such a monadic structure could be set up on the categorical model of parameterized circuits by Rios&Selinger [14]. The main result of the paper is to show how to do it, using a concrete category of quantum channels. We validate the model by showing a soundness property (Theorem 18).

2 Syntax, Types and Operational Semantics

In this section, we present the syntax of a minimal lambda-calculus for manipulating quantum channels and booleans. The language is an extension of Proto-Quipper [15].

In Proto-Quipper, the quantum lambda calculus presented in [17] is extended with circuit operators and constants. Circuit operators give an efficient way to construct circuits instead of having to sequentially apply all the gates one-by-one. Specifically, two operators on circuits are added to the language: the box operator allows us to use quantum circuits as classical data, while the unbox operator applies a boxed circuit to an argument (usually a structured set of qubits called *pattern*). Boxed circuits are first-class objects and can come with useful circuit operators like reverse and control. Technically, a circuit object in Proto-Quipper can be seen as a tuple (p, C, M) where p is structured set of the input wires of a circuit C, matching the input type of the circuit. M is a term corresponding to the output of the circuit. Along the reduction of M, the circuit C is possibly updated with new gates. The term M is open, and the output wires of C are used in M linearly, meaning that each output wire appears in M exactly once.

However, Proto-Quipper does not support dynamic lifting within circuits. To extend Proto-Quipper with dynamic lifting, we replace circuits with quantum channels and redefine the circuit operators box and unbox over quantum channels.

2.1 Quantum Channels

A quantum channel is the generalization of a quantum circuit: a tree structure where branching captures the action of measurement. In essence, quantum channels are instances of QCAlg, defined by the following grammar.

$$\text{(QCAlg)} \qquad Q, Q_1, Q_2 \ ::= \ \epsilon(W) \ \mid \ U(W)\ Q \ \mid \ \text{init } b\ w\ Q \ \mid \ \text{meas } w\ Q_1\ Q_2 \ \mid \ \text{free } w\ Q.$$

The symbols w, b, and W respectively refer to wires, booleans, and finite sets of wires. The channel $\epsilon(W)$ stands for the empty computation on the qubits W. $U(W)\ Q$ represent the unitary operator U acting on the qubits W, followed by the operations stored in the quantum channel Q. In general, U can range over a fixed set of unitary operations: we write arity(U) for the arity of U. The operator init $b\ w\ Q$ creates and initializes the wire w in state b, followed by the channel Q possibly using the newly allocated qubit. The operator meas represents the conditional branching on the result of a measurement. In our interpretation, a measurement is non-destructive: the wire being measured is still allocated and can be acted upon. The two channels Q_1 and Q_2 stand for the two possible branches to follow based on the measurement. Finally, free $w\ Q$ frees the qubit w before running Q. From now on, we call the instance of QCAlg as quantum channel object.

We define a notion of validity for quantum channels: Q is *valid* whenever, for instance, an init-node introduces a non-existing wire, or whenever a free-node acts on an existing wire. One subtlety consists in deciding what is an output wire for a branching quantum channel. For instance, consider $Q = \text{meas } w_1\ (\text{init } b\ w_2\ (\epsilon\{w_1, w_2\}))\ (\epsilon\{w_1\})$. This quantum channel admits as output $\{w_1, w_2\}$ on the left branch and $\{w_1\}$ on the right branch. We formalize this notion and write $\mathbf{out}(Q)$ for a tree-structured set of outputs of Q: Here, $\mathbf{out}(Q)$ is $[\{w_1, w_2\}, \{w_1\}]$. We also define $\mathbf{all}(Q)$ to stand for the set of *all* of the wires appearing in Q, and $\mathbf{in}(Q)$ for the set of input wires. We give a formal definition of validity from the following definition of state of quantum channel.

▶ **Definition 1** (State of quantum channel)**.** A *bunch* of elements of X is a binary tree where only the leaves are indexed, with elements of X. Formally, if x ranges over X, a bunch is built from the grammar $c_1, c_2 \ ::= \ x \mid [c_1, c_2]$. The ternary relation "st" formalizes what it means for a quantum channel to be valid. It is defined as the smallest relation satisfying the rules presented in Table 1. Informally, we say that a quantum channel Q is *valid* whenever there is some set of wires V and a bunch of sets of wires c such that $\mathbf{st}(Q, V, c)$ is derivable. Moreover, such V and c are called input and output wires of Q, respectively.

2.2 Syntax of the Terms

Having extended the notion of circuit to the notion of quantum channel, we turn to the question of the definition of the language. Compared to previous Proto-Quipper instances [15, 14, 5], there are two main changes. The first one concerns the circuit constant; the other one concerns the fact that one has to deal with non-deterministic branching computations.

Table 1 Valid quantum channel.

$$\frac{}{\mathbf{st}(\epsilon(W),\ W,\ W)} \qquad \frac{\begin{array}{l}W_1 \subseteq W\\ \mathrm{arity}(U) = |W_1|\end{array} \quad \mathbf{st}(Q,\ W,\ c)}{\mathbf{st}(U(W_1)\ Q,\ W,\ c)} \qquad \frac{w \notin W \quad \mathbf{st}(Q,\ W \cup \{w\},\ c)}{\mathbf{st}(\mathrm{init}\ b\ w\ Q,\ W,\ c)}$$

$$\frac{w \in W \quad \mathbf{st}(Q_1,\ W,\ c_a) \quad \mathbf{st}(Q_2,\ W,\ c_b)}{\mathbf{st}(\mathrm{meas}\ w\ Q_1\ Q_2,\ W,\ [c_a, c_b])} \qquad \frac{w \in W \quad \mathbf{st}(Q,\ W \setminus \{w\},\ c)}{\mathbf{st}(\mathrm{free}\ w\ Q,\ W,\ c)}$$

Table 2 Proto-Quipper-L: terms, values, patterns, branching terms, branching values, types and pattern types.

$$\begin{array}{rcl}
M, M_a, M_b & ::= & x \mid * \mid \mathrm{tt} \mid \mathrm{ff} \mid (p, Q, m) \mid \lambda x.M \mid M_a M_b \mid \langle M_a, M_b \rangle \mid \\
& & \mathbf{let}\ \langle x, y \rangle = M_a\ \mathbf{in}\ M_b \mid \mathbf{if}\ M\ \mathbf{then}\ M_a\ \mathbf{else}\ M_b \mid \mathrm{box}_P \mid \mathrm{unbox} \\
V, V_a, V_b & ::= & x \mid * \mid \mathrm{tt} \mid \mathrm{ff} \mid \lambda x.M \mid \langle V_a, V_b \rangle \mid (p, Q, v) \mid \mathrm{box}_P \mid \mathrm{unbox} \mid \mathrm{unbox}(V) \\
p, p_a, p_b & ::= & x \mid * \mid \langle p_a, p_b \rangle \\
m, m_a, m_b & ::= & M \mid [m_a, m_b] \\
v, v_a, v_b & ::= & V \mid [v_a, v_b] \\
A, A_a, A_b & ::= & I \mid \mathrm{bool} \mid \mathbf{qubit} \mid \mathrm{QChan}(P, A) \mid A_a \multimap A_b \mid A_a \otimes A_b \mid\ !\,A \\
P, P_a, P_b & ::= & I \mid \mathbf{qubit} \mid P_a \otimes P_b
\end{array}$$

We call the new language Proto-Quipper-L and define it as shown in Table 2. The constant $*$ stands for the unit term, while tt and ff stands for the booleans true and false. The term (p, Q, m) corresponds to a quantum channel object: p is a *pattern*: a structured set of input wires of a *valid* quantum channel Q, and m is a *branching term* that will match the branching structure of Q for valid quantum channel objects. For simplicity wire identifiers and term variables range over the same set of names. We then have the quantum channel operators box and unbox from Proto-Quipper: box makes a quantum channel out of a function, while unbox turns a quantum channel into a function. The rest of the constructors of the language are standard: abstraction, application, pair, let, and conditional statements. We define a notion of value in the standard way, apart from the fact that $\mathrm{unbox}(V)$ is also a value (as it is a function). Finally, branching terms and values are constructed using the branching constructor $[-, -]$. A term of the form $[M, N]$ represents a computation that has probabilistically branched and that is performing either M or N. This is novel compared to Proto-Quipper. We denote the set of free variables of a term m with $\mathrm{FV}(m)$.

One could argue that the language is missing constructors for unitary gates, qubit allocation and measurement. As in the case of Proto-Quipper, they can be defined with the unbox and quantum channel object. For instance, we can construct a measurement operation inputting a qubit and outputting a boolean and the measured wire as $\mathrm{meas} ::= \mathrm{unbox}\,(x, \mathrm{meas}\ x\ (\epsilon\{x\})\ (\epsilon\{x\}), [\langle \mathrm{tt}, x \rangle, \langle \mathrm{ff}, x \rangle])$. The tuple consists of a singleton wire name x, the quantum channel ($\mathrm{meas}\ x\ \epsilon\{x\}\ \epsilon\{x\}$), and the branching tree $[\langle \mathrm{tt}, x \rangle, \langle \mathrm{ff}, x \rangle]$. Note that the measurement operator we wrote here returns both a qubit and a boolean: we could discard the qubit with the use of a quantum channel constructor "free" if we only wanted to output a boolean. Similarly, we can also build the macros init_b and free which respectively allocates a new qubit in state b and frees a qubit, as $\mathrm{init}_b ::= \mathrm{unbox}(*, \mathrm{init}\ b\ x\,(\epsilon\{x\}), x)\,*$ and $\mathrm{free} ::= \mathrm{unbox}\,(x, \mathrm{free}\ x\ (\epsilon(\emptyset)), *)$. We can similarly define terms for unitary application by encapsulating the QCAlg constructors U inside a quantum channel object.

Table 3 Proto-Quipper-L: Typing Rules.

$$\frac{}{!\Delta,\ (x : A) \vdash x : A}\,(\text{var}) \qquad \frac{!\Delta,\ Q \vdash M : !A}{!\Delta,\ Q \vdash M : A}\,(\text{d}) \qquad \frac{!\Delta \vdash V : A \qquad V \text{ is value}}{!\Delta \vdash V : !A}\,(\text{p}) \qquad \frac{}{!\Delta \vdash * : I}\,(\text{I})$$

$$\frac{!\Delta,\ Q,\ (x : A_a) \vdash M : A_b}{!\Delta,\ Q \vdash \lambda x.M : A_a \multimap A_b}\,(\multimap_I) \qquad \frac{!\Delta,\ Q_a \vdash M_a : A_a \multimap A_b \qquad !\Delta,\ Q_b \vdash M_b : A_a}{!\Delta,\ Q_a,\ Q_b \vdash M_a M_b : A_b}\,(\multimap_E)$$

$$\frac{!\Delta,\ Q_a \vdash M : \mathrm{bool} \qquad \begin{matrix} !\Delta,\ Q_b \vdash M_1 : A \\ !\Delta,\ Q_b \vdash M_2 : A \end{matrix}}{!\Delta,\ Q_a,\ Q_b \vdash \mathbf{if}\ M\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2 : A}\,(\text{if}) \qquad \frac{\begin{matrix} !\Delta,\ Q_a \vdash M_a : A_a \otimes A_b \\ !\Delta,\ Q_b,\ (x : A_a),\ (y : A_b) \vdash M_b : A \end{matrix}}{!\Delta,\ Q_a,\ Q_b \vdash \mathbf{let}\ \langle x, y\rangle = M_a\ \mathbf{in}\ M_b : A}\,(\otimes_E)$$

$$\frac{}{!\Delta \vdash \mathrm{tt} : \mathrm{bool}}\,(\text{tt}) \qquad \frac{}{!\Delta \vdash \mathrm{ff} : \mathrm{bool}}\,(\text{ff}) \qquad \frac{!\Delta,\ Q_a \vdash M_a : A_a \qquad !\Delta,\ Q_b \vdash M_b : A_b}{!\Delta,\ Q_a,\ Q_b \vdash \langle M_a, M_b\rangle : A_1 \otimes A_2}\,(\otimes_I)$$

$$\frac{}{!\Delta \vdash \mathrm{box}_P : !(P \multimap A) \multimap !\mathrm{QChan}(P, A)}\,(\text{box}) \qquad \frac{}{!\Delta \vdash \mathrm{unbox} : \mathrm{QChan}(P, A) \multimap (P \multimap A)}\,(\text{unbox})$$

$$\frac{\gamma_a \vdash m_a : A \qquad \gamma_b \vdash m_b : A}{\gamma_a \times \gamma_b \vdash [m_a, m_b] : A}\,(\text{b}) \qquad \frac{p \vDash P \qquad \mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A)}{!\Delta \vdash (p, Q, m) : !\mathrm{QChan}(P, A)}\,(\mathrm{QChan}_I)$$

2.3 Type System

Types of Proto-Quipper-L are defined as in Table 2. Following the standard strategy [17, 15, 3] to account for the non-duplicability brought by the quantum memory, we are using a type system based on linear logic [6]. Types consist in the constant types I, bool, **qubit**; the function type $A_a \multimap A_b$; the type for pairs $A_a \otimes A_b$; the type $!A$ of duplicable terms of type A; the type of quantum channels $\mathrm{QChan}(P, A)$ with input of type P and output of type A, where P refers to patterns, that is, first-order types constructed from **qubit**s and tensors.

Conventionally, a typing judgment consists in a typing context, which maps variables to types, and a term assigned with a type. However, in Proto-Quipper-L, the term can be a branching term. Although the terms of all branches in a branching term are assigned with the same type, they may have different typing contexts. This is formalized in two distinct definitions of typing judgments: regular typing judgments $\Gamma \vdash M : A$ where where Γ is a list of typed variables and M is a non-branching term, and *branching typing judgments* $\gamma \vdash m : A$, where m is a branching term and γ is an branching typing context: $\gamma ::= \Gamma \mid \gamma_1 \times \gamma_2$.

A judgment is valid if it can be derived from the typing rules presented in Table 3. The rules ensure that various constraints necessary for soundness are satisfied. One can note that all terms constituting a branching term share the same type ; that valid branching typing judgments have branching contexts and terms with the same tree structure ; that a quantum channel object is duplicable with type !QChan ; that box sends a duplicable function to a duplicable quantum channel object, and that unbox sends a quantum channel object to a function. One can also note that only values can be promoted to duplicable objects: this is due to the call-by-value reduction strategy we follow. The relation $\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A)$ in the (QChan_I) rule ensures that one can derive typing derivations for each term leaf of m given that the output wires of the quantum channel Q is assigned with type **qubit** within the typing context. The relation $p \vDash P$ simply states that the shapes of p and P match and that the variables occurring in p are pairwise distinct.

The rules for **vBind** are found in Table 4. Note that the non-linear context $!\Delta$ is a list of pairs of variables and non-linear types. We denote by $\mathbf{FV}(!\Delta)$ the set of variables in $!\Delta$. In fact, the condition $(X \cap \mathbf{FV}(!\Delta) = \emptyset)$ is implicitly assumed by the definition of the typing judgment $(!\Delta,\ (x : \mathbf{qubit})_{x \in X} \vdash M : A)$.

Table 4 Validity of binding in quantum channel constant.

$$\frac{X \cap \mathbf{FV}(!\Delta) = \emptyset \qquad !\Delta,\ (x : \mathbf{qubit})_{x \in X} \vdash M : A}{\mathbf{vBind}(!\Delta, X, M, A)}(\mathbf{vBind}_{nb}) \qquad \frac{}{* \vDash I} \qquad \frac{}{x \vDash \mathbf{qubit}}$$

$$\frac{\mathbf{vBind}(!\Delta, c_a, m_a, A) \qquad \mathbf{vBind}(!\Delta, c_b, m_b, A)}{\mathbf{vBind}(!\Delta, [c_a, c_b], [m_a, m_b], A)}(\mathbf{vBind}_{b}) \qquad \frac{\begin{array}{c}\forall i,\ p_i \vDash P_i \\ \mathbf{FV}(p_1) \cap \mathbf{FV}(p_2) = \emptyset\end{array}}{\langle p_1, p_2 \rangle \vDash P_1 \otimes P_2}$$

▶ **Example 2.** In Section 2.2 we defined three macros: meas, free and init$_b$. We can type meas with $!(\mathbf{qubit} \multimap (\mathrm{bool} \otimes \mathbf{qubit}))$ and free with $!(\mathbf{qubit} \multimap I)$. For init$_b$, note that because there is a final argument "$*$", it is really an application and we can therefore only type it with **qubit** and not !**qubit**: this is expected, as we don't want to be able to construct duplicable qubits. With these types, we can now type the term exp in Eq (1) of Section 1: we can derive the judgment $v_c : \mathbf{qubit} \vdash \mathrm{exp} : \mathbf{qubit} \otimes I$.

▶ Remark 3. In general, there can be more than one typing derivation for a typing judgment but, for the types I, **qubit** or bool, there is a unique typing derivation when the term is a value. We call these types *basic types*.

2.4 Operational Semantics

The computational model we have in mind for the language is a reduction-based semantics specialized to circuit construction: the operational semantics is modeling an I/O side-effect, where gates are emitted and buffered in a quantum channel. Based on Proto-Quipper [15], the operational semantics we describe therefore updates a *configuration* consisting of a pair (Q, m): a buffered QCAlg object and a branching term. The term m is reduced up to a value representing the final state of the computation. Along the computation, quantum gates might be emitted to the co-processor: the quantum channel Q keeps track of these. One can notice that a configuration corresponds to a quantum channel constant without the input wires, where there is a minor relaxation on the linearity of the output wires of Q in m which will be recovered when we define well-typed configuration.

▶ **Definition 4.** A *circuit-buffering configuration* is a pair (Q, m) as described above. It is said to be *valid* whenever Q is valid, Q and m share the same tree-structure, and whenever output wires of Q corresponds to free variables of m (following the tree-structure). So for instance, $V \subseteq \mathbf{FV}(M)$ implies the validity of $(\epsilon(V),\ M)$, and whenever $(Q_1,\ m_a)$ and $(Q_2,\ m_b)$ are valid so is $(\mathrm{meas}\ w\ Q_1\ Q_2,\ [m_a, m_b])$.

▶ Remark 5. In order to define the reduction rules, we need to be able to extend a configuration with new wires. For instance, let us consider the term $(\epsilon\{x, y\}, \mathbf{if}\ (N\,x)\ \mathbf{then}\ y\ \mathbf{else}\ y)$ with N some term not containing y. Evaluating this configuration requires to first evaluate $N\,x$ and possibly append a few gates to $\epsilon\{x, y\}$. However, this can be factorized as first evaluating $(\epsilon\{x\}, N\,x)$ to (Q, V) and then adding back the wire y to the resulting quantum channel Q. We therefore define an operator **extend** taking a quantum channel and a set of wire names, adding them as unused wires to the quantum channel.

The reduction rules for Proto-Quipper-L are defined in Table 5. (See Section A.1 for more details.) Rules (a.x) always hold (b ranges over $\{\mathrm{tt}, \mathrm{ff}\}$). In Rules (b.1), p is a pattern of same shape as P made from dynamically allocated fresh variables. In Rule (b.2), p and V have the same shape, and σ is a substitution mapping p to V. Provided that $(Q, m) \to (Q', m')$, we have

Table 5 Reduction rules for operational semantics.

(a.1)	$(\epsilon(W), (\lambda x.M)V) \to (\epsilon(W), M[V/x])$	$(\epsilon(W_{C[M]}),\ C[M]) \to (\mathbf{extend}(Q, W_{C[-]}),\ C[m])$	(c)
(a.2)	$(\epsilon(W), \mathbf{let}\ \langle x, y\rangle = \langle V, U\rangle\ \mathbf{in}\ M) \to (\epsilon(W), M[V/x, U/y])$	$(Q,\ [m_a, m_b]) \to (Q',\ [m_c, m_d])$	(d.1)
(a.3)	$(\epsilon(W), \mathbf{if}\ \mathrm{b}\ \mathbf{then}\ M_{\mathrm{tt}}\ \mathbf{else}\ M_{\mathrm{ff}}) \to (\epsilon(W), M_b)$	$(Q,\ [m_a, v]) \to (Q',\ [m_c, v])$	(d.2)
(b.1)	$(\epsilon(\emptyset), \mathrm{box}_P\, V) \to (\epsilon(\emptyset), (p, \epsilon(\mathrm{FV}(p)), Vp))$	$(Q,\ [v, m_b]) \to (Q',\ [v, m_d])$	(d.3)
(b.2)	$(\epsilon(\mathbf{FV}(V)), (\mathrm{unbox}(p,\ Q,\ u))V) \to (\sigma(Q), \sigma(u))$	$(G\ Q_1,\ m_a) \to (G\ Q_3,\ m_c)$	(d.3)

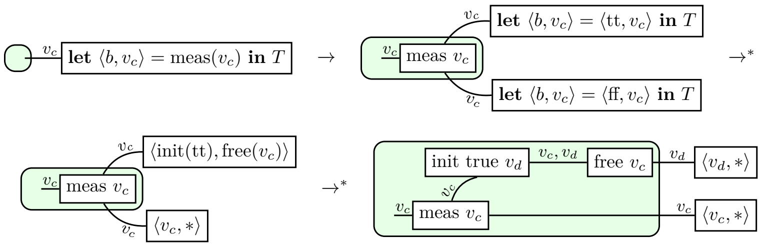

v_c
$\mathbf{let}\ \langle b, v_c\rangle = \mathrm{meas}(v_c)\ \mathbf{in}\ T$
$\to$
v_c
$\mathbf{let}\ \langle b, v_c\rangle = \langle \mathrm{tt}, v_c\rangle\ \mathbf{in}\ T$
v_c
meas v_c
v_c
$\mathbf{let}\ \langle b, v_c\rangle = \langle \mathrm{ff}, v_c\rangle\ \mathbf{in}\ T$
$\to^*$
v_c
$\langle \mathrm{init}(\mathrm{tt}), \mathrm{free}(v_c)\rangle$
v_c
meas v_c
v_c
$\langle v_c, *\rangle$
$\to^*$
init true v_d
v_c, v_d
free v_c
v_d
$\langle v_d, *\rangle$
v_c
v_c
meas v_c
v_c
$\langle v_c, *\rangle$

Figure 1 Reduction of the term of Eq (1).

$(\epsilon(\emptyset), (p, Q, m)) \to (\epsilon(\emptyset), (p, Q', m'))$. Provided that we have that $(\epsilon(W_M),\ M) \to (Q,\ m)$, that $\mathbf{all}(Q) \cap W_N = \emptyset$ and that $\mathbf{all}(Q) \cap W_V = \emptyset$, the class of rules (c) apply. There, $C[-]$ ranges over $[-]N$, $V[-]$, $\langle[-], N\rangle$, $\langle V, [0]\rangle$, $\mathrm{if}\,[-]\mathrm{then}\, M_a\, \mathrm{else}\, M_b$ and $\mathrm{let}\ \langle x, y\rangle = [-]\, \mathrm{in}\, N$. We use syntactic sugar for combining terms and branching terms, as in $C[m]$. It corresponds to the term constructor applied to each leaf of m, for instance: for $m = [[N_1, N_2], N_3]$, $C[m] := [[C[N_1], C[N_2]], C[N_3]]$. In Rules (d.x), Q stands for meas $w\ Q_1\ Q_2$ and Q' for meas $w\ Q_3\ Q_4$. These rules apply whenever $(Q_1,\ m_a) \to (Q_3,\ m_c)$ and $(Q_2,\ m_b) \to (Q_4,\ m_d)$. In (d.3), G ranges over $U(W)$, init $b\ w$ and free w.

▶ **Example 6.** As an example, we show the reduction of the term shown in Eq. (1). For convenience, we define T as $\mathbf{if}\ b\ \mathbf{then}\ \langle \mathrm{init}(\mathrm{tt}), \mathrm{free}(v_c)\rangle\ \mathbf{else}\ \langle v_c, *\rangle$. Figure 1 shows the reduction of the term. (check Section A.2 for more details). We use a graphical representation for configuration. A green box represents a quantum channel whose leaves are linked to square-boxed terms. The edges represent bundles of wires, which can contain multiple wires and can be empty.

In the first line, the measurement in the term is reduced by the structural rule for *let* and the reduction rule for measurement creating a branching term. Then, each term at a leaf of the tree is reduced into the left-most configuration of the second line. Note how classical computation can happen inside the leaves. The second line of the figure shows the application of initialization and free operation. In particular, note how the tree expands as the computation progresses.

2.5 Type safety for Proto-Quipper-L

In order to state the type safety theorem, we need to extend typing derivations to configurations. We write $!\Delta \vdash (Q, m) : A$ whenever $\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A)$ and Q is valid. Note that the definition implies that the output wires of the quantum channel correspond to the linear variables of type $\mathbf{qubit}$ in the context of the typing derivation. In any case, we can now state type safety for Proto-Quipper-L, as follows.

▶ **Lemma 7** (Subject reduction). *For any configurations* (Q_1, m_1) *and* (Q_2, m_2) *such that* $(Q_1, m_1) \to (Q_2, m_2)$*, if* $\vdash (Q_1, m_1) : A$*, then* $\vdash (Q_2, m_2) : A$*.*

▶ **Lemma 8** (Progress). *If* $(\vdash (Q_1, m_1) : A)$*, then either there exists* (Q_2, m_2) *such that* $(Q_1, m_1) \to (Q_2, m_2)$ *or* m_1 *is a branching value.*

▶ **Lemma 9** (Termination). *Given a well-typed configuration* $\vdash (Q, m) : A$*, any reduction sequence starting with* (Q, m) *is terminating.*

3 Categorical semantics

In this section, we turn to the question of developing a categorical semantics for Proto-Quipper-L. The categorical semantics of circuit-description languages and Proto-Quipper in particular originates from Rios&Selinger [14]. They developed a model parametrized by a symmetric monoidal category M. In their model one can therefore interpret higher-order circuit-description languages, and several extensions of the semantics [5, 9] have been discussed. However, none of them were shown to be able to capture dynamic lifting: the possibility to change behavior depending on the result of a measurement.

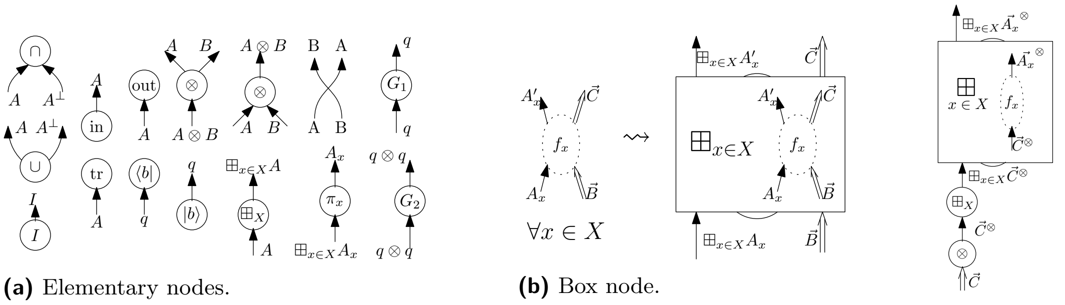
Our proposal. What we propose in this paper is a concrete, symmetric monoidal category M such that applying Rios&Selinger's construction gives us also access to an interpretation of dynamic lifting. The model we propose follows Moggi's categorical interpretation of side effect [10] and models the action of measurement using a (strong) monad. Our semantics is therefore based on: (1) A category of *diagrams*, serving as graph-like abstractions of quantum channels. This category is compact-closed and features products: it matches the requirements of the base category $\overline{M}$ in Rios&Selinger's work. This category is discussed in Section 3.1. (2) The category $\overline{\overline{M}}$, extending $\overline{M}$ with the same procedure as Rios&Selinger. This category is the category of *values*, following Moggi's computational interpretation. It is presented in Section 3.2. (3) A strong monad on $\overline{\overline{M}}$ that we denote with F. This monad encapsulates *computations* involving measurements: a general term of Proto-Quipper-L is therefore interpreted inside the Kleisli category $\overline{\overline{M}}_F$: This is the main novelty compared to other models of Proto-Quipper-like languages [14, 5, 9], and the critical reason for the possibility to interpret dynamic lifting. This is discussed in Section 3.4.

Finally, we discuss the soundness of the model and presents a few examples. For sake of space, the presentation of the definitions and results is only kept to a minimum: more information is available in the appendix.

3.1 Categories of Diagrams

In this section, we aim at building a category of quantum channels. We first define a graph-based language: we call the corresponding terms *diagrams* to distinguish them from the terms of QCAlg of Section 2.1: these are directed graphs with edges labeled with *marks*. We then build the category $\overline{M}$ out of these terms.

Marks. Formally, we define *marks* with the grammar $M ::= q \mid M \otimes M \mid \boxplus_{i \in X} M_i \mid M^{\perp}$, where X ranges over the class of sets, and is subject to the equivalence relation defined as $\boxplus_{i \in I} \boxplus_{j \in J} M_{(i,j)} = \boxplus_{j \in J} \boxplus_{i \in I} M_{(i,j)}$; $(M_1 \otimes M_2)^{\perp} = M_1^{\perp} \otimes M_2^{\perp}$; $(\boxplus_{i \in I} M_i)^{\perp} = \boxplus_{i \in I}(M_i^{\perp})$; $(M^{\perp})^{\perp} = M$; $\boxplus_{l \in L} \boxplus_{x \in l} M_{(l,x)} = \boxplus_{x \in l_1 ++ \cdots ++ l_n} M_{(l,x)}$, whenever $L = [l_1, \ldots, l_n]$. Note that $\boxplus_{i \in \emptyset} M_i$ acts as a unit for $\boxplus$: we denote it with I. If $A = [A_1 \ldots A_n]$ is a list of marks, we use the notation $A^{\otimes}$ for $A_1 \otimes \cdots \otimes A_n$. We also use a binary notation for $\boxplus$ when the indexed set contains 2 elements: $\boxplus_{x \in \{a,b\}} A_x = A_a \boxplus A_b$.



(a) Elementary nodes.

(b) Box node.

(c) Product.

Figure 2 Diagram Nodes and Product.

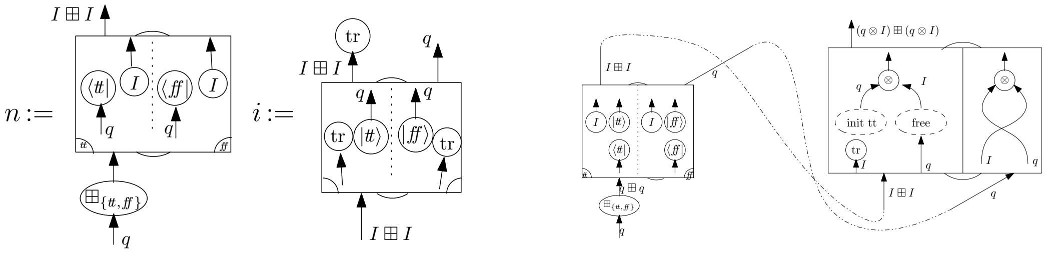
▶ Remark 10. Box node is a way of representing additive connectives of intuitionistic linear logic. It can be considered as a set of different proofs depending on the choice made for the additive connective. Note that we are following the convention of linear logic for $(-)^\perp$, where the $(-)^\perp$ operator is not changing the order of tensors.

Diagrams. A diagram is a (possibly infinite) directed graph with edges indexed with marks and built from *elementary nodes* and *boxes*. A diagram is not necessarily a connected graph. By graphical convention, all edges are flowing upward: a diagram is therefore acyclic.

Elementary nodes make the basic building blocks of diagrams: they are shown on Figure 2a. As we work with directed graphs, each edge connected to a node is either an input or an output for that node. There are several kinds of elementary nodes: the structural nodes for capturing the compact closed structure: \textcircled{$\cup$}, \textcircled{$\cap$}, \textcircled{$\otimes$}, \textcircled{I} and the swap-node (also written \textcircled{σ}); the structural nodes for handling the product: \textcircled{$\boxplus$} for the diagonal map and \textcircled{π} for the projection; the structural nodes for pointing inputs \textcircled{in} and outputs \textcircled{out} of diagrams; the nodes specifically for quantum: \textcircled{$|b\rangle$} and \textcircled{$\langle b|$}, with b ranging over booleans, where the former stands for initialization and the latter for projection onto the corresponding basis, \textcircled{tr} for representing tracing (also useful for products), \textcircled{G_1} for unary unitary gates and \textcircled{G_2} for binary gates. Note that the nodes allows more expressivity than what we need: for instance, \textcircled{tr} and \textcircled{$\langle b|$} are indistinguishable. We nonetheless keep them in order to draw attention on the correspondence with quantum computation and an obvious mapping to completely positive maps. For the sake of legibility, we do not draw in and out nodes unless necessary.

Presented in Figure 2b, a box-node is built from a family of diagrams. They should all share the same input and output marks except for one pair of input/output (represented on the left of the box-node). As a node, box-node has the same input and output marks as its contained diagrams except that the left-most marks: these are the $\boxplus$ of all left-most marks of the family. We represent juxtaposition of edges as a double arrows. This node is the last piece needed for representing products.

Equivalence relation on diagrams. We define an equivalence relation on diagrams. The equivalence is given with local rules that can be extended to larger diagrams coherently: subgraphs can be rewritten inside a larger graph. These rules exactly capture what is needed for the categorical semantics to work. For instance, we include all of the rules for compact closed categories [16]. We also for instance need the fact that the \textcircled{π}-node acts as a projection over box-nodes. The complete list can be found in the appendix.



(a) Morphisms in $\overline{M}$.

(b) Figure for Example 17.

Figure 3 Examples of Morphisms.

Category of Diagrams. Based on the definition of diagrams, we define the category of diagrams $\overline{M}$: object are lists of marks $[A_1, \ldots, A_n]$, and a morphism $[A_1, \ldots, A_n] \to [B_1, \ldots, B_m]$ is a diagram with $\textcircled{\scriptsize in}$-nodes of marks A_i and $\textcircled{\scriptsize out}$-nodes of marks B_i, modulo the equivalence relation defined on diagrams. We use the notation $\vec{A}$ for the list of the A_i's. An identity morphism is a diagram consisting of a bunch of simple edges connecting $\textcircled{\scriptsize in}$ and $\textcircled{\scriptsize out}$ nodes. Composition consists in identifying $\textcircled{\scriptsize out}$ and $\textcircled{\scriptsize in}$ nodes of diagrams. The category $\overline{M}$ is symmetric monoidal: The unit is $I = []$, the empty list, and the monoidal structure is given with $\otimes : \overline{M} \times \overline{M} \to \overline{M}$ defined as $[A_1, \ldots, A_n] \otimes [B_1, \ldots, B_m] = [A_1, \ldots, A_n, B_1, \ldots, B_m]$, and $f \otimes g$ the juxtaposition of diagrams. As for standard graphical representation of symmetric monoidal structure [16], the associativity, unit laws and symmetry of the tensor product follow their graphical conventions. Finally, the operation on marks $(-)^\perp$ lifts to a contravariant functor, giving a compact-closed structure to $\overline{M}$. It then admits an internal hom: $\vec{A} \multimap \vec{B}$ can be defined as $[A_1, \ldots, A_n] \multimap [B_1, \ldots, B_m] = [A_1^\perp, \ldots, A_n^\perp, B_1, \ldots, B_m]$. Thanks to the $\textcircled{\scriptsize π}$-nodes and the corresponding diagram equivalence rules, the category $\overline{M}$ also has products: for any family of objects $\{\vec{A_x} \mid x \in X\}$ indexed by a set X, let $\times_{x\in X} \vec{A_x} = [\boxplus_{x \in X} \vec{A_x}^{\otimes}]$ be the product of the family of objects. Then, the family of projections $\pi_x : \times_{x \in X} A_x \to A_x$ is given by the π-node. Finally, for any family of maps $\{f_x : C \to A_x\}_{x \in X}$, the morphisms $\langle f_x \rangle : C \to \times_{x \in X} A_x$ is the diagram presented in Figure 2c. As an abuse of notation we use one $\textcircled{\scriptsize $\otimes$}$ for tensoring several wires at once.

▶ Remark 11. The category of diagrams is strongly inspired from proof nets: tensor nodes correspond to multiplicative connectives while boxes correspond to additive connectives.

Examples of morphisms in $\overline{M}$. Lastly, we show in Figure 3a two interesting morphisms in the category $\overline{M}$. The morphism $n : [q] \to [I \boxplus I]$ corresponds to the measure: in each branch we perform a projection, and we keep in the output the information of where we were. Note that the semantics does not state what is doing $\langle tt|$: what is important is to (1) "remove" the q-wire, and (2) keep as information if we are on the "true" or the "false" part. The morphism i corresponds to qubit creation: it takes a boolean $I \boxplus I$, initializes a qubit depending on its state and "forgets" the boolean. As a last example we can build the injections $I \to I \boxplus I$ in a similar way to n: first a $\textcircled{\scriptsize $\boxplus$}$-node, followed with a box-node where we trace out the component we do not need.

▶ Remark 12. The object $[I \boxplus I]$ corresponds to the bit-type in Quipper or in Proto-Quipper, corresponding to boolean values within the quantum co-processor, and manipulated with circuits in Quipper. For simplicity we did not include such a bit-type in the language, but it does exist in the model.

▶ **Definition 13** (Intepreting QCAlg terms). Let us use the notation $q^{\otimes n}$ to represent a list $[q, ...q]$ of size n. A QCAlg-term Q can be interpreted as a $\overline{M}$-morphism $[\![Q]\!] : A \to B$, where $A = q^{\otimes \mathbf{in}(Q)}$ and B of tree-shape for instance $(q^{\otimes n_1} \boxplus q^{\otimes n_2}) \boxplus q^{\otimes n_3}$, following the tree-shape of out(Q). The $\overline{M}$-morphism $[\![Q]\!]$ is then defined by induction, using the idea presented above: initialization and unitary gates are simply composed, and the branches of a meas operation are encapsulated inside box-nodes.

3.2 Coproduct completion

Coproduct completion allows us to define families of circuits [14, 5]: the categorical structure clearly separate what is *purely quantum* and what is *parameter* to the computation: we have *parametric* families of quantum channels. Formally, this is done using the coproduct completion $\overline{\overline{M}}$ of $\overline{M}$. In this completion, an object corresponds to a pair $(X, (A_x)_{x \in X})$ where X is a set and A_x is an object of $\overline{M}$ for each $x \in X$: This should be understood as a *parametric* families of objects of $\overline{M}$. A morphism from $(X, (A_x))$ to $(Y, (B_y))$ corresponds to a pair $(f_0, (f_x)_{x \in X})$ where $f_0 : X \to Y$ is a set function and $f_x : A_x \to B_{f_0(x)}$ is a morphism in $\overline{M}$ for each $x \in X$. Intuitively, to each choice of parameter x we have a $\overline{M}$-morphisms $A_x \to A_{f_0(x)}$. Composition is defined with $(g_0, (g_y)) \circ (f_0, (f_x)) = (g_0 \circ f_0, (g_{f_0(x)} \circ f_x))$ where $(g_0, (g_y)) : (Y, (B_y)) \to (Z, (C_x))$ and $(f_0, (f_x)) : (X, (A_x)) \to (Y, (B_y))$ are morphisms in $\overline{\overline{M}}$, while the identity is $\mathbf{id}_A = (\mathbf{id}_X, (\mathbf{id}_{A_x}))$ for an object $A = (X, (A_x))$.

According to Rios&Selinger [14], the category $\overline{\overline{M}}$ is symmetric monoidal closed, and features products and co-products. In particular, the monoidal unit is $(\{\emptyset\}, (I))$ (where $\emptyset$ stands for the only representative of the singleton-set), and when $A = (X, (A_x))$ and $B = (Y, (B_y))$, the tensor on objects is $A \otimes B = (X \times Y, (A_x \otimes B_y)_{(x,y)})$ and the internal hom is $A \multimap B = (X \to Y, (C_f)_{f \in X \to Y})$ ($X \to Y$ is the set of all set-functions from X to Y and C_f refers to the product $\boxplus_{x \in X}(A_x \multimap B_{f(x)})$ of internal homs in $\overline{M}$). Note that the product is defined by $\boxplus$ in the case of the category of diagrams. Also note that compared to [14], we can capitalize on the concrete structure of the category for the proofs involving the coproduct completion. For instance, the associativity is trivial in our category $\overline{M}$.

Finally, in order to model the type operator "!", Rios&Selinger rely on Benton's linear/non-linear model [2], based on an adjunction between a symmetric monoidal closed category and a cartesian closed category. In our case, as in [14] the adjunction is built between the SMCC $\overline{\overline{M}}$ and the cartesian closed category **Set**. The two functors of the adjunction are $p : \mathbf{Set} \to \overline{\overline{M}}$, defined on objects as $p(X) = (X, (I)_X)$, and $b : \overline{\overline{M}} \to \mathbf{Set}$, defined on objects as $b(X, (A_x)) = \sum_{x \in X} \overline{M}(I, A_x)$ where $\overline{M}(I, A_x)$ is the set of morphisms between the objects I and A_x of the category $\overline{M}$ and $\sum_{x \in X} \overline{M}(I, A_x)$ is the disjoint union of all such sets over X . From the adjunction, one can then construct a comonad "!" defined as $! = p \circ b$.

▶ Remark 14. In $\overline{\overline{M}}$ there are two classes of interesting objects. The *parameters* are objects of the form $(X, (I)_{x \in X})$: the family consists in trivial objects of $\overline{M}$, and the only information is given by…the parameter. The *state* object is the dual: the parameter is trivial and the family is of size 1 with only one object of $\overline{M}$. It is then of the form $(\{\emptyset\}, (A))$. One therefore has two booleans: a parameter boolean $b_p = (\{\mathit{tt}, \mathit{ff}\}, (I)_{\{\mathit{tt}, \mathit{ff}\}})$ and the state boolean $b_s = (\{\emptyset\}, (I \boxplus I))$ living in $\overline{M}$.

3.3 Monad for Branching Computation

According to Rios&Selinger, the category $\overline{\overline{M}}$ together with the structure sketched in Section 3.2 forms a model of Proto-Quipper-M. We shall now see how our concrete construction can also support dynamic lifting, therefore forming a model of Proto-Quipper-L.

The main problem consists in *lifting* a branching sitting inside a quantum channel – i.e. inside the category $\overline{M}$ – to turn it into a coproduct on which one can act upon in the classical world, represented by the category $\overline{\overline{M}}$: as in Remark 14, we need to lift a state-boolean into a parameter-boolean. Our strategy consists in defining a strong monad (F, μ, η, t) to capture the action of retrieving such a branching: a term featuring measurement (and dynamic lifting) is therefore represented within the Kleisli category $\overline{\overline{M}}_F$, following Moggi's [10] view on side-effects.

The functor $F : \overline{\overline{M}} \to \overline{\overline{M}}$ is defined as follows. For an object $A = (X, (A_x))$, we define $F(A) = (\mathrm{mset}(X), ([\boxplus_{x\in l} A_x^{\otimes}])_{l\in\mathrm{mset}(X)})$, where $\mathrm{mset}(X)$ is the set of multisets of X, while for a morphism $f = (f_0, (f_x)) : A \to B$ we set $F(f) = (g_0 : \mathrm{mset}(X) \to \mathrm{mset}(Y), g_l : [\boxplus_{x\in l} A_x^{\otimes}] \to [\boxplus_{y\in g_0(l)} B_y^{\otimes}])$, where $g_0 = \{[x_0, \ldots, x_n] \mapsto [f_0(x_0), \ldots, f_0(x_n)]\}$ and where g_l is defined as shown on the right.

▶ **Example 15.** The lifting of the state boolean b_s of Remark 14 to the parameter boolean b_p is then a $\overline{\overline{M}}$-map $\mathrm{lb} : b_s \to F(b_p)$, where $F(b_p)$ is $(\mathrm{mset}\{\mathit{tt}, \mathit{ff}\}, (\boxplus_{x\in l} I)_l)$. The map lb is defined as $(\mathrm{lb}_0, (f_x)_x)$ where $\mathrm{lb}_0 : \{\emptyset\} \to \mathrm{mset}\{\mathit{tt}, \mathit{ff}\}$ sends $\emptyset$ to $[\mathit{tt}, \mathit{ff}]$, and where $\mathrm{lb}_\emptyset : I \boxplus I \to I \boxplus I$ is simply defined as the identity. In the other direction, the $\overline{\overline{M}}$-map $b_p \to b_s$ consists of the constant set-function on $\emptyset$ together with the injections $I \to I \boxplus I$ discussed in Section 3.1.

▶ Remark 16. In Quipper dynamic lifting is implemented in the `Circ` monad which corresponds to the strong monad of F in our model. The branching side-effect corresponds to the `RW_Read` constructor of the `Circ` monad.

3.4 Interpreting Typed Terms and Configurations

In this section, we introduce an interpretation of Proto-Quipper-L within the Kleisli category $\overline{\overline{M}}_F$. As it is customary, types are mapped to objects while typing derivations are mapped to morphisms. When typed terms admit a unique typing derivation this entails a unique denotation for typed terms. In our situation, due to the promotion and dereliction rules typing derivations are not necessarily unique: we therefore adjust the statements of the lemmas and theorems accordingly. However, in the case of values of basic types, thanks to Remark 3 and the type safety properties, the denotation of closed terms of basic types is independent from the choice of typing derivation: this gives Corollary 19.

The interpretation $[\![A]\!]$ of a type A is directly built against the categorical structure: $[\![I]\!] = (\{\emptyset\}, (I))$, $[\![\mathrm{bool}]\!] = (\{\mathrm{tt}, \mathrm{ff}\}, (I, I))$, $[\![\mathbf{qubit}]\!] = (\{\emptyset\}, ([q]))$, $[\![A_a \multimap A_b]\!] = [\![A_a]\!] \multimap_{\overline{\overline{M}}_F} [\![A_b]\!]$ the internal hom in the category $\overline{\overline{M}}_F$, $[\![A_a \otimes B_b]\!] = [\![A_a]\!] \otimes [\![A_b]\!]$, $[\![!A]\!] = ![\![A]\!] = (p \circ b)[\![A]\!]$. Finally, for quantum channels we follow Rios&Selinger's strategy by defining $[\![\mathrm{QChan}(P, B)]\!] = p(\overline{\overline{M}}_F([\![P]\!], [\![A]\!]))$. In our situation, the set $\overline{\overline{M}}_F(A, B)$ is isomorphic to $\overline{M}(A, B)$ when A and B are state objects: in this situation, QChan-types indeed correspond to morphisms of the category $\overline{M}$, i.e. quantum channels: this is used to interpret the box and unbox operators. The quantum channel constant is just an encapsulation over Definition 13. Finally, a typed configuration $!\Delta \vdash (Q, m) : A$ is interpreted as the composition of Q (i.e. we first "compute" Q) followed with the interpretation of M.

▶ **Example 17.** The term exp of Eq.(1) in Section 1 has for interpretation a morphism $(\{\emptyset\}, (q)) \to (\mathrm{mset}\{(\emptyset, \emptyset)\}, (q)_l)$ defined as $(f_0, (f_\emptyset))$ where $f_0(\emptyset) = [(\emptyset, \emptyset), (\emptyset, \emptyset)]$ and $f_\emptyset$ is defined as shown in Figure 3b (the dashed lines are meant to be vertical). The bottom box-node represents the measurement ($I \boxplus I$ being the result) and the upper one the test. The top result is a $\boxplus$-superposition of 2 copies of $q \otimes I$, as expected: these stand for the two "classical" possibilities.

In general, soundness of categorical semantics states that the categorical interpretation of the typing derivation is preserved over the reduction. However, there can be multiple type derivations for each type judgement, in our type system, because of the reason explained above. Therefore, in this paper, we show that for a type judgement and a typing derivation, there exists a particular typing judgement of the reduced type judgement which has the same interpretation of the original typing derivation.

▶ **Theorem 18** (Soundness). *For any configurations (Q_1, m_1) and (Q_2, m_2) such that $(Q_1, m_1) \to (Q_2, m_2)$, if $\vdash (Q_1, m_1) : A$, then for any typing derivation π_1 of $\vdash (Q_1, m_1) : A$, there exists a typing derivation π_2 of $\vdash (Q_2, m_2) : A$ such that $[\![\pi_1]\!] = [\![\pi_2]\!]$.*

Finally, from the type safety properties, we can derive the following, making it possible to define the interpretation of a closed term of basic type.

▶ **Corollary 19.** *All the typing derivations of a closed term of basic type share the same interpretation.*

4 Conclusion

In this paper, we introduce the language Proto-Quipper-L which formalizes several features of Quipper (dynamic lifting, higher-order function, circuit composition, and branching) while treating the qubits linearly using the type system. On one hand we propose a type system and an operational semantics which explains the meaning of programs as a set of reduction rules. On the other hand, we propose a concrete categorical model of the language which is proven to be sound, meaning that the semantics is preserved over the operational semantics.

On one side, the model is closely related to models of intuitionistic linear logic. Diagrams are akin to proof nets: tensor nodes correspond to multiplicative connectives while boxes correspond to additive connectives. On the other side, they can be considered as an extension of diagrammatic languages for quantum processes [19].

Our concrete semantics makes it possible to describe a monad, following closely Quipper's operational semantics encoded in Haskell's type system. With this semantics we are able to answer an open question in the community: finding a categorical representation of dynamic lifting for a circuit-description language.

References

1 Linda Anticoli, Carla Piazza, Leonardo Taglialegne, and Paolo Zuliani. Towards quantum programs verification: from Quipper circuits to qpmc. In *International Conference on Reversible Computation*, pages 213–219. Springer, 2016.

2 P Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *International Workshop on Computer Science Logic*, pages 121–135. Springer, 1994.

3 Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin T. Vechev. Silq: a high-level quantum language with safe uncomputation and intuitive semantics. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 286–300. ACM, 2020. doi:10.1145/3385412.3386007.

4 Qingxiuxiong Dong, Marco Túlio Quintino, Akihito Soeda, and Mio Murao. Success-or-draw: A strategy allowing repeat-until-success in quantum computation. *Phys. Rev. Lett.*, 126:150504, April 2021. doi:10.1103/PhysRevLett.126.150504.

5 Peng Fu, Kohei Kishida, and Peter Selinger. Linear dependent type theory for quantum programming languages. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 440–453, 2020.

6 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

7 Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: A scalable quantum programming language. In Hans-Juergen Boehm and Cormac Flanagan, editors, *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'13*, pages 333–342. ACM, 2013. doi:10.1145/2491956.2462177.

8 Emmanuel Knill. Conventions for quantum pseudocode. Technical report, Los Alamos National Lab., NM (United States), 1996.

9 Bert Lindenhovius, Michael Mislove, and Vladimir Zamdzhiev. Enriching a linear/non-linear lambda calculus: A programming language for string diagrams. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 659–668. ACM, 2018.

10 Eugenio Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991.

11 Jennifer Paykin, Robert Rand, and Steve Zdancewic. QWIRE: a core language for quantum circuits. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL'17*, pages 846–858. ACM, 2017. doi:10.1145/3009837.3009894.

12 Robert Rand, Jennifer Paykin, and Steve Zdancewic. QWIRE practice: Formal verification of quantum circuits in coq. In Bob Coecke and Aleks Kissinger, editors, *Proceedings 14th International Conference on Quantum Physics and Logic, QPL 2017*, volume 266 of *Electronic Proceedings in Theoretical Computer Science*, pages 119–132, 2017. doi:10.4204/EPTCS.266.8.

13 Mathys Rennela and Sam Staton. Classical control, quantum circuits and linear logic in enriched category theory. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:30)2020.

14 Francisco Rios and Peter Selinger. A categorical model for a quantum circuit description language. In Bob Coecke and Aleks Kissinger, editors, *Proceedings 14th International Conference on Quantum Physics and Logic, QPL 2017*, volume 266 of *Electronic Proceedings in Theoretical Computer Science*, pages 164–178, 2018. doi:10.4204/EPTCS.266.11.

15 Neil J. Ross. *Algebraic and logical methods in quantum computation.* PhD thesis, Dalhousie University, 2015.

16 Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.

17 Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16(3):527–552, 2006.

18 Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture. *arXiv preprint*, 2016. arXiv:1608.03355.

19 Sam Staton. Algebraic effects, linearity, and quantum programming languages. *SIGPLAN Not.*, 50(1):395–406, January 2015. doi:10.1145/2775051.2676999.

20 Krysta M. Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling scalable quantum computing and development with a high-level domain-specific language. *arXiv preprint*, 2018. arXiv:1803.00652.

21 Benoît Valiron. *Semantics for a higher order functional programming language for quantum computation.* PhD thesis, University of Ottawa, 2008.

22 Dave Wecker and Krysta M. Svore. LIQUi$|\rangle$: A software design architecture and domain-specific language for quantum computing. *arXiv preprint*, 2014. arXiv:1402.4467.

23 William K. Wootters and Wojciech H. Zurek. A single quantum cannot be cloned. *Nature*, 299:802–803, October 1982. doi:10.1038/299802a0.

24 Bernhard Ömer. *Structured quantum programming.* PhD thesis, Technical University of Vienna, 2003.

A Operational semantics

A.1 Reduction

The reduction rules for Proto-Quipper-L are defined as follows.

Reduction rules for classical computation. The following rules always hold (b is tt or ff)

$$\begin{aligned}
(\epsilon(W),(\lambda x.M)V) &\to (\epsilon(W), M[V/x]) \\
(\epsilon(W), \textbf{let } \langle x,y\rangle = \langle V,U\rangle \textbf{ in } M) &\to (\epsilon(W), M[V/x,U/y]) \\
(\epsilon(W), \textbf{if } \mathrm{b} \textbf{ then } M_{\mathrm{tt}} \textbf{ else } M_{\mathrm{ff}}) &\to (\epsilon(W), M_b)
\end{aligned}$$

Reduction rules for circuit operations. Provided that **new** is an operator that creates free variables during the computation, meaning that these free variables do not appear in both classical and quantum contexts and that the term $\mathbf{new}(P)$ is a pattern of same shape as P made out of these new variables, we have

$$\frac{p = \mathbf{new}(P) \qquad W_p = \mathbf{supp}(p)}{(\epsilon(\emptyset), \mathrm{box}_P\ V) \to (\epsilon(\emptyset), (p, \epsilon(W_p), Vp))} \qquad \frac{\mathbf{shape}(p) = \mathbf{shape}(V) \qquad \sigma = \mathbf{bind}(p, V)}{(\epsilon(\mathbf{FV}(V)), (\mathrm{unbox}(p,\ Q,\ u))V) \to (\sigma(Q), \sigma(u))}$$

Structural reduction rule for quantum channel constant. Provided that $(Q,m) \to (Q',m')$, we have $(\epsilon(\emptyset),(p,Q,m)) \to (\epsilon(\emptyset),(p,Q',m'))$.

Structural reduction rules for empty quantum channel. Provided that $(\epsilon(W_M),\ M) \to (Q,\ m)$, that $\mathbf{all}(Q) \cap W_N = \emptyset$ and that $\mathbf{all}(Q) \cap W_V = \emptyset$, we have

$$\begin{aligned}
(\epsilon(W_M \cup W_N),\ MN) &\to (\mathbf{extend}(Q, W_N),\ mN) \\
(\epsilon(W \cup W_V),\ VM) &\to (\mathbf{extend}(Q, W_V),\ Vm) \\
(\epsilon(W_M \cup W_N),\ \langle M,N\rangle) &\to (\mathbf{extend}(Q, W_N),\ \langle m,N\rangle) \\
(\epsilon(W_M \cup W_V),\ \langle V,M\rangle) &\to (\mathbf{extend}(Q, W_V),\ \langle V,m\rangle) \\
(\epsilon(W_M \cup W_N),\ \textbf{if } M \textbf{ then } M_a \textbf{ else } M_b)) &\to (\mathbf{extend}(Q, W_N),\ \textbf{if } m \textbf{ then } M_a \textbf{ else } M_b) \\
(\epsilon(W_M),\ \textbf{let } \langle x,y\rangle = M \textbf{ in } N) &\to (\mathbf{extend}(Q, W_N),\ \textbf{let } \langle x,y\rangle = m \textbf{ in } N)
\end{aligned}$$

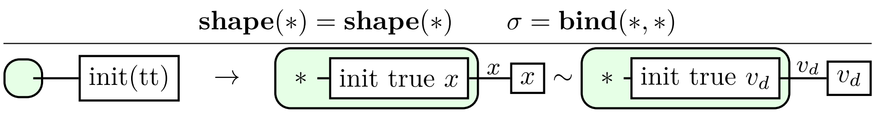
We use syntactic sugar combining terms and branching terms, as in mM. It corresponds to the term constructor applied to every leafs of m, for instance: for $m = [[N_1,N_2],N_3]$, $[[N_1,N_2],N_3]M := [[N_1M, N_2M], N_3M]$.

Structural reduction rules for non-empty quantum channel. Assume that $(Q_1,\ m_a) \to (Q_3,\ m_c)$ and $(Q_2,\ m_b) \to (Q_4,\ m_d)$. Then

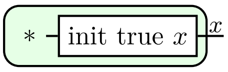
$$\begin{aligned}
((\mathrm{meas}\ w\ Q_1\ Q_2),\ [m_a, m_b]) &\to ((\mathrm{meas}\ w\ Q_3\ Q_4),\ [m_c, m_d]) \\
((\mathrm{meas}\ w\ Q_1\ Q_2),\ [m_a, v]) &\to ((\mathrm{meas}\ w\ Q_3\ Q_2),\ [m_c, v]) \\
((\mathrm{meas}\ w\ Q_1\ Q_2),\ [v, m_b]) &\to ((\mathrm{meas}\ w\ Q_1\ Q_4),\ [v, m_d]) \\
(U(W)\ Q_1,\ m_a) &\to (U(W)Q_3,\ m_c) \\
(\mathrm{init}\ b\ w\ Q_1,\ m_a) &\to (\mathrm{init}\ b\ w\ Q_3,\ m_c) \\
(\mathrm{free}\ w\ Q_1,\ m_a) &\to (\mathrm{free}\ w\ Q_3,\ m_c)
\end{aligned}$$

A.2 Derivation of the example of Example 6

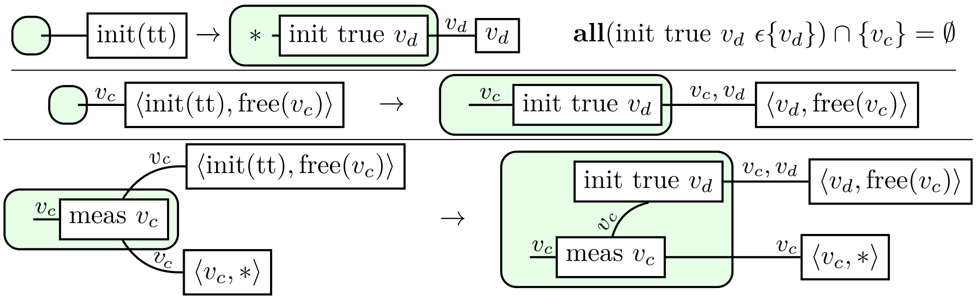
Let us explain how the tree expands as the computation progresses for example 6. First, we show that $((\epsilon\{\}, \mathrm{init}(\mathrm{tt})) \to ((\mathrm{init\ true}\ x\ \epsilon\{x\}, x)$ as follows.



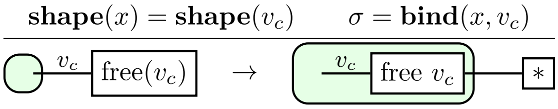
where we let

init(tt) = unbox (∗,  , x) (∗).

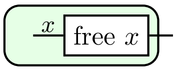
Then we can show the following reduction:



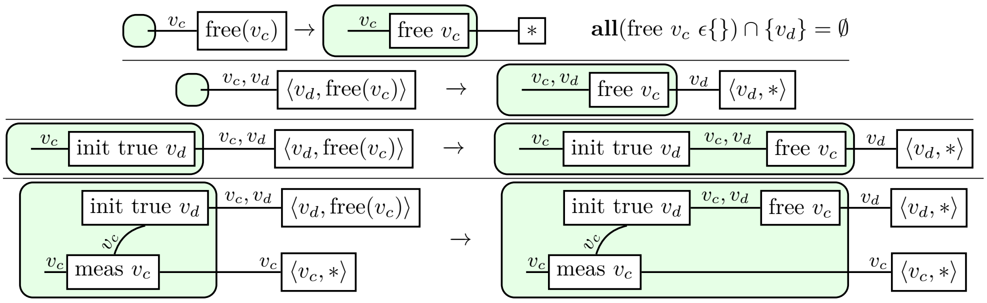
Next, we show the last reduction step of the example.



Recall that

free = unbox (x,  ∗ , ∗) .

Then we can show the following reduction:



B Categorical semantics

B.1 Equivalence of diagrams

Complete list of the equivalence rules that are used to construct the categorical model is shown in Figure 4.

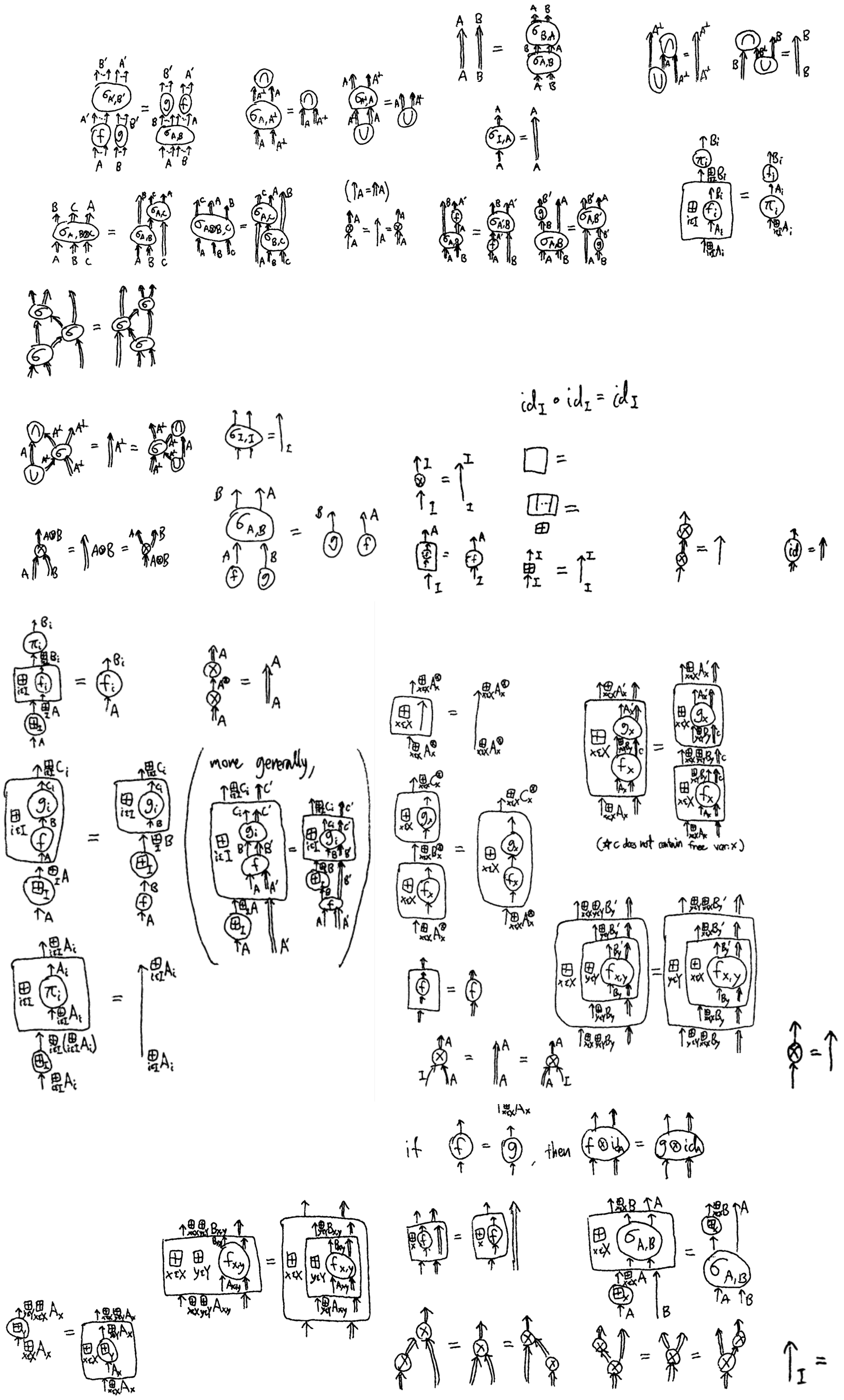
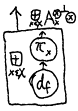
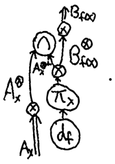
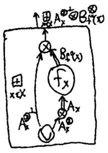


Figure 4 Equivalence relation of diagrams.

Table 6 Definition of isomorphism between $b(A \multimap_{\overline{\overline{M}}} B)$ and $\overline{\overline{M}}(A, B)$.

$\mathrm{iso}_{\rightarrow} : b(A \multimap_{\overline{\overline{M}}} B) \to \overline{\overline{M}}(A, B)$:	$\mathrm{iso}_{\leftarrow} : \overline{\overline{M}}(A, B) \to b(A \multimap_{\overline{\overline{M}}} B)$:
Given (f, d_f), which is $\Bigg(f,$  $\Bigg)$, let $f_0 = f$ and $f_x =$ .	Given $(f_0, (f_x)_X)$, let $f = f_0$ and $d_f =$ 

B.2 Interpretation of type system

For a typing context $\Gamma = x_1 : A_1, \dots, x_k : A_k$, assuming the variables are ordered by some linear order, $[\![\Gamma]\!] = [\![A_1]\!] \otimes \dots \otimes [\![A_k]\!]$. Next, the branching typing context is interpreted as the coproduct of the objects assigned to the smaller branching typing contexts, namely: $[\![\gamma_1, \gamma_2]\!] = [\![\gamma_1]\!] + [\![\gamma_2]\!]$. Lastly, we interpret the typing derivation as a morphism in $\overline{\overline{M}}_F$.

B.2.1 Quantum channel types, Box and Unbox

As in [14], we interpret the quantum channel types $\mathbf{QChan}(A, B)$ as an object $p(\overline{\overline{M}}_F(A, B)) = (\overline{\overline{M}}_F(A, B), (I))$ in $\overline{\overline{M}}_F$ and $\overline{\overline{M}}$. Note that the object is a parameter object as in [14], which means that the object has the form of $(X, (I)_X)$ for some X. When we define the quantum channel types $\mathbf{QChan}(A, B)$ as a parameter object, box and unbox can be interpreted based on an isomorphism between the set $b(A \multimap_{\overline{\overline{M}}} B)$ and $\overline{\overline{M}}(A, B)$. In specific, we can define an isomorphism as in Table 6.

Given the isomorphism, we can define the morphisms for box and unbox as morphisms in $\overline{\overline{M}}$ as follows:

$$\begin{aligned} \mathrm{unbox} &= p(\overline{\overline{M}}(A, F(B))) \xrightarrow{p(\mathrm{iso}_{\rightarrow})} (p \circ b)(A \multimap_{\overline{\overline{M}}} F(B)) \xrightarrow{\epsilon(A \multimap_{\overline{\overline{M}}} F(B))} (A \multimap_{\overline{\overline{M}}} F(B)) \\ \mathrm{box} &= (p \circ b)(A \multimap_{\overline{\overline{M}}} F(B)) \xrightarrow{p(\mathrm{iso}_{\leftarrow})} p(\overline{\overline{M}}(A, F(B))) \\ &\qquad \xrightarrow{p(\eta(\overline{\overline{M}}(A, F(B))))} (p \circ b \circ p)(\overline{\overline{M}}(A, F(B))). \end{aligned}$$

where ϵ refers to the counit from the comonad !.

B.2.2 Quantum channel constants

We define a natural transformations called bif and merge for the measurement as in Table 7.

A quantum channel Q is interpreted as a morphism $[\![\mathbf{in}(Q)]\!] \to_{\overline{\overline{M}}_F} [\![\mathbf{out}(Q)]\!]$, where

$$[\![\mathbf{in}(Q)]\!] = (\{\emptyset\}, ([q]^{\otimes |\mathbf{in}(Q)|}))$$

$$[\![\mathbf{out}(Q)]\!] = \begin{cases} (\{\emptyset\}, ([q]^{\otimes |V|})) & \text{if } \mathbf{out}(Q) \text{ is a set } V \\ [\![o_1]\!] + [\![o_2]\!] & \text{if } \mathbf{out}(Q) = [o_1, o_2] \end{cases}$$

Table 7 Definition of bif and merge.

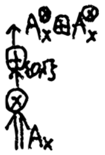
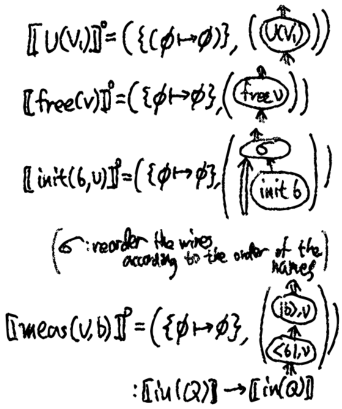
$\mathrm{bif}(A) : A \to_{\overline{\overline{M}}_F} A + A$	$\mathrm{merge}(A, B) : F(A) + F(B) \to_{\overline{\overline{M}}_F} A + B$
For an object $A = (X, (A_x))$, we let $\mathrm{bif}(X, (A_x)) = \begin{pmatrix} \{x \mapsto [(0, x), (1, x)]\}, \\ (f_x : A_x \to A_x^{\otimes} \boxplus A_x^{\otimes}) \end{pmatrix}$ where $f_x =$  $A_x^{\otimes} \boxplus A_x^{\otimes}$ $\{0,1\}$ A_x .	For objects A, B, we let $\mathrm{merge}(A, B) = (\{$ $(0, [x_1, \ldots, x_k]) \mapsto [(0, x_1), \ldots, (0, x_k)],$ $(1, [y_1, \ldots, y_n]) \mapsto [(1, y_1), \ldots, (1, y_n)]\},$ $(\mathbf{id}_{[\boxplus_{x \in l} A_x^{\otimes}]})_{l:\mathbf{mset}(X)}$ $++(\mathbf{id}_{[\boxplus_{y \in l} B_y^{\otimes}]})_{l:\mathbf{mset}(Y)})$
It satisfies the following commute diagram for naturality: $\begin{tikzcd} A \arrow[r, "\mathrm{bif}(A)"] \arrow[d, "f"] & F(A + A) \arrow[d, "F(f+f)"] \\ B \arrow[r, "\mathrm{bif}(B)"] & F(B + B) \end{tikzcd}$	It satisfies the following commute diagram for naturality: $\begin{tikzcd} F(A) + F(B) \arrow[r, "\mathrm{merge}(A,B)"] \arrow[d, "F(f)+F(g)"] & F(A + B) \arrow[d, "F(f+g)"] \\ F(A') + F(B') \arrow[r, "\mathrm{merge}(A',B')"] & F(A' + B') \end{tikzcd}$

Table 8 Interpretation of quantum channel constants.

$[\![\epsilon(V)]\!] = \eta(\{\emptyset\}, ([q]^{\otimes\lvert V\rvert}))$ $[\![U(V_1)\ Q]\!] = [\![Q]\!] \circ [\![U(V_1)]\!]^0$ $[\![\mathrm{free}\ v\ Q]\!] = [\![Q]\!] \circ [\![\mathrm{free}(V)]\!]^0$ $[\![\mathrm{init}\ b\ v\ Q]\!] = [\![Q]\!] \circ [\![\mathrm{init}(b, v)]\!]^0$ $[\![\mathrm{meas}\ v\ Q_1\ Q_2]\!] =$ $\begin{tikzcd} {[\![\mathbf{in}(\mathrm{meas}\ v\ Q_1\ Q_2)]\!]} \arrow[d, "\mathrm{bif}; F\begin{pmatrix} {[\![Q_1]\!]} \circ {[\![\mathrm{meas}(v,0)]\!]^0} \\ +{[\![Q_2]\!]} \circ {[\![\mathrm{meas}(v,1)]\!]^0} \end{pmatrix}"] \\ F(F[\![\mathbf{out}(Q_1)]\!] + F[\![\mathbf{out}(Q_2)]\!]) \arrow[d, "F(\mathrm{merge}); \mu"] \\ F([\![\mathbf{out}(Q_1)]\!] + [\![\mathbf{out}(Q_2)]\!]) \end{tikzcd}$	where  $[\![U(V_1)]\!]^0 = (\{(\phi \mapsto \phi)\}, ($ $U(V_1)$ $))$ $[\![\mathrm{free}(v)]\!]^0 = (\{\phi \mapsto \phi\}, ($ free v $))$ $[\![\mathrm{init}(b, v)]\!]^0 = (\{\phi \mapsto \phi\}, ($ σ init b $))$ (σ : reorder the wires according to the order of the names) $[\![\mathrm{meas}(v, b)]\!]^0 = (\{\phi \mapsto \phi\}, ($ $\lvert b\rangle, v$ $\langle b\rvert, v$ $))$ $: [\![\mathrm{in}(Q)]\!] \to [\![\mathrm{in}(Q)]\!]$

The interpretation of quantum channel is defined inductively as in Table 8 where μ represents the multiplication of the monad F. In addition, the elementary nodes in Table 8–$(U(V_1))$, (free v), $(\mid b\rangle,\ v)$ and $(\langle b \mid,\ v)$–refers to the unitary gate node Ⓤ (which is either 1 or 2-qubits gate) applied to wires V_1, (tr) node applied to wire v, and $(\langle b\rvert)$ and $(\lvert b\rangle)$ nodes applied to wire v, respectively.

Linear-Time Temporal Logic with Team Semantics: Expressivity and Complexity

Jonni Virtema
Institute for Theoretical Computer Science, Leibniz Universität Hannover, Germany
Department of Computer Science, University of Sheffield, UK

Jana Hofmann
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Juha Kontinen
Department of Mathematics and Statistics, University of Helsinki, Finland

Fan Yang
Department of Mathematics and Statistics, University of Helsinki, Finland

Abstract

We study the expressivity and complexity of model checking of linear temporal logic with team semantics (TeamLTL). TeamLTL, despite being a purely modal logic, is capable of defining hyperproperties, i.e., properties which relate multiple execution traces. TeamLTL has been introduced quite recently and only few results are known regarding its expressivity and its model checking problem. We relate the expressivity of TeamLTL to logics for hyperproperties obtained by extending LTL with trace and propositional quantifiers (HyperLTL and HyperQPTL). By doing so, we obtain a number of model checking results for TeamLTL and identify its undecidability frontier. In particular, we show decidability of model checking of the so-called *left-flat* fragment of any downward closed TeamLTL-extension. Moreover, we establish that the model checking problem of TeamLTL with Boolean disjunction and inclusion atoms is undecidable.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Modal and temporal logics

Keywords and phrases Linear temporal logic, Hyperproperties, Model Checking, Expressivity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.52

Related Version *Full Version*: `https://arxiv.org/abs/2010.03311`

Funding *Jana Hofmann and Bernd Finkbeiner:* Partially supported by the German Research Foundation (DFG) as part of the Collaborative Research Center "Foundations of Perspicuous Software Systems" (TRR 248, 389792660) and by the European Research Council (ERC) Grant OSARES (No. 683300).
Jonni Virtema: Supported by the DFG grant VI 1045/1-1.
Juha Kontinen: Supported by grant 308712 of the Academy of Finland.
Fan Yang: Supported by grants 330525 and 308712 of Academy of Finland, and Research Funds of University of Helsinki.

1 Introduction

Linear-time temporal logic (LTL) is one of the most prominent logics for the specification and verification of reactive and concurrent systems. Practical model checking tools like SPIN, NuSMV, and many others ([29, 6, 11]) automatically verify whether a given computer system, such as a hardware circuit or a communication protocol, is correct with respect to

41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021).
Editors: Mikołaj Bojańczyk and Chandra Chekuri; Article No. 52; pp. 52:1–52:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

its LTL specification. The basic principle, as introduced in 1977 by Amir Pnueli [39], is to specify the correctness of a program as a set of infinite sequences, called *traces*, which define the acceptable executions of the system.

Hyperproperties, i.e., properties which relate multiple execution traces, cannot be specified in LTL. Such properties are of prime interest in information flow security, where dependencies between the secret inputs and the publicly observable outputs of a system are considered potential security violations. Commonly known properties of that type are noninterference [41, 37] or observational determinism [49]. In other settings, relations between traces are explicitly desirable: robustness properties, for example, state that similar inputs lead to similar outputs. Hyperproperties are not limited to the area of information flow control. E.g., distributivity and other system properties like fault tolerance can be expressed as hyperproperties [17].

The main approach to specify hyperproperties has been to extend temporal logics like LTL, CTL, and QPTL with explicit trace and path quantification, resulting in logics like HyperLTL [7], HyperCTL* [7], and HyperQPTL [40, 9]. Most frequently used is HyperLTL, which can express noninterference as follows: $\forall\pi.\forall\pi'.\Box(\bigwedge_{i\in I} i_\pi \leftrightarrow i_{\pi'}) \to \Box(\bigwedge_{o\in O} o_\pi \leftrightarrow o_{\pi'})$. The formula states that any two traces which globally agree on the value of the public inputs I also globally agree on the public outputs O. Consequently, the value of secret inputs cannot affect the value of the publicly observable outputs.

It is not clear, however, whether quantification over traces is the best way to express hyperproperties. The success of LTL over first-order logics for the specification of linear-time properties stems from the fact that its modal operators replace explicit quantification of points in time. This allows for a much more concise and readable formulation of the same property. The natural question to ask is whether a purely modal logic for hyperproperties would have similar advantages. A candidate for such a logic is LTL with *team semantics* [34]. Under team semantics, LTL expresses hyperproperties without explicit references to traces. Instead, each subformula is evaluated with respect to a set of traces, called a *team*. Temporal operators advance time on all traces of the current team. Using the split operator $\vee$, teams can be split during the evaluation of a formula, which enables us to express properties of subsets of traces.

As an example, consider the property that there is a point in time, common for all traces, after which a certain event a does not occur any more. We need a propositional and a trace quantifier to express such a property in HyperQPTL (it is not expressible in HyperLTL). The formula $\exists p.\forall\pi.\Diamond p \wedge \Box(p \to \Box\neg a_\pi)$ states that there is a p-sequence $s \in (2^{\{p\}})^\omega$ such that p is set at least once, and if $p \in s[i]$, then a is not set on all traces π on all points in time starting from i. The same property can be expressed in TeamLTL without any quantification simply as $\Diamond\Box\neg a$. The formula exploits the synchronous semantics of TeamLTL by stating that there is a point such that for all future points all traces have a not set. As a second example, consider the case that an unknown input determines the behaviour of the system. Depending on the input, its execution traces either agree on a or on b. We can express the property in HyperLTL with three trace quantifiers: $\exists\pi_1.\exists\pi_2.\forall\pi.\Box(a_{\pi_1} \leftrightarrow a_\pi) \vee \Box(b_{\pi_2} \leftrightarrow b_\pi)$. In TeamLTL, the same property can be simply expressed as $\Box(a \ovee \neg a) \vee \Box(b \ovee \neg b)$. The Boolean or operator $\ovee$ expresses that in the current team, either the left side holds on all traces or the right side does.

The use of the $\ovee$ operator reveals another strength of TeamLTL: its modularity. The research on team semantics (see related work section) has a rich tradition of studying extensions of team logics with new atomic statements and operators. They constitute a well-defined way to increase a logic's expressiveness in a step-by-step manner. Besides $\ovee$, examples are Boolean negation $\sim$, the inclusion atom $\subseteq$, and universal subteam quantifiers

A and $\overset{1}{\mathsf{A}}$. Inclusion atoms have been found to be fascinating for their ability to express recursion in the first-order setting; the expressivity of FO($\subseteq$) coincides with greatest fixed point logic and hence PTIME [20]. In turn, all LTL-definable properties can be expressed by TeamLTL-formulae of the form $\overset{1}{\mathsf{A}}\varphi$. With the introduction of generalised atoms, TeamLTL even permits custom extensions. Possibly most interesting in the context of hyperproperties are dependence atoms. A dependence atom $\mathrm{dep}(x_1, \dots, x_n)$ is satisfied by a team X if any two assignments assigning the same values to the variables $x_1, \dots, x_{n-1}$ also assign the same value to x_n. For example, the TeamLTL formula $(\Box\,\mathrm{dep}(i_1, i_2, o)) \vee (\Box\,\mathrm{dep}(i_2, i_3, o))$ states that the executions of the system can be decomposed into two parts; in the first part, the output o is determined by the inputs i_1 and i_2, and in the second part, o is determined by the inputs i_2 and i_3.

Temporal team logics constitute a new, fundamentally different approach to specify hyperproperties. While HyperLTL and other quantification-based hyperlogics have been studied extensively (see section on related work), only few results are known about the expressive power and complexity of TeamLTL and its variants. In particular, we know very little about how the expressivity of the two approaches compares. What is known is that HyperLTL and TeamLTL are incomparable in expressivity [34] and that the model checking problem of TeamLTL without splitjunctions $\vee$ (what makes the logic significantly weaker) is in PSPACE [34]. On the other hand, it was recently shown that the complexity of satisfiability and model checking of TeamLTL with Boolean negation $\sim$ is equivalent to the decision problem of third-order arithmetic [35] and hence highly undecidable.

Our contribution. We advance the understanding of team-based logics for hyperproperties by exploring the relative expressivity of TeamLTL and temporal hyperlogics like HyperLTL, as well as the decidability frontier of the model checking problem of TeamLTL. Our expressivity and model checking results are summarized in Table 1 and Table 2. We identify expressively complete extensions of TeamLTL (displayed on the left of Table 1) that can express all (all downward closed, resp.) Boolean relations on LTL-properties of teams, and present several translations from team logics to hyperlogics. We begin by approaching the decidability frontier of TeamLTL from above, and tackle a question posed in [35]: *Does some sensible restriction to the use of Boolean negation in* TeamLTL($\sim$) *yield a decidable logic?* We show that already a very restricted access to $\sim$ leads to high undecidability, whereas already the use of inclusion atoms $\subseteq$ together with Boolean disjunctions $\varovee$ suffices for undecidable model checking. Furthermore, we establish that these complexity results transfer to the satisfiability problem of the related logics. Next, regarding the expressivity of TeamLTL, we show that its extensions with all (all downward closed, resp.) atomic LTL-properties of teams translate to simple fragments of HyperQPTL$^+$. Consequently, known decidability results for quantification-based hyperlogics enable us to approach the decidability frontier of TeamLTL extensions from below. We establish an efficient translation from the so-called *k-coherent fragment* of TeamLTL($\sim$) to the universal fragment of HyperLTL (for which model checking is PSPACE-complete [19]) and thereby obtain EXPSPACE model checking for the fragment. Finally, we show that the so-called *left-flat* fragment of TeamLTL($\varovee$, $\overset{1}{\mathsf{A}}$) enjoys decidable model checking via a translation to $\overset{u}{\exists}{}^*_p\forall^*_\pi$HyperQPTL.

Related work. The development of team semantics began with the introduction of Dependence Logic [45], which adds the concept of functional dependence to first-order logic by means of new atomic dependence formulae. During the past decade, team semantics has been generalised to propositional [48], modal [46], temporal [33], and probabilistic [13]

Table 1 Expressivity results. The logics TeamLTL($\ovee, \sim\bot, \overset{1}{\mathsf{A}}$) and TeamLTL($\overset{1}{\mathsf{A}}, \ovee$) can express *all*/*all downward closed* atomic LTL-properties of teams (see the discussion at the end of Section 2). † holds since TeamLTL($\overset{1}{\mathsf{A}}, \ovee$) is downward closed.

		(assuming left-flatness)
TeamLTL($\ovee, \overset{1}{\mathsf{A}}$)	$\overset{\text{Thm.14}}{\leq}$	$\overset{u}{\exists}{}^*_q \forall_\pi$HyperQPTL
	$\overset{\text{Thm.6}}{\leq}$	$\exists_p \overset{u}{Q}{}^*_p \forall_\pi$HyperQPTL$^+$
$\wedge^\dagger$		
TeamLTL($\ovee, \sim\bot, \overset{1}{\mathsf{A}}$)	$\overset{\text{Thm.6}}{\leq}$	$\exists_p \overset{u}{Q}{}^*_p \exists^*_\pi \forall_\pi$HyperQPTL$^+$
$\shortmid\wedge$ [35]		(assuming k-coherence)
TeamLTL($\sim$)	$\overset{\text{Thm.9}}{\leq}$	$\forall^k$HyperLTL

Table 2 Complexity results.

Logic	Model Checking Result
TeamLTL without $\vee$	in PSPACE [34]
k-coherent TeamLTL($\sim$)	in EXPSPACE [Thm. 10]
left-flat TeamLTL($\ovee, \overset{1}{\mathsf{A}}$)	in EXPSPACE [Thm. 15]
TeamLTL($\subseteq, \ovee$)	Σ^0_1-hard [Thm. 2]
TeamLTL($\subseteq, \ovee, \mathsf{A}$)	Σ^1_1-hard [Thm. 3]
TeamLTL($\sim$)	complete for third-order arithmetic [35]

frameworks, and fascinating connections to fields such as database theory [23], statistics [12], real valued computation [24], and quantum information theory [30] has been identified. In the modal team semantics setting, model checking and satisfiability problems have been shown to be decidable, see [26, page 627] for an overview of the complexity landscape. Expressivity and definability of related logics is also well understood, see, e.g. [27, 32, 42]. The study of temporal logics with team semantics, was initiated in [33], where team semantics for computational tree logic CTL was given. The idea to develop team-based logics for hyperproperties was coined in [34], where TeamLTL was first introduced and shown incomparable to HyperLTL. The interest on logics for hyperproperties, so-called hyperlogics, was sparked by the introduction of HyperLTL and HyperCTL* [7]. Many temporal logics have since been extended with trace and path quantification to obtain various hyperlogics, e.g., to express asynchronous hyperproperties [22, 4], hyperproperties on finite traces [21], probabilistic hyperproperties [1], or timed hyperproperties [28]. Model checking HyperLTL and the strictly more expressive HyperQPTL is decidable, though k-EXPSPACE-complete, where k is the number of quantifier alternations in the formula [19, 40]. Model checking HyperQPTL$^+$, on the other hand, is undecidable [16]. The expressivity of HyperLTL, HyperCTL*, and HyperQPTL has been compared to first-order and second-order hyperlogics resulting in a hierarchy of hyperlogics [9]. Beyond model checking and expressivity questions, especially HyperLTL has been studied extensively. This includes its satisfiability [15, 36], runtime monitoring [18, 2] and enforcement problems [10], as well as synthesis [17].

2 Basics of TeamLTL

Let us start by recalling the syntax of LTL from the literature. Fix a set AP of *atomic propositions*. The set of formulae of LTL (over AP) is generated by the following grammar:

$$\varphi ::= p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi \mid \varphi \mathcal{W} \varphi, \qquad \text{where } p \in \mathrm{AP}.$$

We adopt, as is common in studies on team logics, the convention that formulae are given in negation normal form. The logical constants $\top, \bot$ and connectives $\rightarrow, \leftrightarrow$ are defined as usual (e.g., $\bot := p \wedge \neg p$ and $\top := p \vee \neg p$), and $\Diamond \varphi := \top \mathcal{U} \varphi$ and $\Box \varphi := \varphi \mathcal{W} \bot$.

A *trace* t over AP is an infinite sequence from $(2^{\mathrm{AP}})^\omega$. For a natural number $i \in \mathbb{N}$, we denote by $t[i]$ the ith element of t and by $t[i, \infty]$ the postfix $(t[j])_{j \geq i}$ of t. The satisfaction relation $(t, i) \models \varphi$, for LTL formulae φ, is defined as usual, see e.g., [38]. We use $[\![\varphi]\!]_{(t,i)} \in \{0, 1\}$ to denote the truth value of φ on (t, i). A *(temporal) team* is a pair (T, i) consisting a set of traces $T \subseteq (2^{\mathrm{AP}})^\omega$ and a natural number $i \in \mathbb{N}$ representing the time step. We write $T[i]$ and $T[i, \infty]$ to denote the sets $\{t[i] \mid t \in T\}$ and $\{t[i, \infty] \mid t \in T\}$, respectively.

Let us next introduce the logic LTL interpreted with team semantics (denoted TeamLTL). TeamLTL was first studied in [34], where it was called LTL with *synchronous* team semantics. The satisfaction relation $(T, i) \models \varphi$ for TeamLTL is defined as follows:

$$\begin{array}{lll}
(T,i) \models p & \text{iff} & \forall t \in T : p \in t[i] \\
(T,i) \models \neg p & \text{iff} & \forall t \in T : p \notin t[i] \\
(T,i) \models \varphi \wedge \psi & \text{iff} & (T,i) \models \varphi \text{ and } (T,i) \models \psi \\
(T,i) \models \bigcirc \varphi & \text{iff} & (T,i+1) \models \varphi \\
(T,i) \models \varphi \wedge \psi & \text{iff} & (T,i) \models \varphi \text{ and } (T,i) \models \psi \\
(T,i) \models \bigcirc \varphi & \text{iff} & (T,i+1) \models \varphi \\
(T,i) \models \varphi \vee \psi & \text{iff} & (T_1,i) \models \varphi \text{ and } (T_2,i) \models \psi, \text{ for some } T_1, T_2 \text{ s.t. } T_1 \cup T_2 = T \\
(T,i) \models \varphi \mathcal{U} \psi & \text{iff} & \exists k \geq i \text{ such that } (T,k) \models \psi \text{ and } \forall m : i \leq m < k \Rightarrow (T,m) \models \varphi \\
(T,i) \models \varphi \mathcal{W} \psi & \text{iff} & \forall k \geq i : (T,k) \models \varphi \text{ or } \exists m \text{ such that } i \leq m \leq k \text{ and } (T,m) \models \psi
\end{array}$$

Note that $(T, i) \models \bot$ iff $T = \emptyset$. Two formulae φ and ψ are *equivalent* (written $\varphi \equiv \psi$), if the equivalence $(T, i) \models \varphi$ iff $(T, i) \models \psi$ holds for every (T, i). We say that a logic $\mathcal{L}_2$ is *at least as expressive* as a logic $\mathcal{L}_1$ (written $\mathcal{L}_1 \leq \mathcal{L}_2$) if for every $\mathcal{L}_1$-formula φ, there exists an $\mathcal{L}_2$-formula ψ such that $\varphi \equiv \psi$. We write $\mathcal{L}_1 \equiv \mathcal{L}_2$ if both $\mathcal{L}_1 \leq \mathcal{L}_2$ and $\mathcal{L}_2 \leq \mathcal{L}_1$ hold. The following are important semantic properties of formulae from the team semantics literature:

(Downward closure) If $(T, i) \models \varphi$ and $S \subseteq T$, then $(S, i) \models \varphi$.
(Empty team property) $(\emptyset, i) \models \varphi$.
(Flatness) $(T, i) \models \varphi$ iff $(\{t\}, i) \models \varphi$ for all $t \in T$.
(Singleton equivalence) $(\{t\}, i) \models \varphi$ iff $(t, i) \models \varphi$.

A logic has one of the above properties if every formula of the logic has the property. TeamLTL satisfies downward closure, singleton equivalence, and the empty team property [34]. However, it does not satisfy flatness; for instance, the formula $\Diamond p$ is not flat.

The power of team semantics comes with the ability to enrich logics with novel atomic statements describing properties of teams. We thereby easily get a hierarchy of team logics of different expressiveness. The most prominent examples of such atoms are *dependence atoms* $\mathrm{dep}(\varphi_1, \ldots, \varphi_n, \psi)$ and *inclusion atoms* $\varphi_1, \ldots, \varphi_n \subseteq \psi_1, \ldots, \psi_n$, with $\varphi_1, \ldots, \varphi_n, \psi, \psi_1, \ldots, \psi_n$ being LTL-formulae. The dependence atom states that the truth value of ψ is functionally determined by that of $\varphi_1, \ldots, \varphi_n$. The inclusion atom states that each value combination of $\varphi_1, \ldots, \varphi_n$ must also occur as a value combination for $\psi_1, \ldots, \psi_n$. Their formal semantics is defined as:

$$(T,i) \models \mathrm{dep}(\varphi_1, \ldots, \varphi_n, \psi) \text{ iff } \forall t, t' \in T : \Big(\bigwedge_{1 \leq j \leq n} [\![\varphi_j]\!]_{(t,i)} = [\![\varphi_j]\!]_{(t',i)} \Big) \Rightarrow [\![\psi]\!]_{(t,i)} = [\![\psi]\!]_{(t',i)}$$

$$(T,i) \models \varphi_1, \ldots, \varphi_n \subseteq \psi_1, \ldots, \psi_n \quad \text{iff} \quad \forall t \in T \, \exists t' \in T : \bigwedge_{1 \leq j \leq n} [\![\varphi_j]\!]_{(t,i)} = [\![\psi_j]\!]_{(t',i)}$$

As an example, let $o_1, \ldots, o_n$ be some observable outputs and s be a secret. The atom $(o_1, \ldots, o_n, s) \subseteq (o_1, \ldots, o_n, \neg s)$ expresses a form of *non-inference* by stating that an observer cannot infer the current value of the secret from the outputs. We also consider other connectives known in the team semantics literature: *Boolean disjunction* $\varovee$, *Boolean negation* $\sim$, and *universal subteam quantifiers* A and $\overset{1}{\mathsf{A}}$, with their semantics defined as:

$$\begin{array}{lll} (T,i) \models \varphi \varovee \psi & \text{iff} & (T,i) \models \varphi \text{ or } (T,i) \models \psi \\ (T,i) \models \sim \varphi & \text{iff} & (T,i) \not\models \varphi \\ (T,i) \models \mathsf{A}\varphi & \text{iff} & \forall S \subseteq T : (S,i) \models \varphi \\ (T,i) \models \overset{1}{\mathsf{A}}\varphi & \text{iff} & \forall t \in T : (\{t\},i) \models \varphi \end{array}$$

If $\mathcal{A}$ is a collection of atoms and connectives, we let TeamLTL($\mathcal{A}$) denote the extension of TeamLTL with the atoms and connectives in $\mathcal{A}$. For any atom or connective $\circ$, we write simply TeamLTL($\mathcal{A}, \circ$) instead of TeamLTL($\mathcal{A} \cup \{\circ\}$).

TeamLTL($\sim$) is a very expressive logic; all of the above connectives and atoms, as well as many others, have been shown to be definable in TeamLTL($\sim$) [25, 35]. To systematically explore less expressive variants of TeamLTL, we introduce two representative logics of different expressiveness, namely TeamLTL($\varovee, \overset{1}{\mathsf{A}}$) and TeamLTL($\varovee, \sim\bot, \overset{1}{\mathsf{A}}$). The expression $\sim\bot$ can be used to enforce non-emptiness of a team. What makes these logics good representatives is their semantic property that they can express a general class of Boolean relations. Let B be a set of n-ary Boolean relations. We define the property $[\varphi_1, \ldots, \varphi_n]_B$ for an n-tuple $(\varphi_1, \ldots, \varphi_n)$ of LTL-formulae:

$$(T,i) \models [\varphi_1, \ldots, \varphi_n]_B \quad \text{iff} \quad \{([\![\varphi_1]\!]_{(t,i)}, \ldots, [\![\varphi_n]\!]_{(t,i)}) \mid t \in T\} \in B.$$

The logic TeamLTL($\varovee, \sim\bot, \overset{1}{\mathsf{A}}$) is expressively complete with respect to all $[\varphi_1, \ldots, \varphi_n]_B$. That is, for every set of Boolean relations B and LTL-formulae $\varphi_1, \ldots, \varphi_n$, the property $[\varphi_1, \ldots, \varphi_n]_B$ is expressible in TeamLTL($\varovee, \sim\bot, \overset{1}{\mathsf{A}}$). Furthermore, TeamLTL($\varovee, \overset{1}{\mathsf{A}}$) can express all downward closed ($S_1 \in B$ & $S_2 \subseteq S_1$ imply $S_2 \in B$) B. These results are reformulated and proved using so-called *generalised atoms* in the extended version of this paper [47]. Note that, e.g., k-ary inclusion and dependence atoms can be defined using suitable Boolean relations B. Indeed it follows that, from the expressivity point-of-view, TeamLTL($\varovee, \overset{1}{\mathsf{A}}$) and TeamLTL($\varovee, \sim\bot, \overset{1}{\mathsf{A}}$) subsume all extensions of TeamLTL with downward closed (resp. all) *atomic notions of dependence*, i.e., atoms which state some sort of functional (in)dependence, like the dependence atom (which is downward closed) or the inclusion atom (which is not).

3 Undecidable Extensions of TeamLTL

In [35], Lück established that the model checking problem for TeamLTL($\sim$) is highly undecidable. The proof heavily utilises the interplay between Boolean negation $\sim$ and disjunction $\vee$; it was left as an open problem whether some sensible restrictions on the use of the Boolean negation would lead toward discovering decidable logics. We show that, on the contrary, the decidability bounds are much tighter. Already TeamLTL$(\subseteq, \varovee)$ (which is subsumed by TeamLTL$(\varovee, \sim\bot, \overset{1}{\mathsf{A}})$) is undecidable, and already very restricted access to $\sim$ (namely, a single use of the A quantifier) leads to high undecidability.

We define the model checking problem based on Kripke structures $\mathrm{K} = (W, R, \eta, w_0)$, where W is a finite set of states, $R \subseteq W^2$ the transition relation, $\eta\colon W \to 2^{\mathrm{AP}}$ a labelling function, and $w_0 \in W$ an initial state of W. A path σ through K is an infinite sequence $\sigma \in W^\omega$ such that $\sigma[0] = w_0$ and $(\sigma[i], \sigma[i+1]) \in R$ for every $i \geq 0$. The trace of σ is defined as $t(\sigma) := \eta(\sigma[0])\eta(\sigma[1]) \cdots \in (2^{\mathrm{AP}})^\omega$. A Kripke structure K induces a set of traces $\mathrm{Traces}(\mathrm{K}) = \{t(\sigma) \mid \sigma \text{ is a path through } \mathrm{K}\}$.

▶ **Definition 1.** *The* model checking problem of a logic $\mathcal{L}$ *is the following decision problem: Given a formula $\varphi \in \mathcal{L}$ and a Kripke structure K over* AP, *determine whether $(\mathit{Traces}(K), 0) \models \varphi$.*

Our undecidability results are obtained by reductions from *non-deterministic 3-counter machines*. A non-deterministic 3-counter machine M consists of a list I of n instructions that manipulate three counters C_l, C_m, and C_r. All instructions are of the following forms:

- C_a^+ goto $\{j_1, j_2\}$, C_a^- goto $\{j_1, j_2\}$, if $C_a = 0$ goto $j_1$else goto j_2,

where $a \in \{l, m, r\}$, $0 \leq j_1, j_2 < n$. A *configuration* is a tuple (i, j, k, t), where $0 \leq i < n$ is the next instruction to be executed, and $j, k, t \in \mathbb{N}$ are the current values of the counters C_l, C_m, and C_r. The execution of the instruction $i\colon C_a^+$ goto $\{j_1, j_2\}$ ($i\colon C_a^-$ goto $\{j_1, j_2\}$, resp.) increments (decrements, resp.) the value of the counter C_a by 1. The next instruction is selected nondeterministically from the set $\{j_1, j_2\}$. The instruction $i\colon$ if $C_a = 0$ goto j_1, else goto j_2 checks whether the value of the counter C_a is currently 0 and proceeds to the next instruction accordingly. The *consecution relation* $\rightsquigarrow_{\mathrm{c}}$ of configurations is defined as usual. The *lossy consecution relation* $(i_1, i_2, i_3, i_4) \rightsquigarrow_{\mathrm{lc}} (j_1, j_2, j_3, j_4)$ of configurations holds if $(i_1, i_2', i_3', i_4') \rightsquigarrow_{\mathrm{c}} (j_1, j_2', j_3', j_4')$ holds for some $i_2', i_3', i_4', j_2', j_3', j_4'$ with $i_2 \geq i_2'$, $i_3 \geq i_3'$, $i_4 \geq i_4'$, $j_2' \geq j_2$, $j_3' \geq j_3$, and $j_4' \geq j_4$. A *(lossy) computation* is an infinite sequence of (lossy) consecutive configurations starting from the initial configuration $(0, 0, 0, 0)$. A (lossy) computation is *b-recurring* if the instruction labelled b occurs infinitely often in it. Deciding whether a given non-deterministic 3-counter machine has a b-recurring (b-recurring lossy) computation for a given b is Σ_1^1-complete (Σ_1^0-complete, resp.) [3, 43].

We reduce the existence of a b-recurring lossy computation of a given 3-counter machine M and an instruction label b to the model checking problem of TeamLTL$(\subseteq, \varovee)$. We also illustrate that with a single instance of A we can enforce non-lossy computation instead.

▶ **Theorem 2.** *Model checking for* TeamLTL$(\subseteq, \varovee)$ *is Σ_1^0-hard.*

Proof. Given a set I of instructions of a 3-counter machine M, and an instruction label b, we construct a TeamLTL$(\subseteq, \varovee)$-formula $\varphi_{I,b}$ and a Kripke structure K_I such that

$$\big(\mathrm{Traces}(\mathrm{K}_I), 0\big) \models \varphi_{I,b} \quad \text{iff} \quad M \text{ has a } b\text{-recurring lossy computation.} \tag{1}$$

The Σ_1^0-hardness then follows since our construction is clearly computable. The idea is the following: Put $n := |I|$. A set T of traces using propositions $\{c_l, c_m, c_r, d, 0, \dots, n-1\}$ encodes the sequence $(\vec{c}_j)_{j \in \mathbb{N}}$ of configurations, if for each $j \in \mathbb{N}$ and $\vec{c}_j = (i, v_l, v_m, v_r)$

- $t[j] \cap \{0, \dots, n-1\} = \{i\}$, for all $t \in T$,
- $|\{t[j, \infty] \mid c_s \in t[j], t \in T\}| = v_s$, for each $s \in \{l, m, r\}$.

Hence, we use $T[j,\infty]$ to encode the configuration $\vec{c}_j$; the propositions $0,\dots,n-1$ are used to encode the next instruction, and c_l, c_m, c_r, d are used to encode the values of the counters. The proposition d is a dummy proposition used to separate traces with identical postfixes with respect to c_l, c_m, and c_r. The Kriple structure $\mathrm{K}_I = (W, R, \eta, w_0)$ over the set of propositions $\{c_l, c_m, c_r, d, 0, \dots, n-1\}$ is defined such that every possible sequence of configurations of M starting from $(0,0,0,0)$ can be encoded by some team $(T, 0)$, where $T \subseteq \mathrm{Traces}(\mathrm{K}_I)$. A detailed construction of the formula $\varphi_{I,b}$ and the Kripke structure K_I together with a detailed proof for the fact that (1) indeed holds can be found in the full version of this paper [47]. ◀

The underlying reason for utilising lossy computations in the above proof is the following: In our encoding, we use the cardinality of the set $|\{t[j,\infty] \mid c_l \in t[j], t \in T\}|$ to encode the value of the counter C_l in the jth configuration. It might, however, happen that two distinct traces $t, t' \in T$ have the same postfix, that is, $t[j,\infty] = t'[j,\infty]$, for some $j \in \mathbb{N}$. The collapse of two traces encoding distinct increments of the counter C_l then corresponds to the uncontrollable decrement of the counter values in lossy computations. Using the universal team quantifier A we can forbid this effect, and encode non-lossy computations. The proof of the following theorem can be found in the full version of this paper [47].

▶ **Theorem 3.** *Model checking for* $\mathrm{TeamLTL}(\subseteq, \varovee, \mathsf{A})$ *is* Σ^1_1*-hard. This holds already for the fragment with a single occurrence of* A.

As is common in the LTL-setting, the model checking problem of $\mathrm{TeamLTL}(\subseteq, \varovee)$ can be embedded in its satisfiability problem using auxiliary propositions and $\mathrm{TeamLTL}(\subseteq, \varovee)$-formulae. A formula φ is satisfiable, if there exists a non-empty T such that $(T, 0) \models \varphi$. We thus obtain the following corollary, which is also proven in the full version of this paper [47].

▶ **Corollary 4.** *The satisfiability problems for* $\mathrm{TeamLTL}(\subseteq, \varovee)$ *and* $\mathrm{TeamLTL}(\subseteq, \varovee, \mathsf{A})$ *are* Σ^0_1*-hard and* Σ^1_1*-hard, resp.*

4 Quantification-based Hyperlogics and Team Semantics

In this section, we define those quantification-based hyperlogics against which we compare TeamLTL in the rest of the paper. TeamLTL and HyperLTL are known to have orthogonal expressivity [34] but apart from that, nothing is known about the relationship between the different variants of TeamLTL and other temporal hyperlogics such as HyperQPTL [40, 9]. We aim to identify fragments of the logics with similar expressivity to better understand the relative expressivity of TeamLTL for the specification of hyperproperties.

HyperQPTL$^+$ [16] is a temporal logic for hyperproperties. It subsumes HyperLTL and HyperQPTL, so we proceed to give a definition of HyperQPTL$^+$ and define the latter logics as fragments. HyperQPTL$^+$ extends LTL with explicit trace quantification and quantification of atomic propositions. As such, it also subsumes QPTL, which can express all ω-regular properties. Fix an infinite set $\mathcal{V}$ of trace variables. HyperQPTL$^+$ has three types of quantifiers, one for traces and two for propositional quantification.

$$\begin{aligned}
\varphi &::= \forall\pi.\,\varphi \mid \exists\pi.\,\varphi \mid \overset{u}{\forall}p.\,\varphi \mid \overset{u}{\exists}p.\,\varphi \mid \forall p.\,\varphi \mid \exists p.\,\varphi \mid \psi\\
\psi &::= p_\pi \mid \neg p_\pi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \bigcirc\psi \mid \psi\,\mathcal{U}\,\psi \mid \psi\,\mathcal{W}\,\psi
\end{aligned}$$

Here, $p \in \mathrm{AP}$, $\pi \in \mathcal{V}$, and $\forall\pi$ and $\exists\pi$ stand for universal and existential trace quantifiers, $\forall p$ and $\exists p$ stand for (non-uniform) propositional quantifiers, and $\overset{u}{\forall}p$ and $\overset{u}{\exists}p$ stand for uniform propositional quantifiers. We also study two syntactic fragments of HyperQPTL$^+$.

HyperQPTL is HyperQPTL$^+$ without non-uniform propositional quantifiers, and HyperLTL is HyperQPTL$^+$ without any propositional quantifiers. In the context of HyperQPTL, we also write $\forall p$ and $\exists p$ instead of $\overset{u}{\forall} p$ and $\overset{u}{\exists} p$. For an LTL-formula φ and trace variable π, we let φ_π denote the HyperLTL-formula obtained from φ by replacing all proposition symbols p by their indexed versions p_π. We extend this convention to tuples of formulae as well.

The semantics of HyperQPTL$^+$ is defined over a set T of traces. Intuitively, the atomic formula p_π asserts that p holds on trace π. Uniform propositional quantifications $\overset{u}{\forall} p$ and $\overset{u}{\exists} p$ add an atomic proposition p such that all traces agree on the valuation of p on any given time step i, whereas non-uniform propositional quantifications $\forall p$ and $\exists p$ colour the traces in T in an arbitrary manner. Non-uniform propositional quantification thus implements true second-order quantification, whereas uniform propositional quantification can be interpreted as a quantification of a set of points in time.

A *trace assignment* is a function $\Pi : \mathcal{V} \to T$ that maps each trace variable in $\mathcal{V}$ to some trace in T. A *modified trace assignment* $\Pi[\pi \mapsto t]$ is equal to Π except that $\Pi[\pi \mapsto t](\pi) = t$. For any subset $A \subseteq \mathrm{AP}$, we write $t \upharpoonright A$ for the projection of t on A (i.e., $(t \upharpoonright A)[i] := t[i] \cap A$ for all $i \in \mathbb{N}$). For any two trace assignments Π and Π', we write $\Pi =_A \Pi'$, if $\big(\Pi(\pi) \upharpoonright A\big) = \big(\Pi'(\pi) \upharpoonright A\big)$ for all $\pi \in \mathcal{V}$. Similarly, $T =_A T'$ whenever $\{t \upharpoonright A \mid t \in T\} = \{t \upharpoonright A \mid t \in T'\}$. For a sequence $s \in (2^{\{p\}})^\omega$ over a single propositional variable p, we write $T[p \mapsto s]$ for the set of traces obtained from T by reinterpreting p on all traces as in s while ensuring that $T[p \mapsto s] =_{\mathrm{AP}\setminus\{p\}} T$. We use $\Pi[p \mapsto s]$ accordingly. The satisfaction relation $\Pi, i \models_T \varphi$ for HyperQPTL$^+$-formulae φ is defined as follows:

$$\begin{array}{lll}
\Pi, i \models_T p_\pi & \text{iff} & p \in \Pi(\pi)[i] \\
\Pi, i \models_T \varphi_1 \vee \varphi_2 & \text{iff} & \Pi, i \models_T \varphi_1 \text{ or } \Pi, i \models_T \varphi_2 \\
\Pi, i \models_T \neg p_\pi & \text{iff} & p \notin \Pi(\pi)[i] \\
\Pi, i \models_T \varphi_1 \wedge \varphi_2 & \text{iff} & \Pi, i \models_T \varphi_1 \text{ and } \Pi, i \models_T \varphi_2 \\
\Pi, i \models_T p_\pi & \text{iff} & p \in \Pi(\pi)[i] \\
\Pi, i \models_T \neg p_\pi & \text{iff} & p \notin \Pi(\pi)[i] \\
\Pi, i \models_T \varphi_1 \vee \varphi_2 & \text{iff} & \Pi, i \models_T \varphi_1 \text{ or } \Pi, i \models_T \varphi_2 \\
\Pi, i \models_T \varphi_1 \wedge \varphi_2 & \text{iff} & \Pi, i \models_T \varphi_1 \text{ and } \Pi, i \models_T \varphi_2 \\
\Pi, i \models_T \bigcirc \varphi & \text{iff} & \Pi, i+1 \models_T \varphi \\
\Pi, i \models_T \varphi_1 \,\mathcal{U}\, \varphi_2 & \text{iff} & \exists k \geq i \text{ s.t. } \Pi, k \models_T \varphi_2 \text{ and } \forall m : i \leq m < k \Rightarrow \Pi, m \models_T \varphi_1 \\
\Pi, i \models_T \varphi_1 \,\mathcal{W}\, \varphi_2 & \text{iff} & \forall k \geq i : \Pi, k \models_T \varphi_1 \text{ or } \exists m : i \leq m \leq k : \Pi, m \models_T \varphi_2 \\
\Pi, i \models_T \exists \pi. \varphi & \text{iff} & \Pi[\pi \mapsto t], i \models_T \varphi \text{ for some } t \in T \\
\Pi, i \models_T \forall \pi. \varphi & \text{iff} & \Pi[\pi \mapsto t], i \models_T \varphi \text{ for all } t \in T \\
\Pi, i \models_T \overset{u}{\exists} p. \varphi & \text{iff} & \Pi[p \mapsto s], i \models_{T[p\mapsto s]} \varphi \text{ for some } s \in (2^{\{p\}})^\omega \\
\Pi, i \models_T \overset{u}{\forall} p. \varphi & \text{iff} & \Pi[p \mapsto s], i \models_{T[p\mapsto s]} \varphi \text{ for all } s \in (2^{\{p\}})^\omega \\
\Pi, i \models_T \exists p. \varphi & \text{iff} & \Pi', i \models_{T'} \varphi \text{ for some } T' \subseteq (2^{\mathrm{AP}})^\omega \text{and } \Pi' \colon \mathcal{V} \to T' \text{ such that} \\
 & & T =_{\mathrm{AP}\setminus\{p\}} T' \text{ and } \Pi =_{\mathrm{AP}\setminus\{p\}} \Pi' \\
\Pi, i \models_T \forall p. \varphi & \text{iff} & \Pi', i \models_{T'} \varphi \text{ for all } T' \subseteq (2^{\mathrm{AP}})^\omega \text{and } \Pi' \colon \mathcal{V} \to T' \text{ such that} \\
 & & T =_{\mathrm{AP}\setminus\{p\}} T' \text{ and } \Pi =_{\mathrm{AP}\setminus\{p\}} \Pi'
\end{array}$$

In the sequel, we describe fragments of HyperQPTL$^+$ by restricting the quantifier prefixes of formulae. We use $\exists_\pi \,/\, \forall_\pi$ to denote trace quantification, $\overset{u}{\exists}_p \,/\, \overset{u}{\forall}_p$ for uniform propositional quantification, and $\exists_p \,/\, \forall_p$ for non-uniform propositional quantification. We use $\exists$ ($\forall$, resp.)

if we do not need to distinguish between the different types of existential (universal, resp.) quantifiers. We write Q to refer to both $\exists$ and $\forall$. For a logic L and a regular expression e, we write eL to denote the set of L-formulae whose quantifier prefixes are generated by e. E.g., $\forall^*\exists^*$HyperQPTL refers to HyperQPTL-formulae with quantifier prefix $\{\overset{u}{\forall}_p, \forall_\pi\}^*\{\overset{u}{\exists}_p, \exists_\pi\}^*$.

Next we relate the expressivity of extensions of TeamLTL to fragments of HyperQPTL$^+$. We show that TeamLTL($\ovee, \overset{1}{\mathsf{A}}$) and TeamLTL($\ovee, \sim\bot, \overset{1}{\mathsf{A}}$) can be translated to the prefix fragments $\exists_p\overset{u}{Q}{}^*_p\exists^*_\pi\forall_\pi$ and $\exists_p\overset{u}{Q}{}^*_p\forall_\pi$ of HyperQPTL$^+$. The translations provide insight into the limits of the expressivity of different extensions of TeamLTL. In particular, they show that in order to simulate the generation of subteams with the $\vee$-operator in TeamLTL, one existential second-order quantifier $\exists_p$ is sufficient. Meanwhile, the difference between downward closed team properties and general team properties manifests itself by a different need for trace quantifiers: for downward closed properties, a single $\forall_\pi$ quantifier is enough, whereas in the general case, a $\exists^*_\pi\forall_\pi$ quantifier alternation is needed.

As a prerequisite for the translation, we establish that evaluating TeamLTL($\ovee, \sim\bot, \overset{1}{\mathsf{A}}$)-formulae can only create countably many different teams. For a given team (T, i) and TeamLTL($\ovee, \sim\bot, \overset{1}{\mathsf{A}}$)-formula φ, the verification of $(T, i) \models \varphi$ boils down to checking statements of the form $(S, j) \models \psi$, where $S \in \mathcal{S}_T \subseteq 2^T$ for some set $\mathcal{S}_T$, $j \in \mathbb{N}$, and ψ is an atomic formula, together with expressions of the form $S_1 = S_2 \cup S_3$, where $S_1, S_2, S_3 \in \mathcal{S}_T$. The following lemma, proven in the full version of this paper [47], implies that the set $\mathcal{S}_T$ can be fixed as a countable set that depends only on T.

▶ **Lemma 5.** *For every set T of traces over a countable* AP*, there exists a countable $\mathcal{S}_T \subseteq 2^T$ such that, for every* TeamLTL($\ovee, \sim\bot, \overset{1}{\mathsf{A}}$)*-formula φ and $i \in \mathbb{N}$, $(T, i) \models \varphi$ iff $(T, i) \models^* \varphi$, where the satisfaction relation $\models^*$ is defined the same way as $\models$ except that in the semantic clause for $\vee$ we require additionally that the two subteams $T_1, T_2 \in \mathcal{S}_T$.*

Using of the above lemma, we obtain translations from the most interesting extensions of TeamLTL to weak prefix fragments of HyperQPTL$^+$; for details and proofs see the full version of this paper [47].

▶ **Theorem 6.** *For every $\varphi \in$* TeamLTL($\ovee, \sim\bot, \overset{1}{\mathsf{A}}$) *there exists an equivalent* HyperQPTL$^+$*-formula φ^* in the $\exists_p\overset{u}{Q}{}^*_p\exists^*_\pi\forall_\pi$ fragment. If $\varphi \in$* TeamLTL($\ovee, \overset{1}{\mathsf{A}}$)*, φ^* can be defined in the $\exists_p\overset{u}{Q}{}^*_p\forall_\pi$ fragment. The size of φ^* is linear w.r.t. the size of φ.*

5 Decidable fragments of TeamLTL

In this section, we further study the expressivity landscape between the frameworks of TeamLTL and HyperLTL. We utilise these connections to prove decidability of the model checking problem of certain variants of TeamLTL. We compare the expressivity of extensions of TeamLTL that satisfy certain semantic invariances to that of $\forall^*$HyperLTL and $\overset{u}{\exists}{}^*_p\forall_\pi$HyperQPTL. Thereby, we provide a partial answer to an open problem posed in [34] concerning the complexity of the model checking problem of TeamLTL and its extensions. The problem is known to be in PSPACE for the fragment of TeamLTL without $\vee$ [34]. However, for TeamLTL with $\vee$, no meaningful upper bounds for the problem was known before. The best previous upper bound could be obtained from TeamLTL($\sim$), for which the problem is highly undecidable [35]. The reason for this lack of results is that developing algorithms for team logics with $\vee$ turned out to be comparatively hard. The main source of difficulty is that the semantic definition of $\vee$ does not yield any reasonable compositional brute force algorithm: the verification of $(T, i) \models \varphi \vee \psi$ with T generated by a finite Kripke structure proceeds by checking that $(T_1, i) \models \varphi$ and $(T_2, i) \models \psi$ for some $T_1 \cup T_2 = T$, but it can well be that T_1 and T_2 cannot be generated from any finite Kripke structure whatsoever. The main results of this section are the decidability of the model checking problems of the k-*coherent fragment* of TeamLTL($\sim$) and the *left-flat fragment* of TeamLTL($\varovee, \overset{1}{\mathsf{A}}$). We obtain inclusions to EXPSPACE by translations to $\forall^*$HyperLTL and $\overset{u}{\exists}{}^*_p\forall_\pi$HyperQPTL.

5.1 The k-coherent fragment and ∀*HyperLTL

The universal fragment of HyperLTL is one of the most studied fragments as it contains the set of *safety* hyperproperties expressible in HyperLTL [8]. In particular, formulae of the form $\forall\pi_1 \ldots \forall\pi_k.\, \psi$ state k-safety properties (if ψ is a safety LTL formula) [14], where non-satisfying trace sets contain bad prefixes of at most k traces. In general, $\forall^k$HyperLTL formulae satisfy the following inherent invariance: $\emptyset, i \models_T \varphi$ iff $\emptyset, i \models_{T'} \varphi$, for all $T' \subseteq T$ s.t. $|T'| \leq k$. That is, a $\forall^k$HyperLTL-formula φ is satisfied by a trace set T, iff it is satisfied by all subsets of T of size at most k. This property is called k-*coherence* in the team semantics literature [31]. The main result of this section is that all k-coherent properties expressible in TeamLTL($\sim$) are expressible in $\forall^k$HyperLTL. This implies that, with respect to trace properties, all logics between TeamLTL($\overset{1}{\mathsf{A}}$) and TeamLTL($\sim$) are equi-expressive to $\forall$HyperLTL, i.e., LTL.

▶ **Definition 7.** *Let $\mathcal{A}$ be any collection of atoms and connectives introduced so far. A formula φ in* TeamLTL($\mathcal{A}$) *is said to be k-*coherent *($k \in \mathbb{N}$) if for every team (T, i),*

$$(T, i) \models \varphi \quad \textit{iff} \quad (S, i) \models \varphi \textit{ for every } S \subseteq T \textit{ with } |S| \leq k.$$

We will next show that, with respect to k-coherent properties, TeamLTL($\sim$) is at most as expressive as $\forall^k$HyperLTL. We define a translation from TeamLTL($\sim$) to $\forall^*$HyperLTL that preserves the satisfaction relation with respect to teams of bounded size. Given a finite set Φ of trace variables, the translation is defined as follows:

$$\begin{array}{lll}
p^\Phi := \bigwedge_{\pi\in\Phi} p_\pi & (\neg p)^\Phi := \bigwedge_{\pi\in\Phi} \neg p_\pi & (\sim\varphi)^\Phi := \neg\varphi^\Phi \\
(\bigcirc\varphi)^\Phi := \bigcirc\varphi^\Phi & (\varphi\wedge\psi)^\Phi := \varphi^\Phi\wedge\psi^\Phi & (\varphi\vee\psi)^\Phi := \bigvee_{\Phi_0\cup\Phi_1=\Phi} \varphi^{\Phi_0}\wedge\psi^{\Phi_1} \\
(\varphi\,\mathcal{U}\,\psi)^\Phi := \varphi^\Phi\,\mathcal{U}\,\psi^\Phi & (\varphi\,\mathcal{W}\,\psi)^\Phi := \varphi^\Phi\,\mathcal{W}\,\psi^\Phi &
\end{array}$$

where $\neg\varphi^\Phi$ stands for the negation of φ^Φ in negation normal form. The following lemma, from which the subsequent theorem follows, is proved by induction. See the full version of this paper [47] for detailed proofs.

▶ **Lemma 8.** *Let φ be a formula of* TeamLTL($\sim$) *and $\Phi = \{\pi_1, \ldots, \pi_k\}$ a finite set of trace variables. For any team (T, i) with $|T| \leq k$, any set $S \supseteq T$ of traces, and any assignment Π with $\Pi[\Phi] = T$, we have that $(T, i) \models \varphi$ iff $\Pi, i \models_S \varphi^\Phi$. Furthermore, if φ is downward closed and $T \neq \emptyset$, then $(T, i) \models \varphi$ iff $\emptyset, i \models_T \forall\pi_1 \ldots \forall\pi_k.\, \varphi^\Phi$.*

▶ **Theorem 9.** *Every k-coherent property that is definable in* TeamLTL($\sim$) *is also definable in* $\forall^k$HyperLTL.

Since model checking for $\forall^*$HyperLTL is PSPACE-complete and its data complexity (model checking with a fixed formula) is NL-complete [19], and since the above translation from TeamLTL($\sim$) to $\forall^*$HyperLTL is exponential for any k, we get the following corollary:

▶ **Corollary 10.** *For any fixed $k \in \mathbb{N}$, the model checking problem for* TeamLTL($\sim$)*, restricted to k-coherent properties, is in* EXPSPACE*, and in* NL *for data complexity.*

Clearly $(T, i) \models \overset{1}{\mathsf{A}}\varphi$ iff $\emptyset, i \models_T \forall\pi.\, \varphi_\pi$ for any $\varphi \in$ LTL, and hence we obtain the following:

▶ **Corollary 11.** *The restriction of* TeamLTL($\overset{1}{\mathsf{A}}$) *to formulae of the form $\overset{1}{\mathsf{A}}\varphi$ is expressively equivalent to* $\forall$HyperLTL*.*

While model checking for k-coherent properties is decidable, checking whether a given formula defines a k-coherent property is not, in general, decidable.

▶ **Theorem 12.** *Checking whether a* TeamLTL($\subseteq, \varovee$)*-formula is* 1*-coherent is undecidable.*

Proof. The idea of the undecidability proof is as follows: Given any TeamLTL($\subseteq, \varovee$)-formula φ, we can use a simple rewriting rule to obtain an LTL-formula φ^* such that φ is not satisfiable (in the sense of TeamLTL) if and only if φ is 1-coherent and φ^* is not satisfiable (in the LTL-sense). Now, since checking LTL-satisfiability can be done in PSPACE [44] and non-satisfiability for TeamLTL($\subseteq, \varovee$) is Π^0_1-hard by Corollary 4, it follows that checking 1-coherence is Π^0_1-hard as well. For a detailed proof, see the full version of this paper [47]. ◀

The same holds for any extension of TeamLTL with an undecidable satisfiability or validity problem and whose formulae can be computably translated to TeamLTL while preserving satisfaction over singleton teams.

5.2 The left-flat fragment and HyperQPTL$^+$

In this subsection, we show that formulae φ from the left-flat fragment of TeamLTL($\varovee, \overset{1}{\mathsf{A}}$) (defined below) can be translated to HyperQPTL formulae that are linear in the size of φ. The known model checking algorithm of HyperQPTL [40] then immediately yields a model checking algorithm for the left-flat fragment of TeamLTL($\varovee, \overset{1}{\mathsf{A}}$).

▶ **Definition 13** (The left-flat fragment)**.** *Let $\mathcal{A}$ be a collection of atoms and connectives. A* TeamLTL($\mathcal{A}$)*-formula belongs to the left-flat fragment if in each of its subformulae of the form $\psi\,\mathcal{U}\,\varphi$ or $\psi\,\mathcal{W}\,\varphi$, ψ is a flat formula (as defined in Section 2).*

Such defined fragment allows for arbitrary use of the $\Diamond$ operator, and therefore remains incomparable to HyperLTL [34]. For instance, $\Diamond\,\mathrm{dep}(a, b) \vee \Diamond\,\mathrm{dep}(c, d)$ is a nontrivial formula in this fragment. It states that the set of traces can be partitioned into two parts, one where eventually a determines the value of b, and another where eventually c determines the value of d. The property is not expressible in HyperLTL, because HyperLTL cannot state the property "there is a point in time at which p holds on all (or infinitely many) traces" [5].

It follows from Theorem 12 that checking whether a TeamLTL($\subseteq, \varovee$)-formula belongs to the left-flat fragment is undecidable (as flatness equals 1-coherency). Nevertheless, a decidable syntax for left-flat formulae can be obtained by using the operator $\overset{1}{\mathsf{A}}$. Formulae $\overset{1}{\mathsf{A}}\psi$ are always flat and equivalent to ψ if ψ is flat. Therefore, in Definition 13, instead of imposing the semantic condition of ψ being flat in subformulae $\psi\,\mathcal{U}\,\varphi$ and $\psi\,\mathcal{W}\,\varphi$, we could require that the subformulae must be of the form $(\overset{1}{\mathsf{A}}\psi)\,\mathcal{U}\,\varphi$ or $(\overset{1}{\mathsf{A}}\psi)\,\mathcal{W}\,\varphi$.

We now describe a translation from the left-flat fragment of TeamLTL($\varovee, \overset{1}{\mathsf{A}}$) to the $\overset{u}{\exists}{}^*_p\forall_\pi$ fragment of HyperQPTL. In this translation, we make use of the fact that satisfaction of flat formulae φ can be determined with the usual (single-traced) LTL semantics. In the evaluation of φ, it is thus sufficient to consider only finitely many subteams, whose temporal behaviour can be reflected by existentially quantified q-sequences. The quantified sequences refer to points in time, at which subformulae have to hold for a trace to belong to a team.

A left-flat TeamLTL($\varovee$, $\overset{1}{\mathsf{A}}$)-formula φ will be translated into a formula with existential propositional quantifiers followed by a single trace quantifier. The existential propositional quantifiers either indicate a point in time at which a subformula of φ_i is evaluated or resolve the decision for $\varovee$-choices. For subformulae, we use propositions r^{φ_i}, and if the same subformula occurs multiple times, it is associated with different r^{φ_i}. For the resolution of $\varovee$-choices we use propositions $d^{\varphi \varovee \psi}$. Additionally, r is a free proposition for the point in time at which φ is to be evaluated. The universal quantifier $\forall\pi$ sorts each trace into one of the finitely many teams.

Let $\forall\pi.\,\hat{\psi}$ be the HyperLTL formula given by Theorem 9 for any flat formula ψ (since a flat formula is 1-coherent). We translate φ inductively with respect to r:

$$
\begin{aligned}
[p, r] &\coloneqq \Box(r_\pi \rightarrow p_\pi)\\
[\neg p, r] &\coloneqq \Box(r_\pi \rightarrow \neg p_\pi)\\
[\bigcirc \varphi, r] &\coloneqq \Box(r_\pi \leftrightarrow \bigcirc r^{\varphi}_\pi) \wedge [\varphi, r^{\varphi}]\\
[\overset{1}{\mathsf{A}}\varphi, r] &\coloneqq \Box(r_\pi \rightarrow \hat{\varphi})\\
[\varphi \wedge \psi, r] &\coloneqq [\varphi, r] \wedge [\psi, r]\\
[\varphi \vee \psi, r] &\coloneqq [\varphi, r] \vee [\psi, r]\\
[\varphi \varovee \psi, r] &\coloneqq (d^{\varphi \varovee \psi}_\pi \rightarrow [\varphi, r]) \wedge (\neg d^{\varphi \varovee \psi}_\pi \rightarrow [\psi, r])\\
[\varphi \mathcal{W} \psi, r] &\coloneqq \Box(r_\pi \rightarrow r^{\varphi}_\pi \mathcal{W}(r^{\psi}_\pi \wedge \bigcirc\Box\neg r^{\psi}_\pi))
\end{aligned}
$$

Now, let $r^1, \dots r^n$ be the free propositions occurring in $[\varphi, r]$ and π the free trace variable. Define the following $\overset{u}{\exists}{}^*_p\forall_\pi$ HyperQPTL formula: $\overset{u}{\exists} r \overset{u}{\exists} r^1 \dots \overset{u}{\exists} r^n.\,\forall\pi.\, r_\pi \wedge \bigcirc\Box\neg r_\pi \wedge [\varphi, r]$. Correctness of the translation can be argued intuitively as follows. The left-flat formula φ can be evaluated independently from the other traces in a team. Therefore, the operators $\mathcal{U}$ and $\mathcal{W}$, whose right-hand sides argue only about a single point in time, can only generate finitely many teams. Thus, there are only finitely many points of synchronization, all of which are quantified existentially. Every trace fits into one of the teams described by the quantified propositional variables. We verify that the translation is indeed correct in the full version of this paper [47]. As the construction in Theorem 9 yields a formula $\hat{\varphi}$ whose size is linear in the original formula φ, the translation is obviously linear. We therefore state the following theorem.

▶ **Theorem 14.** *For every formula φ from the left-flat fragment of* TeamLTL($\varovee$, $\overset{1}{\mathsf{A}}$)*, we can compute an equivalent $\overset{u}{\exists}{}^*_p\forall_\pi$* HyperQPTL *formula of size linear in the size of φ.*

Recall that the model checking problem of HyperLTL formulae with one quantifier alternation is EXPSPACE-complete [19] in the size of the formula, and PSPACE-complete in the size of the Kripke structure [19]. These results directly transfer to HyperQPTL [40] (in which HyperQPTL was called HyperLTL *with extended quantification* instead): for model checking a HyperQPTL formula, the Kripke structure can be extended by two states generating all possible q-sequences. Since the translation from TeamLTL($\varovee$, $\overset{1}{\mathsf{A}}$) to HyperQPTL yields a formula in the $\overset{u}{\exists}{}^*_p\forall_\pi$ fragment with a single quantifier alternation and preserves the size of the formula, we obtain the following theorem.

▶ **Theorem 15.** *The model checking problem for left-flat* TeamLTL($\varovee$, $\overset{1}{\mathsf{A}}$)*-formulae is in* EXPSPACE*, and in* PSPACE *for data complexity.*

6 Conclusion

We studied TeamLTL under the synchronous semantics. TeamLTL is a powerful but not yet well-studied logic that can express hyperproperties without explicit quantification over traces or propositions. As such, properties which need various different quantifiers in traditional (quantification-based) hyperlogics become expressible in a concise fashion. One of the main advantages of TeamLTL is the ability to equip it with a range of atomic statements and connectives to obtain logics of varying expressivity and complexity.

We systematically studied TeamLTL with respect to two of the main questions related to logics: the decision boundary of its model checking problem and its expressivity compared to other logics for hyperproperties. We related the expressivity of TeamLTL to the hyperlogics HyperLTL, HyperQPTL, and HyperQPTL$^+$, which are obtained by extending the traditional temporal logics LTL and QPTL with trace quantifiers. We discovered that the logics TeamLTL($\ovee, \overset{1}{\mathsf{A}}$) and TeamLTL($\ovee, \sim\bot, \overset{1}{\mathsf{A}}$) are expressively complete with respect to all downward closed, and all atomic notions of dependence, respectively. We were able to show that TeamLTL($\ovee, \sim\bot, \overset{1}{\mathsf{A}}$) can be expressed in a fragment of HyperQPTL$^+$. Furthermore, for k-coherent properties, TeamLTL($\sim$) is subsumed by $\forall^*$HyperLTL. Finally, the left-flat fragment of TeamLTL($\ovee, \overset{1}{\mathsf{A}}$) can be translated to HyperQPTL. The last two results induce efficient model checking algorithms for the respective logics. In addition, we showed that model checking of TeamLTL($\subseteq, \ovee$) is already undecidable, and that the additional use of the A quantifier makes the problem highly undecidable.

We conclude with some open problems and directions for future work: What is the complexity of model checking for TeamLTL (with the disjunction $\vee$ but without additional atoms and connectives)? Is it decidable, and is there a translation to HyperQPTL? An interesting avenue for future work is also to explore team semantics of more expressive logics than LTL such as linear time μ-calculus, or branching time logics such as CTL* and the full modal μ-calculus.

References

1 Erika Ábrahám and Borzoo Bonakdarpour. Hyperpctl: A temporal logic for probabilistic hyperproperties. In Annabelle McIver and András Horváth, editors, *Quantitative Evaluation of Systems - 15th International Conference, QEST 2018, Beijing, China, September 4-7, 2018, Proceedings*, volume 11024 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2018. doi:10.1007/978-3-319-99154-2_2.

2 Shreya Agrawal and Borzoo Bonakdarpour. Runtime verification of k-safety hyperproperties in hyperltl. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 239–252. IEEE Computer Society, 2016. doi:10.1109/CSF.2016.24.

3 Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994. doi:10.1145/174644.174651.

4 Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. A temporal logic for asynchronous hyperproperties. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*, volume 12759 of *Lecture Notes in Computer Science*, pages 694–717. Springer, 2021. doi:10.1007/978-3-030-81685-8_33.

5 Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Unifying hyper and epistemic temporal logics. In *Proceedings of FoSSaCS*, volume 9034 of *LNCS*, pages 167–182. Springer, 2015. doi:10.1007/978-3-662-46678-0_11.

6 Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NuSMV: A new symbolic model verifier. In *Proc. International Conference on Computer Aided Verification*, pages 495–499, July 1999.

7 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *POST 2014*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.

8 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.

9 Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785713.

10 Norine Coenen, Bernd Finkbeiner, Christopher Hahn, Jana Hofmann, and Yannick Schillo. Runtime enforcement of hyperproperties. *To appear at the 19th International Symposium on Automated Technology for Verification and Analysis (ATVA 2021)*, 2021.

11 Byron Cook, Eric Koskinen, and Moshe Vardi. Temporal property verification as a program analysis task. In *Proc. Computer Aided Verification*, pages 333–348, July 2011.

12 Jukka Corander, Antti Hyttinen, Juha Kontinen, Johan Pensar, and Jouko Väänänen. A logical approach to context-specific independence. *Ann. Pure Appl. Logic*, 170(9):975–992, 2019. doi:10.1016/j.apal.2019.04.004.

13 Arnaud Durand, Miika Hannula, Juha Kontinen, Arne Meier, and Jonni Virtema. Probabilistic team semantics. In *FoIKS*, volume 10833 of *Lecture Notes in Computer Science*, pages 186–206. Springer, 2018. doi:10.1007/978-3-319-90050-6_11.

14 Bernd Finkbeiner, Lennart Haas, and Hazem Torfah. Canonical representations of k-safety hyperproperties. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, pages 17–31. IEEE, 2019. doi:10.1109/CSF.2019.00009.

15 Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.13.

16 Bernd Finkbeiner, Christopher Hahn, Jana Hofmann, and Leander Tentrup. Realizing ømega-regular hyperproperties. 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 40–63. Springer, 2020. doi:10.1007/978-3-030-53291-8_4.

17 Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. Synthesis from hyperproperties. *Acta Informatica*, 57(1-2):137–163, 2020. doi:10.1007/s00236-019-00358-2.

18 Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. *Formal Methods Syst. Des.*, 54(3):336–363, 2019. doi:10.1007/s10703-019-00334-z.

19 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In *Proceedings of CAV*, volume 9206 of *LNCS*, pages 30–48. Springer, 2015. doi:10.1007/978-3-319-21690-4_3.

20 Pietro Galliani and Lauri Hella. Inclusion Logic and Fixed Point Logic. In *CSL 2013*, pages 281–295, 2013.

21 Giuseppe De Giacomo, Paolo Felli, Marco Montali, and Giuseppe Perelli. Hyperldlf: a logic for checking properties of finite traces process logs. 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 1859–1865. ijcai.org, 2021. doi:10.24963/ijcai.2021/256.

22 Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021. doi:10.1145/3434319.

23 Miika Hannula and Juha Kontinen. A finite axiomatization of conditional independence and inclusion dependencies. *Inf. Comput.*, 249:121–137, 2016. doi:10.1016/j.ic.2016.04.001.

24 Miika Hannula, Juha Kontinen, Jan Van den Bussche, and Jonni Virtema. Descriptive complexity of real computation and probabilistic independence logic. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 550–563. ACM, 2020. doi:10.1145/3373718.3394773.

25 Miika Hannula, Juha Kontinen, Jonni Virtema, and Heribert Vollmer. Complexity of propositional logics in team semantic. *ACM Trans. Comput. Log.*, 19(1):2:1–2:14, 2018. doi:10.1145/3157054.

26 Lauri Hella, Antti Kuusisto, Arne Meier, and Jonni Virtema. Model checking and validity in propositional and modal inclusion logics. *J. Log. Comput.*, 29(5):605–630, 2019. doi:10.1093/logcom/exz008.

27 Lauri Hella, Kerkko Luosto, Katsuhiko Sano, and Jonni Virtema. The expressive power of modal dependence logic. In Rajeev Goré, Barteld P. Kooi, and Agi Kurucz, editors, *Advances in Modal Logic 10, invited and contributed papers from the tenth conference on "Advances in Modal Logic," held in Groningen, The Netherlands, August 5-8, 2014*, pages 294–312. College Publications, 2014. URL: http://www.aiml.net/volumes/volume10/Hella-Luosto-Sano-Virtema.pdf.

28 Hsi-Ming Ho, Ruoyu Zhou, and Timothy M. Jones. On verifying timed hyperproperties. In Johann Gamper, Sophie Pinchinat, and Guido Sciavicco, editors, *26th International Symposium on Temporal Representation and Reasoning, TIME 2019, October 16-19, 2019, Málaga, Spain*, volume 147 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.TIME.2019.20.

29 Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23:279–295, 1997.

30 Tapani Hyttinen, Gianluca Paolini, and Jouko Väänänen. A Logic for Arguing About Probabilities in Measure Teams. *Arch. Math. Logic*, 56(5-6):475–489, 2017. doi:10.1007/s00153-017-0535-x.

31 Jarmo Kontinen. Coherence and computational complexity of quantifier-free dependence logic formulas. *Studia Logica*, 101(2):267–291, 2013. doi:10.1007/s11225-013-9481-8.

32 Juha Kontinen, Julian-Steffen Müller, Henning Schnoor, and Heribert Vollmer. A van benthem theorem for modal team semantics. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 277–291. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.277.

33 Andreas Krebs, Arne Meier, and Jonni Virtema. A team based variant of CTL. In Fabio Grandi, Martin Lange, and Alessio Lomuscio, editors, *22nd International Symposium on Temporal Representation and Reasoning, TIME 2015, Kassel, Germany, September 23-25, 2015*, pages 140–149. IEEE Computer Society, 2015. doi:10.1109/TIME.2015.11.

34 Andreas Krebs, Arne Meier, Jonni Virtema, and Martin Zimmermann. Team Semantics for the Specification and Verification of Hyperproperties. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.MFCS.2018.10.

35 Martin Lück. On the complexity of linear temporal logic with team semantics. *Theoretical Computer Science*, 2020. doi:10.1016/j.tcs.2020.04.019.

36 Corto Mascle and Martin Zimmermann. The keys to decidable hyperltl satisfiability: Small models or very simple formulas. 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 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CSL.2020.29.

37 John McLean. Proving noninterference and functional correctness using traces. *Journal of Computer Security*, 1(1):37–58, 1992. doi:10.3233/JCS-1992-1103.

38 Nir Piterman and Amir Pnueli. Temporal logic and fair discrete systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 27–73. Springer, 2018. doi:10.1007/978-3-319-10575-8_2.

39 Amir Pnueli. The Temporal Logic of Programs. In *FOCS 1977*, pages 46–57, 1977.

40 Markus N. Rabe. *A Temporal Logic Approach to Information-Flow Control*. PhD thesis, Saarland University, 2016.

41 A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 8-10, 1995*, pages 114–127. IEEE Computer Society, 1995. doi:10.1109/SECPRI.1995.398927.

42 Katsuhiko Sano and Jonni Virtema. Characterising modal definability of team-based logics via the universal modality. *Ann. Pure Appl. Log.*, 170(9):1100–1127, 2019. doi:10.1016/j.apal.2019.04.009.

43 Philippe Schnoebelen. Lossy counter machines decidability cheat sheet. In Antonín Kucera and Igor Potapov, editors, *Reachability Problems, 4th International Workshop, RP 2010, Brno, Czech Republic, August 28-29, 2010. Proceedings*, volume 6227 of *Lecture Notes in Computer Science*, pages 51–75. Springer, 2010. doi:10.1007/978-3-642-15349-5_4.

44 A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837.

45 Jouko Väänänen. *Dependence Logic*. Cambridge University Press, 2007.

46 Jouko Väänänen. Modal dependence logic. In *New Perspectives on Games and Interaction*, 2008.

47 Jonni Virtema, Jana Hofmann, Bernd Finkbeiner, Juha Kontinen, and Fan Yang. Linear-time temporal logic with team semantics: Expressivity and complexity. *arXiv preprint*, 2020. arXiv:2010.03311.

48 Fan Yang and Jouko Väänänen. Propositional team logics. *Annals of Pure and Applied Logic*, 168(7):1406–1441, 2017. doi:10.1016/j.apal.2017.01.007.

49 Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *16th IEEE Computer Security Foundations Workshop (CSFW-16 2003), 30 June - 2 July 2003, Pacific Grove, CA, USA*, page 29. IEEE Computer Society, 2003. doi:10.1109/CSFW.2003.1212703.