

# **36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science**

**FSTTCS 2016, December 13–15, 2016, Chennai, India**

Edited by

Akash Lal

S. Akshay

Saket Saurabh

Sandeep Sen



### *Editors*

Akash Lal  
MSR India  
Bangalore 560001  
India  
akashl@microsoft.com

S. Akshay  
IIT Bombay  
Mumbai 76  
India  
akshayss@cse.iitb.ac.in

Saket Saurabh  
IMSc Chennai  
Chennai  
India  
saket@imsc.res.in

Sandeep Sen  
IIT Delhi  
Delhi  
India  
ssen@cse.iitd.ac.in

### *ACM Classification 1998*

D.2.4 Software/Program Verification, F.1.1 Models of Computation, F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic, F.4.3 Formal Languages

## **ISBN 978-3-95977-027-9**

### *Published online and open access by*

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <http://www.dagstuhl.de/dagpub/978-3-95977-027-9>.

### *Publication date*

December, 2016

### *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 <http://dnb.d-nb.de>.

### *License*

This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0): <http://creativecommons.org/licenses/by/3.0/legalcode>.



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

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

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2016.0

**ISBN 978-3-95977-027-9**

**ISSN 1868-8969**

**<http://www.dagstuhl.de/lipics>**

## LIPICs – Leibniz International Proceedings in Informatics

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

### *Editorial Board*

- Susanne Albers (TU München)
- Chris Hankin (Imperial College London)
- Deepak Kapur (University of New Mexico)
- Michael Mitzenmacher (Harvard University)
- Madhavan Mukund (Chennai Mathematical Institute)
- Catuscia Palamidessi (INRIA)
- Wolfgang Thomas (*Chair*, RWTH Aachen)
- Pascal Weil (CNRS and University Bordeaux)
- Reinhard Wilhelm (Saarland University)

**ISSN 1868-8969**

**<http://www.dagstuhl.de/lipics>**





## ■ Contents

Preface	
<i>Akash Lal, S. Akshay, Saket Saurabh and Sandeep Sen</i> .....	ix
Conference Organization	
.....	xi
External Reviewers	
.....	xiii

### Invited Talks

Fast and Powerful Hashing Using Tabulation	
<i>Mikkel Thorup</i> .....	1:1–1:2
Simple Invariants for Proving the Safety of Distributed Protocols	
<i>Mooly Sagiv</i> .....	2:1–2:1
My $\mathcal{O}$ Is Bigger Than Yours	
<i>Holger Hermanns</i> .....	3:1–3:2
Continuous Optimization: The “Right” Language for Graph Algorithms?	
<i>Aleksander Mądry</i> .....	4:1–4:2
Graph Decompositions and Algorithms	
<i>Fedor V. Fomin</i> .....	5:1–5:1
Side Channel Analysis Using a Model Counting Constraint Solver and Symbolic Execution	
<i>Tevfik Bultan</i> .....	6:1–6:2

### Contributed Papers

#### Session 1A

Mixed-Criticality Scheduling to Minimize Makespan	
<i>Sanjoy Baruah, Arvind Easwaran, and Zhishan Guo</i> .....	7:1–7:13
Capacitated $k$ -Center Problem with Vertex Weights	
<i>Aounon Kumar</i> .....	8:1–8:14
Improved Pseudo-Polynomial-Time Approximation for Strip Packing	
<i>Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, and Arindam Khan</i> .....	9:1–9:14
Embedding Approximately Low-Dimensional $\ell_2^2$ Metrics into $\ell_1$	
<i>Amit Deshpande, Prahladh Harsha, and Rakesh Venkat</i> .....	10:1–10:13

#### Session 1B

Relational Logic with Framing and Hypotheses	
<i>Anindya Banerjee, David A. Naumann, and Mohammad Nikouei</i> .....	11:1–11:16

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen

Leibniz International Proceedings in Informatics



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

FO-Definable Transformations of Infinite Strings <i>Vrunda Dave, Shankara Narayanan Krishna, and Ashutosh Trivedi</i> .....	12:1–12:14
Aperiodicity of Rational Functions Is PSPACE-Complete <i>Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote</i> .....	13:1–13:15
Homomorphism Problems for First-Order Definable Structures <i>Bartek Klin, Sławomir Lasota, Joanna Ochremiak, and Szymon Toruńczyk</i> .....	14:1–14:15
<b>Session 2A</b>	
On the Sensitivity Conjecture for Disjunctive Normal Forms <i>Karthik C. S. and Sébastien Tavenas</i> .....	15:1–15:15
The Zero-Error Randomized Query Complexity of the Pointer Function <i>Jaikumar Radhakrishnan and Swagato Sanyal</i> .....	16:1–16:13
Robust Multiplication-Based Tests for Reed-Muller Codes <i>Prahladh Harsha and Srikanth Srinivasan</i> .....	17:1–17:14
<b>Session 2B</b>	
Querying Regular Languages over Sliding Windows <i>Moses Ganardi, Danny Hucke, and Markus Lohrey</i> .....	18:1–18:14
Decidability and Complexity of Tree Share Formulas <i>Xuan Bach Le, Aquinas Hobor, and Anthony W. Lin</i> .....	19:1–19:14
One-Counter Automata with Counter Observability <i>Benedikt Bollig</i> .....	20:1–20:14
<b>Session 3A</b>	
Strong Parameterized Deletion: Bipartite Graphs <i>Ashutosh Rai and M. S. Ramanujan</i> .....	21:1–21:14
Parameterized Algorithms for List $K$ -Cycle <i>Fahad Panolan and Meirav Zehavi</i> .....	22:1–22:15
Lossy Kernels for Graph Contraction Problems <i>R. Krithika, Pranabendu Misra, Ashutosh Rai, and Prafullkumar Tale</i> .....	23:1–23:14
Faster Exact and Parameterized Algorithm for Feedback Vertex Set in Bipartite Tournaments <i>Mithilesh Kumar and Daniel Lokshтанov</i> .....	24:1–24:15
<b>Session 3B</b>	
Probabilistic Mu-Calculus: Decidability and Complete Axiomatization <i>Kim G. Larsen, Radu Mardare, and Bingtian Xue</i> .....	25:1–25:18

Interval vs. Point Temporal Logic Model Checking: an Expressiveness Comparison <i>Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron, and Pietro Sala</i> .....	26:1–26:14
Model Checking Population Protocols <i>Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar</i> .....	27:1–27:14
Visibly Linear Dynamic Logic <i>Alexander Weinert and Martin Zimmermann</i> .....	28:1–28:14

## Session 4A1

Stable Matching Games: Manipulation via Subgraph Isomorphism <i>Sushmita Gupta and Sanjukta Roy</i> .....	29:1–29:14
The Adwords Problem with Strict Capacity Constraints <i>Umang Bhaskar, Ajil Jalal, and Rahul Vaze</i> .....	30:1–30:14

## Session 4A2

Most Likely Voronoi Diagrams in Higher Dimensions <i>Nirman Kumar, Benjamin Raichel, Subhash Suri, and Kevin Verbeek</i> .....	31:1–31:14
Fréchet Distance Between a Line and Avatar Point Set <i>Aritra Banik, Fahad Panolan, Venkatesh Raman, and Vibha Sahlot</i> .....	32:1–32:14

## Session 5A1

Greed is Good for Deterministic Scale-Free Networks <i>Ankit Chauhan, Tobias Friedrich, and Ralf Rothenberger</i> .....	33:1–33:15
Independent-Set Reconfiguration Thresholds of Hereditary Graph Classes <i>Mark de Berg, Bart M. P. Jansen, and Debankur Mukherjee</i> .....	34:1–34:15

## Session 5A2

LZ77 Factorisation of Trees <i>Paweł Gawrychowski and Artur Jeż</i> .....	35:1–35:15
Finger Search in Grammar-Compressed Strings <i>Philip Bille, Anders Roy Christiansen, Patrick Hagge Cording, and Inge Li Gørtz</i> .....	36:1–36:16

## Session 6A

Characterization and Lower Bounds for Branching Program Size Using Projective Dimension <i>Krishnamoorthy Dinesh, Sajin Korothe, and Jayalal Sarma</i> .....	37:1–37:14
Finer Separations Between Shallow Arithmetic Circuits <i>Mrinal Kumar and Ramprasad Saptharishi</i> .....	38:1–38:12

Sum of Products of Read-Once Formulas <i>Ramya C. and B. V. Raghavendra Rao</i> .....	39:1–39:15
Understanding Cutting Planes for QBFs <i>Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla</i> .....	40:1–40:15
<b>Session 6B</b>	
Summaries for Context-Free Games <i>Lukáš Holík, Roland Meyer, and Sebastian Muskalla</i> .....	41:1–41:16
Admissibility in Quantitative Graph Games <i>Romain Brenguier, Guillermo A. Pérez, Jean-François Raskin, and Ocan Sankur</i> .....	42:1–42:14
Prompt Delay <i>Felix Klein and Martin Zimmermann</i> .....	43:1–43:14
Mean-Payoff Games on Timed Automata <i>Shibashis Guha, Marcin Jurdziński, Shankara Narayanan Krishna, and Ashutosh Trivedi</i> .....	44:1–44:14
<b>Session 7A</b>	
The Power and Limitations of Uniform Samples in Testing Properties of Figures <i>Piotr Berman, Meiram Murzabulatov, and Sofya Raskhodnikova</i> .....	45:1–45:14
Local Testing for Membership in Lattices <i>Karthekeyan Chandrasekaran, Mahdi Cheraghchi, Venkata Gandikota, and Elena Grigorescu</i> .....	46:1–46:14
Super-Fast MST Algorithms in the Congested Clique Using $o(m)$ Messages <i>Sriram V. Pemmaraju and Vivek B. Sardeshmukh</i> .....	47:1–47:15
<b>Session 7B</b>	
Why Liveness for Timed Automata Is Hard, and What We Can Do About It <i>Frédéric Herbreteau, B. Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz</i> .....	48:1–48:14
Verified Analysis of List Update Algorithms <i>Maximilian P. L. Haslbeck and Tobias Nipkow</i> .....	49:1–49:15
Tunable Online MUS/MSS Enumeration <i>Jaroslav Bendík, Nikola Beneš, Ivana Černá, and Jiří Barnat</i> .....	50:1–50:13

## ■ Preface

The 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016), organized annually by the Indian Association for Research in Computing Science (IARCS), was held at the Chennai Mathematical Institute, Chennai, from December 13 to December 15, 2016.

The program consisted of 6 invited talks and 44 contributed papers. This proceedings volume contains the contributed papers and abstracts of invited talks presented at the conference. The proceedings of FSTTCS 2016 is published as a volume in the LIPIcs series under a Creative Commons license, with free online access to all, and with authors retaining rights over their contributions.

The 44 contributed papers were selected from a total of 112 submissions. We thank the program committee for its efforts in carefully evaluating and making these selections. We thank all those who submitted their papers to FSTTCS 2016. We also thank the external reviewers for sending their informative and timely reviews.

We are particularly grateful to the invited speakers: Tefvik Bultan (University of California, Santa Barbara), Fedor V. Fomin (University of Bergen), Holger Hermanns (Saarland University), Aleksander Mądry (Massachusetts Institute of Technology), Mooly Sagiv (Tel Aviv University), and Mikkel Thorup (University of Copenhagen) who readily accepted our invitation to speak at the conference.

There was one pre-conference workshop, *Rangoli of Algorithms* (RoA) and one post-conference workshop, *Algorithmic Verification of Real-Time Systems* (AVeRTS). We thank Fedor V. Fomin (University of Bergen), Krishna S. (IIT Bombay), Saket Saurabh (IMSc & University of Bergen), Roohani Sharma (IMSc), Ashutosh Trivedi (University of Colorado, Boulder), and Meirav Zehavi (University of Bergen), for organizing these workshops.

On the administrative side, we thank the entire Computer Science Group, Chennai Mathematical Institute (CMI), who put in many months of effort in ensuring excellent conference and workshop arrangements at the Chennai Mathematical Institute.

We would also like to thank G. Ramalingam, Madhavan Mukund, S. P. Suresh, Supratik Chakraborty and Venkatesh Raman for promptly responding to our numerous questions and requests relating to the organization of the conference.

We also thank the Easychair team whose software has made it very convenient to do many conference related tasks. Finally, we thank the Dagstuhl LIPIcs staff for their coordination in the production of this proceedings, particularly Marc Herbstritt who was very prompt and helpful in answering our questions.

Akash Lal, S. Akshay, Saket Saurabh and Sandeep Sen  
December 2016





## ■ Conference Organization

### Programme Chairs

Akash Lal (MSR India)  
S. Akshay (IIT Bombay)  
Saket Saurabh (IMSc)  
Sandeep Sen (IIT Delhi)

### Programme Committee

Olaf Beyersdorff (University of Leeds)  
Umang Bhaskar (TIFR)  
Parinya Chalermsook (MPI)  
Arkadev Chattopadhyay (TIFR)  
Ajit Diwan (IIT Bombay)  
Samir Datta (CMI)  
Fabrizio Grandoni (IDSIA)  
Daniel Lokshtanov (University of Bergen)  
M. S. Ramanujan (TU Wien)  
Chandan Saha (IISc Bangalore)  
Mohit Singh (MSR, Redmond)  
Chaitanya Swamy (University of Waterloo)  
Kasturi R. Varadarajan (University of Iowa)  
Neal Young (University of California, Riverside)  
Patricia Bouyer (LSV Cachan)  
Krishnendu Chatterjee (IST Austria)  
Pedro D'Argenio (Universidad Nacional de Córdoba)  
Alastair Donaldson (Imperial College, London)  
Pranav Garg (Amazon India)  
Aditya Kanade (IISc Bangalore)  
Steve Kremer (LORIA, Nancy)  
Shuvendu Lahiri (MSR)  
Slawomir Lasota (University of Warsaw)  
Madhavan Mukund (CMI)  
Sophie Pinchinat (University of Rennes 1)  
M Praveen (CMI)  
Ashish Tiwari (SRI)  
James Worrell (University of Oxford)

### Organizing Committee

Computer Science Group,  
Chennai Mathematical Institute (CMI).







## ■ External Reviewers

Ágnes Cseh  
Akanksha Agrawal  
Amit Kumar  
Andreas Herzig  
Andrew Drucker  
Anindya De  
Antoine Durand-Gasselín  
Anton Freund  
Arindam Khan  
Arnab Bhattacharyya  
Barbara Koenig  
Benjamin Monmege  
C. Seshadhri  
Christoph Lenzen  
Eduard Eiben  
Eric Allender  
Ernst Moritz Hahn  
Filip Murlak  
Guilhem Jaber  
Holger Dell  
Ian Pratt-Hartmann  
Isolde Adler  
Jan Obdrzalek  
Janardhan Kulkarni  
Kamal Lodaya  
Keerti Choudhary  
Krishna S.  
Laurent Doyen  
Lin Yang  
Lorenzo Clemente  
Luc Dartois  
Magnus Wahlström  
Mamadou Moustapha Kanté  
Marco Faella  
Markus Lohrey  
Massimo Lauria  
Matthias Mnich  
Maximilien Colange  
Michael Walter  
Mohamed Faouzi Atig  
Nathanaël François  
Neeraj Kayal  
Nikhil Mande  
Nir Piterman  
Nutan Limaye  
Aiswarya Cyriac  
Amer Mouawad  
Amt Deshpande  
Andreas Pavlogiannis  
Anil Seth  
Anish Mukherjee  
Antoine Mottet  
Anuj Dawar  
Aritra Banik  
Arnaud Sangnier  
Barnaby Martin  
Bernhard Kragl  
Christoph Haase  
Deepak D'Souza  
Elliot Anshelevich  
Erik Jan van Leeuwen  
Fahad Panolan  
Geevarghese Philip  
Henning Fernau  
Hongfei Fu  
Irit Dinur  
Jakub Łącki  
Jan Otop  
Jeff Phillips  
Karl Bringmann  
Keren Censor-Hillel  
Laura Bozzelli  
Li Zhiwu  
Linda Farczadi  
Louigi Addario-Berry  
Lukasz Kowalik  
Mahsa Shirmohammadi  
Marcin Wrochna  
Marie Dufлот  
Martin Schuster  
Matthew Hague  
Maximilian Schlund  
Meirav Zehavi  
Mingyu Xiao  
Mrinal Kumar  
Neeldhara Misra  
Nikhil Balaji  
Nikola Benes  
Nitín Saurabh  
Ocan Sankur

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen



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

Oded Regev  
Omer Angel  
Paul Beame  
Paweł Parys  
Peter Habermehl  
Pierre-Malo Denielou  
Pooya Hatami  
R. Ramanujam  
Rajat Mittal  
Rakesh Venkat  
Rasmus Ibsen-Jensen  
Rohit Chadha  
S. P. Suresh  
Sandy Heydrich  
Sebastian Ordyniak  
Shi Li  
Shreyas Sekar  
Sourav Chakraborty  
Stasys Jukna  
Stefan Schwoon  
Stéphane Graham-Lengrand  
Steven Chaplick  
Supartha Podder  
Susmit Jha  
Sylvain Schmitz  
Tanmay Inamdar  
Telikepalli Kavitha  
Thomas Wies  
Tingting Han  
Travis Gagie  
Tyler Sorensen  
Vaishnavi Sundararajan  
Vladimir Braverman  
Wataru Matsubara  
Wojciech Plandowski  
Yuval Filmus  
Olivier Carton  
Partha Mukhopadhyay  
Paweł Gawrychowski  
Pc Discussion  
Petr Novotny  
Piyush Kurur  
Prakash Saivasan  
Raghunath Tewari  
Rajesh Chitnis  
Ramanathan Thinniyam  
Rob van Stee  
Rohit Gurjar  
Sahil Singla  
Sayan Bhattacharya  
Sebastien Tavenas  
Shinnosuke Seki  
Siddharth Barman  
Sreejith A. V.  
Stefan Göller  
Stephan Merz  
Stephen Chestnut  
Sumedha Uniyal  
Surender Baswana  
Syamantak Das  
Tamara Rezk  
Tatiana Romina Hartinger  
Thatchaphol Saranurak  
Tim Wood  
Tobias Friedrich  
Troy Lee  
V. Arvind  
Vikrant Vaze  
Vojtech Forejt  
Wojciech Czerwiński  
Wojciech Rytter  
Zhewei Wei

# Fast and Powerful Hashing Using Tabulation\*

Mikkel Thorup

University of Copenhagen, Dept. of Computer Science, Copenhagen, Denmark  
mikkel2thorup@gmail.com

---

## Abstract

---

Randomized algorithms are often enjoyed for their simplicity, but the hash functions employed to yield the desired probabilistic guarantees are often too complicated to be practical. Here we survey recent results on how simple hashing schemes based on tabulation provide unexpectedly strong guarantees.

*Simple tabulation hashing* dates back to Zobrist [1970]. Keys are viewed as consisting of  $c$  characters and we have precomputed character tables  $h_1, \dots, h_q$  mapping characters to random hash values. A key  $x = (x_1, \dots, x_c)$  is hashed to  $h_1[x_1] \oplus h_2[x_2] \dots \oplus h_c[x_c]$ . This scheme is very fast with character tables in cache. While simple tabulation is not even 4-independent, it does provide many of the guarantees that are normally obtained via higher independence, e.g., linear probing and Cuckoo hashing.

Next we consider *twisted tabulation* where one character is "twisted" with some simple operations. The resulting hash function has powerful distributional properties: Chernoff-Hoeffding type tail bounds and a very small bias for min-wise hashing.

Finally, we consider *double tabulation* where we compose two simple tabulation functions, applying one to the output of the other, and show that this yields very high independence in the classic framework of Carter and Wegman [1977]. In fact, w.h.p., for a given set of size proportional to that of the space consumed, double tabulation gives fully-random hashing.

While these tabulation schemes are all easy to implement and use, their analysis is not.

This invited talk surveys results from the papers in the reference list. The reader is referred to [8] for more details.

**1998 ACM Subject Classification** E.1 [Data Structures] Tables, E.2 [Data Storage Representations] Hash Table Representations, F.2.2 [Nonnumerical Algorithms and Problems] Sorting and Searching, H.3 [Information Search and Retrieval] Search Process

**Keywords and phrases** Hashing, Randomized Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.1

**Category** Invited Talk

---

## References

---

- 1 Tobias Christiani, Rasmus Pagh, and Mikkel Thorup. From independence to expansion and back again. In *Proceedings of the 47th ACM Symposium on Theory of Computing (STOC)*, pages 813–820, 2015.
- 2 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. The power of two choices with simple tabulation. In *Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1631–1642, 2016.

---

\* Research is partly supported by Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research.



© Mikkel Thorup;

licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 1; pp. 1:1–1:2



Leibniz International Proceedings in Informatics

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

- 3 Søren Dahlgaard and Mikkel Thorup. Approximately minwise independence with twisted tabulation. In *Proc. 14th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 134–145, 2014.
- 4 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. Hashing for statistics over k-partitions. In *Proceedings of the 56th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1292–1310, 2015.
- 5 Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation-based hashing. *Journal of the ACM*, 59(3):Article 14, 2012. Announced at STOC’11.
- 6 Mihai Pătraşcu and Mikkel Thorup. Twisted tabulation hashing. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–228, 2013.
- 7 Mikkel Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *Proc. 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 90–99, 2013.
- 8 Mikkel Thorup. Fast and powerful hashing using tabulation. *CoRR*, abs/1505.01523, 2015. Invited as a Research Highlight to *Comm. ACM*. URL: <http://arxiv.org/abs/1505.01523>.

# Simple Invariants for Proving the Safety of Distributed Protocols

Mooly Sagiv

Tel Aviv University, Tel Aviv, Israel  
mooly.sagiv@gmail.com

---

## Abstract

Safety of a distributed protocol means that the protocol never reaches a bad state, e.g., a state where two nodes become leaders in a leader-election protocol. Proving safety is obviously undecidable since such protocols are run by an unbounded number of nodes, and their safety needs to be established for any number of nodes. I will describe a deductive approach for proving safety, based on the concept of universally quantified inductive invariants – an adaptation of the mathematical concept of induction to the domain of programs. In the deductive approach, the programmer specifies a candidate inductive invariant and the system automatically checks if it is inductive. By restricting the invariants to be universally quantified, this approach can be effectively implemented with a SAT solver.

This is a joint work with Ken McMillan (Microsoft Research), Oded Padon (Tel Aviv University), Aurojit Panda (UC Berkeley), and Sharon Shoham (Tel Aviv University) and was integrated into the IVY system<sup>1</sup>. The work is inspired by Shachar Itzhaky's thesis<sup>2</sup>.

**1998 ACM Subject Classification** D.2.4 Software/Program Verification, Formal Methods

**Keywords and phrases** Program verification, Distributed protocols, Deductive reasoning

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.2

**Category** Invited Talk

---

<sup>1</sup> <http://microsoft.github.io/ivy/>

<sup>2</sup> <http://people.csail.mit.edu/shachari>



© Mooly Sagiv;

licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 2; pp. 2:1–2:1



Leibniz International Proceedings in Informatics

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



# My $\mathcal{O}$ Is Bigger Than Yours\*

Holger Hermanns

Saarland University, Saarland Informatics Campus, Saarbrücken, Germany  
hermanns@cs.uni-saarland.de

---

## Abstract

This invited talk starts off with a review of probabilistic safety assessment (PSA) methods currently exercised across the nuclear power plant domain worldwide. It then elaborates on crucial aspects of the Fukushima Dai-ichi accident which are not considered properly in contemporary PSA studies [6, 8, 7]. New kinds of PSA are needed so as to take into account external hazards, dynamic aspects of accident progression, and partial information. All of these come with obvious increases in algorithmic analysis complexity. This motivates our ongoing work to gradually tackle the resulting modelling and analysis problems. They revolve around static and dynamic fault trees [5, 1], open interpretations of compositional Markov models [2, 4] and advances in their effective numerical analysis [3].

**1998 ACM Subject Classification** F.1.1 Models of Computation

**Keywords and phrases** Probabilistic Safety Analysis, Fault Trees, Compositionality, Markov Models, Model Checking

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.3

**Category** Invited Talk

---

## References

- 1 Ola Bäckström, Yuliya Butkova, Holger Hermanns, Jan Krcál, and Pavel Krcál. Effective static and dynamic fault tree analysis. In Amund Skavhaug, Jérémie Guiochet, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security – 35th International Conference, SAFECOMP 2016, Trondheim, Norway, September 21-23, 2016, Proceedings*, volume 9922 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2016. doi:10.1007/978-3-319-45477-1\_21.
- 2 Tomáš Brázdil, Holger Hermanns, Jan Krcál, Jan Kretínský, and Vojtech Rehák. Verification of Open Interactive Markov Chains. In Deepak D’Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 474–485. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012. URL: <http://www.dagstuhl.de/dagpub/978-3-939897-47-7>, doi:10.4230/LIPIcs.FSTTCS.2012.474.
- 3 Yuliya Butkova, Hassan Hatefi, Holger Hermanns, and Jan Krcál. Optimal Continuous Time Markov Decisions. 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 166–182. Springer, 2015. doi:10.1007/978-3-319-24953-7\_12.
- 4 Holger Hermanns, Jan Krcál, and Jan Kretínský. Compositional Verification and Optimization of Interactive Markov Chains. In Pedro R. D’Argenio and Hernán C. Melgratti, editors,

---

\* This work has received partial supported through ERC Advanced Investigator Grant 695614 (POWVER).



© Holger Hermanns;

licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 3; pp. 3:1–3:2

Leibniz International Proceedings in Informatics



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

- 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 364–379. Springer, 2013. doi:10.1007/978-3-642-40184-8\_26.
- 5 Jan Krcál and Pavel Krcál. Scalable Analysis of Fault Trees with Dynamic Features. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22–25, 2015*, pages 89–100. IEEE Computer Society, 2015. URL: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7265894>, doi:10.1109/DSN.2015.29.
  - 6 A. Lyubarskiy, I. Kuzmina, and M. El-Shanawany. Notes on Potential Areas for Enhancement of the PSA Methodology based on Lessons Learned from the Fukushima Accident. In *UK’s 2nd Probabilistic Safety Analysis / Human Factors Assessment Forum, Warrington, UK*, 2011. URL: [https://nucleus.iaea.org/sites/gsan/lib/Lists/Papers/Attachments/1/Lessons%20from%20FA%20for%20PSA\\_UK-Forum\\_Sept2011.pdf](https://nucleus.iaea.org/sites/gsan/lib/Lists/Papers/Attachments/1/Lessons%20from%20FA%20for%20PSA_UK-Forum_Sept2011.pdf).
  - 7 ASME Presidential Task Force on Response to Japan Nuclear Power Plant Events. Forging a New Nuclear Safety Construct. Technical report, American Society of Mechanical Engineers, 2012. URL: [https://www.asme.org/getmedia/73081de8-e963-4557-9498-f856b56dabd1/Forging\\_a\\_new\\_nuclear\\_safety\\_construct.aspx](https://www.asme.org/getmedia/73081de8-e963-4557-9498-f856b56dabd1/Forging_a_new_nuclear_safety_construct.aspx).
  - 8 N. Siu, D. Marksberry, S. Cooper, K. Coyne, and M. Stutzke. PSA Technology Challenges Revealed by the Great East Japan Earthquake. In *PSAM Topical Conference In Light of the Fukushima Dai-ichi Accident, Tokyo, Japan*, 2013. URL: <http://www.nrc.gov/docs/ML1309/ML13099A347.pdf>.



# Continuous Optimization: The “Right” Language for Graph Algorithms?\*

Aleksander Mądry

MIT, Cambridge, MA, USA  
mądry@mit.edu

---

## Abstract

Traditionally, we view graphs as purely combinatorial objects and tend to design our graph algorithms to be combinatorial as well. In fact, in the context of algorithms, “combinatorial” became a synonym of “fast”.

Recent work, however, shows that a number of such “inherently combinatorial” graph problems can be solved much faster using methods that are very “non-combinatorial”. Specifically, by approaching these problems with tools and notions borrowed from linear algebra and, more broadly, from continuous optimization. A notable examples here are the recent lines of work on the maximum flow problem [5, 1, 4, 6, 9, 3, 8, 7, 2], the bipartite matching problem [6, 7, 2], and the shortest path problem in graphs with negative-length arcs [2].

This raises an intriguing question: Is continuous optimization a more suitable and principled optics for fast graph algorithms than the classic combinatorial view? In this talk, I will discuss this question as well as the developments that motivated it.

**1998 ACM Subject Classification** F.2 Analysis of Algorithms

**Keywords and phrases** maximum flow problem, bipartite matchings, interior-point methods, gradient decent method, Laplacian linear systems

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.4

**Category** Invited Talk

---

## References

- 1 Paul Christiano, Jonathan Kelner, Aleksander Mądry, Daniel Spielman, and Shang-Hua Teng. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In *STOC’11: Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 273–281, 2011.
- 2 Michael B. Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in  $\tilde{O}(m^{10/7} \log W)$  time. In *SODA’17: Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2017.
- 3 Jonathan Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *SODA’14: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 217–226, 2014.
- 4 Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A new approach to computing maximum flows using electrical flows. In *STOC’13: Proceedings of the 45th Annual ACM Symposium on the Theory of Computing*, pages 755–764, 2013.

---

\* This work was partially supported by NSF grant CCF-1553428, and Sloan Research Fellowship.



- 5 Aleksander Mądry. Fast approximation algorithms for cut-based problems in undirected graphs. In *FOCS'10: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 245–254, 2010.
- 6 Aleksander Mądry. Navigating central path with electrical flows: from flows to matchings, and back. In *FOCS'13: Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, pages 253–262, 2013.
- 7 Aleksander Mądry. Computing maximum flow with augmenting electrical flow. In *FOCS'16: Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, pages 593–602, 2016.
- 8 Richard Peng. Approximate undirected maximum flows in  $O(\text{mpolylog}(n))$  time. In *SODA'16: Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1862–1867, 2016.
- 9 Jonah Sherman. Nearly maximum flows in nearly linear time. In *FOCS'13: Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, pages 263–269, 2013.

# Graph Decompositions and Algorithms

Fedor V. Fomin

University of Bergen, Norway  
fomin@ii.uib.no

---

## Abstract

We overview the recent progress in solving intractable optimization problems on planar graphs as well as other classes of sparse graphs. In particular, we discuss how tools from Graph Minors theory can be used to obtain

- subexponential parameterized algorithms
- approximation algorithms, and
- preprocessing and kernelization algorithms

on these classes of graphs.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

**Keywords and phrases** Algorithms, logic, graph minor

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.5

**Category** Invited Talk



© Fedor V. Fomin;

licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 5; pp. 5:1–5:1



Leibniz International Proceedings in Informatics

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



# Side Channel Analysis Using a Model Counting Constraint Solver and Symbolic Execution\*

Tevfik Bultan

Dept. of Computer Science, University of California, Santa Barbara, CA, USA  
bultan@cs.ucsb.edu

---

## Abstract

A crucial problem in software security is the detection of side-channels [5, 2, 7]. Information gained by observing non-functional properties of program executions (such as execution time or memory usage) can enable attackers to infer secret information (such as a password). In this talk, I will discuss how symbolic execution, combined with a model counting constraint solver, can be used for quantifying side-channel leakage in Java programs. In addition to computing information leakage for a single run of a program, I will also discuss computation of information leakage for multiple runs for a type of side channels called segmented oracles [3]. In segmented oracles, the attacker is able to explore each segment of a secret (for example each character of a password) independently. For segmented oracles, it is possible to compute information leakage for multiple runs using only the path constraints generated from a single run symbolic execution. These results have been implemented as an extension to the symbolic execution tool Symbolic Path Finder (SPF) [8] using the SMT solver Z3 [4] and two model counting constraint solvers LattE [6] and ABC [1].

**1998 ACM Subject Classification** D.4.6 Security and Protection, Verification, D.2.4 Software/Program Verification, Formal Methods

**Keywords and phrases** Side-channels, quantitative information flow, symbolic execution, model counting, constraint solvers

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.6

**Category** Invited Talk

---

## References

- 1 Abdulkaki Aydin, Lucas Bang, and Tevfik Bultan. Automata-based model counting for string constraints. In *Proceedings of the 27th International Conference on Computer Aided Verification (CAV)*, pages 255–272, 2015.
- 2 Michael Backes, Boris Kopf, and Andrey Rybalchenko. Automatic Discovery and Quantification of Information Leaks. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy, SP'09*, pages 141–153, Washington, DC, USA, 2009. IEEE Computer Society.
- 3 Lucas Bang, Abdulkaki Aydin, Quoc-Sang Phan, Corina S. Pasareanu, and Tevfik Bultan. String analysis for side channels with segmented oracles. In *Proceedings of the 24th ACM*

---

\* This material is based on research sponsored by NSF under grant CCF-1548848 and by DARPA under agreement number FA8750-15-2-0087. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.



© Tevfik Bultan;

licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 6; pp. 6:1–6:2



Leibniz International Proceedings in Informatics

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

*SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*, 2016.

- 4 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008*, pages 337–340, 2008.
- 5 Boris Köpf and David A. Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 286–296, 2007.
- 6 Jesús A. De Loera, Raymond Hemmecke, Jeremiah Tauzer, and Ruriko Yoshida. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation*, 38(4):1273–1302, 2004. Symbolic Computation in Algebra and Geometry.
- 7 Corina S. Păsăreanu, Quoc-Sang Phan, and Pasquale Malacaria. Multi-run side-channel analysis using Symbolic Execution and Max-SMT. In *Proceedings of the 2016 IEEE 29th Computer Security Foundations Symposium, CSF'16, Washington, DC, USA, 2016*. IEEE Computer Society.
- 8 Corina S. Păsăreanu, Willem Visser, David Bushnell, Jaco Geldenhuys, Peter Mehlitz, and Neha Rungta. Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis. *Automated Software Engineering*, pages 1–35, 2013.

# Mixed-Criticality Scheduling to Minimize Makespan\*

Sanjoy Baruah<sup>1</sup>, Arvind Easwaran<sup>2</sup>, and Zhishan Guo<sup>3</sup>

1 University of North Carolina at Chapel Hill, USA  
baruah@cs.unc.edu

2 Nanyang Technological University, Singapore  
arvinde@ntu.edu.sg

3 Missouri University of Science and Technology, Rolla, USA  
zsguo@cs.unc.edu

---

## Abstract

In the mixed-criticality job model, each job is characterized by two execution time parameters, representing a smaller (less conservative) estimate and a larger (more conservative) estimate on its actual, unknown, execution time. Each job is further classified as being either less critical or more critical. The desired execution semantics are that all jobs should execute correctly provided all jobs complete upon being allowed to execute for up to the smaller of their execution time estimates, whereas if some jobs need to execute beyond their smaller execution time estimates (but not beyond their larger execution time estimates), then only the jobs classified as being more critical are required to execute correctly. The scheduling of collections of such mixed-criticality jobs upon identical multiprocessor platforms in order to minimize the makespan is considered here.

**1998 ACM Subject Classification** D.4.1 Scheduling, D.4.7 Real-time systems and embedded systems

**Keywords and phrases** Scheduling, Mixed criticality, Identical parallel machines, Makespan minimization, Approximation algorithm

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.7

## 1 Introduction and motivation

The problem studied in this paper has its genesis in a collaborative project between our universities-based research group and a major US defense contractor. The defense contractor is developing fleets of unmanned aerial vehicles (UAVs) that are capable of coordinating with one another autonomously in order to accomplish goals that are broadly specified at a relatively high level. The embedded computer control systems on board such UAVs are responsible for two general classes of functions:

1. safety-critical functions relating to the safe flight of the UAV – these functions are expected to be subject to mandatory certification by the US Federal Aviation Authority (FAA); and
2. mission-critical functions that enable the UAV to actually accomplish its stated mission. The mission-critical functions are not subject to certification (although our collaborator –

---

\* Work supported by NSF grants CNS 1115284, CNS 1218693, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, a grant from General Motors Corp., and a Tier-2 grant (ARC9/14) from the Ministry of Education, Singapore.



© Sanjoy Baruah, Arvind Easwaran, and Zhishan Guo;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 7; pp. 7:1–7:13



Leibniz International Proceedings in Informatics

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

## 7:2 Mixed-Criticality Scheduling to Minimize Makespan

the defense contractor manufacturing the UAV – will subject them to analysis using their own correctness criteria).

Systems such as this are *mixed-criticality* systems; in mixed criticality (MC) systems, functionalities of different degrees of importance (or *criticalities*) are implemented upon a common platform. As stated above, the more critical functionalities may be required to have their correctness validated to a higher level of assurance than less critical functionalities. This difference in correctness criteria may be expressed by different *Worst-Case Execution Time* (WCET) estimates for the execution of a piece of real-time code. For validating the timing correctness of critical functionalities it is desirable to use WCET estimates that are obtained using extremely conservative tools (for example, some certification standards require that particular “certified” tools based on static code-analysis be used for determining WCET of highly safety-critical code), while less critical functionalities are often validated using (less conservative) measurement-based WCET tools. Vestal [11, page 239] articulated the practical implication of such practices in this manner: “the more confidence one needs in a task execution time bound [...] the larger and more conservative that bound tends to become.” He proposed that each piece of code therefore be characterized by *multiple* WCET parameters, which are obtained by analyzing the (same) piece of code using different WCET-analysis tools and methodologies. Different sets of WCET estimates are then used to validate different correctness properties. We illustrate the essence of Vestal’s idea via a simple (contrived) example.

► **Example 1.** Consider two jobs  $J_1$  and  $J_2$  executing upon a shared processor, with job  $J_1$  being more critical than  $J_2$ . Both jobs are released at time 0, and share a common deadline at time 10. Let us suppose that the WCET of  $J_1$ , as determined by a more conservative WCET tool, is equal to 5, while the WCET of  $J_2$ , as determined using a less conservative WCET tool (since  $J_2$  is less critical), is equal to 6. Since the sum of these WCETs exceeds the duration between the jobs’ common release time and their deadline, conventional scheduling techniques cannot schedule both jobs to guarantee completion by their deadlines. However, Vestal observed in [11] that

- with regards to validating the more critical functionality (e.g., from the perspective of a certification process), it may be *irrelevant* whether the less critical job  $J_2$  completes on time or not; and
- assigning  $J_1$ ’s WCET parameter the value of 5 may be deemed *too conservative* for validating less critical functionalities.

Let us suppose that the WCET of  $J_1$  is estimated once again, this time using the less conservative WCET-determination tool;  $J_1$ ’s WCET is determined by this tool to be equal to 3 (rather than 5). If we were now to schedule the jobs by assigning  $J_1$  greater priority than  $J_2$ ,

- In validating the more critical functionalities, we would determine that  $J_1$  completes by time-instant 5 and hence meets its deadline.
- The validation process for less critical functionalities concludes that  $J_1$  completes by time-instant 3, and  $J_2$  by time-instant 9. Hence they both complete by the deadline.

We thus see that the system is deemed as being correct in both analyses, despite our initial observation that the sum of the relevant WCETs (5 for  $J_1$ ; 6 for  $J_2$ ) exceeds the duration between the jobs’ common release time and deadline.

The idea exposed in Example 1 – that the same system, represented using more and less conservative models, may be demonstrated to satisfy different correctness criteria for



functionalities of different criticalities – has been widely explored since first proposed by Vestal [11]; there is a nice review of the current state of the art in [3].

**This research.** Much of the prior study on mixed-criticality scheduling has focused upon the scheduling of mixed-criticality workloads that are executed upon a single processor. A few pieces of work (e.g., [2, 9, 8, 7]) have considered multiprocessor scheduling, but they have all dealt with a very different workload model: systems of *recurrent* (periodic or sporadic) *tasks* that need to meet deadlines, rather than collections of independent jobs. In this paper, we seek to initiate the study of mixed-criticality scheduling of collections of independent jobs upon multiprocessor platforms, by considering a simple multiprocessor scheduling problem for such workloads – that of scheduling a given collection of mixed-criticality jobs (each of the kind described in Example 1 above) upon a specified number of identical processors in order to minimize the makespan of the resulting schedule. Makespan minimization is one of the basic and fundamental problems studied in multiprocessor scheduling, and we are optimistic that obtaining a better understanding of this fundamental problem will facilitate the development of a more comprehensive theory of multiprocessor mixed-criticality scheduling. Although this specific mixed-criticality problem is a highly simplified version of the motivating application problem – it was obtained by applying a large number of simplifying assumptions to the actual application system under analysis – it is hoped that exposing this problem domain to the FST&TCS community will motivate further work upon less simple, but more realistic, variants.

**Our results.** We derive algorithms for both non-preemptive scheduling (in which a job, once it begins execution, is allowed to execute through to completion upon the same processor on which it started to execute), and preemptive scheduling (in which an executing job may be preempted during execution, and its execution resumed later upon any processor) of collections of mixed-criticality jobs to minimize makespan upon identical multiprocessor platforms. The non-preemptive problem is NP-hard, but can be solved approximately in polynomial time to any desired degree of accuracy by a polynomial-time approximation scheme. We do not yet know whether or not the preemptive version of the problem is solvable in polynomial time; we derive here a polynomial-time  $\frac{4}{3}$ -rds-approximation algorithm for solving it. To our knowledge, the precise computational complexity of determining the minimum makespan under preemptive scheduling remains open.

## 2 System model

In this section we formally define the semantics of the mixed-criticality model and specify the problem that we are seeking to solve. An instance  $I$  of the scheduling problem we consider is specified as follows.

1. A collection  $\mathcal{J}$  of  $n$  mixed-criticality jobs  $J_1, J_2, \dots, J_n$ . Each job  $J_i$  is characterized by the parameters  $(\chi_i, c_i^L, c_i^H)$ , with  $\chi_i \in \{\text{LO}, \text{HI}\}$  and  $c_i^L \leq c_i^H$ . The  $\chi_i$  parameter denotes the *criticality* of job  $J_i$ ; a job  $J_i$  with  $\chi_i = \text{LO}$  is called a LO-criticality job, and one with  $\chi_i = \text{HI}$  is called a HI-criticality job. The parameters  $c_i^L$  and  $c_i^H$  are the *LO-criticality WCET estimate* and the *HI-criticality WCET estimate* of job  $J_i$ ; since the LO-criticality WCET estimates are assumed to be made using a less conservative tool than the HI-criticality WCET estimates, we require that  $c_i^L \leq c_i^H$  for all  $J_i \in \mathcal{J}$ . (For LO-criticality jobs, we assume that  $c_i^L = c_i^H$ .)
2. A number  $m$  of unit-speed processors upon which the jobs in  $\mathcal{J}$  are to be executed.

## 7:4 Mixed-Criticality Scheduling to Minimize Makespan

Some additional notation: let  $\mathcal{J}_H \subseteq \mathcal{J}$  denote the collection of all the jobs  $J_i \in \mathcal{J}$  for which  $\chi_i = \text{HI}$ , and  $\mathcal{J}_L \subseteq \mathcal{J}$  denote the collection of all the jobs  $J_i \in \mathcal{J}$  for which  $\chi_i = \text{LO}$ .

**System behavior.** Our mixed-criticality model has the following semantics. Each job  $J_i$  is released at time 0, and needs to execute for a total duration  $\gamma_i$ . This execution must be sequential, meaning  $J_i$  is not allowed to simultaneously execute on more than one processor. The value of  $\gamma_i$  is not known prior to running time; it can only be discovered by actually allowing  $J_i$  to execute until it *signals* that it has completed execution. These values  $\langle \gamma_1, \gamma_2, \dots, \gamma_n \rangle$  upon a particular execution of the collection of jobs  $\mathcal{J}$  together define the kind of *behavior* exhibited by  $\mathcal{J}$  during that execution.

- If  $\gamma_i \leq c_i^L$  for each  $i$  (i.e., each  $J_i$  signals completion without exceeding  $c_i^L$  units of execution),  $\mathcal{J}$  is said to have exhibited *LO-criticality behavior*.
- If  $c_i^L < \gamma_i \leq c_i^H$  for any  $i$  (i.e., some job  $J_i$  only signals completion upon executing for more than  $c_i^L$  but no more than  $c_i^H$  units of execution),  $\mathcal{J}$  is said to have exhibited *HI-criticality behavior*.
- If  $c_i^H < \gamma_i$  for any  $i$  (i.e., some job  $J_i$  does not signal completion despite having executed for  $c_i^H$  units),  $\mathcal{J}$  is said to have exhibited *erroneous behavior*.

**Correctness criteria.** We define an algorithm for scheduling mixed-criticality instances to be correct if it is able to schedule any instance in such a manner that (i) during all LO-criticality behaviors of the instance, all jobs receive enough execution to be able to signal completion; and (ii) during all HI-criticality behaviors of the instance, all HI-criticality jobs receive enough execution to be able to signal completion. This is formally stated in the following definition:

► **Definition 2 (MC-correct).** A scheduling algorithm for mixed-criticality instances is MC-correct if it ensures that:

- during any execution of an instance in which it exhibits LO-criticality behavior, all jobs signal completion; and
- during any execution of an instance in which it exhibits HI-criticality behavior, all HI-criticality jobs signal completion (although LO-criticality jobs may fail to do so).

We point out that upon some job failing to signal completion despite having executed for up to its LO-criticality WCET, (i) an MC-correct scheduling algorithm may immediately discard all LO-criticality jobs; and (ii) only those HI-criticality jobs that have not already signaled completion may need to execute for up to their HI-criticality WCETs – those that have already signaled completion (upon executing for  $\leq$  their LO-criticality WCET) do not now need further execution.

**Problem statement.** Given an instance  $I$  comprising a collection  $\mathcal{J}$  of  $n$  mixed-criticality jobs to be scheduled upon  $m$  unit-speed processors, obtain an MC-correct scheduling algorithm that minimizes the makespan of the resulting schedule.

Since the instance  $I$  may generate arbitrarily many different behaviors (the values of the actual running times  $\langle \gamma_1, \gamma_2, \dots, \gamma_n \rangle$ ) during different executions, we clarify what we mean by minimizing makespan: *we desire that the maximum makespan over all non-erroneous behaviors of the instance be minimized.*

### 3 MC-correct scheduling algorithms that minimize makespan

Observe that the maximum amount of execution that could be required in any LO-criticality behavior is equal to  $(\sum_{J_i \in \mathcal{J}} c_i^L)$ , while in any HI-criticality behavior in which each HI-criticality job executes to its HI-criticality WCET, the amount of execution that must occur is at least equal to  $(\sum_{J_i \in \mathcal{J}_H} c_i^H)$ . It is therefore evident that upon an  $m$ -processor platform,

$$\frac{\max \{ \sum_{J_i \in \mathcal{J}} c_i^L, \sum_{J_i \in \mathcal{J}_H} c_i^H \}}{m}$$

is a lower bound on this desired makespan. An obvious upper bound on the makespan is given by

$$\sum_{J_i \in \mathcal{J}} c_i^H .$$

If we had a procedure for validating whether mixed-criticality instance  $I$  could be scheduled by some MC-correct scheduling algorithm with a makespan no larger than some specified constant, we could use bisection search ("binary search") between the upper and lower makespan bounds obtained above, in order to determine the minimum makespan to any desired degree of accuracy. In the remainder of this section, we will therefore attempt to design MC-correct scheduling algorithms that generate schedules with makespan no greater than some specified constant  $D$ .

#### 3.1 Non-preemptive scheduling

The non-preemptive version of this problem is easily seen to be solved by transforming it to a two-dimensional *vector scheduling problem* [12], for which a PTAS is known [4, 5]. The transformation is fairly straightforward: given an instance  $I$  of the mixed-criticality scheduling problem comprising the  $n$  jobs  $\mathcal{J}$  to be scheduled upon  $m$  processors with a makespan  $\leq D$ , we seek to partition  $\mathcal{J}$  into the sub-sets  $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_m$  satisfying the constraints that for each  $j$ ,  $1 \leq j \leq m$ ,

$$\left( \sum_{J_i \in \mathcal{J}_j} c_i^L \leq D \right) \text{ and } \left( \sum_{J_i \in \mathcal{J}_j \wedge \chi_i = \text{HI}} c_i^H \leq D \right) .$$

If such a partitioning is found, then during run-time we would execute the HI-criticality jobs in  $\mathcal{J}_j$  upon the  $j$ 'th processor first for each  $j$ ,  $1 \leq j \leq m$ . If each job  $J_i$  completes within  $c_i^L$  units of execution, then we execute the LO-criticality jobs in  $\mathcal{J}_j$  next upon the  $j$ 'th processor for each  $j$ , while if some  $J_i$  does not complete within  $c_i^L$  units of execution we simply discard the LO-criticality jobs and execute the HI-criticality jobs each to completion.

Observe that obtaining such a partitioning is equivalent to

1. First representing each job  $J_i$  by a 2-dimensional vector of size  $c_i^L$  along the first dimension, and size along the second dimension depending upon the value of  $\chi_i$ : if  $\chi_i = \text{HI}$  then the size along this dimension is set equal to  $c_i^H$  while if  $\chi_i = \text{LO}$  then the size along this dimension is set equal to zero.
2. Next, partitioning the  $n$  vectors so obtained into  $m$  sub-sets, such that each partition sums to  $\leq D$  along each of the two dimensions – this is exactly the 2-dimensional vector scheduling problem of [12].

The non-preemptive version of our scheduling problem, which is easily seen to be NP-hard (since the specialization of the problem to "regular" – non mixed-criticality – scheduling, in which all the jobs in  $\mathcal{J}$  are of the same criticality, is already known to be NP-hard), is thus solvable in polynomial time to any desired degree of accuracy by a PTAS.

- Each  $J_i$  is initially executed at a constant rate  $\phi_i^L$
- If some  $J_i$  does not signal completion despite having received  $c_i^L$  units of execution, then
  - All LO-criticality jobs are immediately discarded, and
  - Each HI-criticality job henceforth executes at a constant rate  $\phi_i^H$  until completion
- **Figure 1** Our preemptive run-time scheduling algorithm.

### 3.2 Preemptive scheduling

In contrast to the regular (i.e., not mixed-criticality) case, where preemptive scheduling of independent jobs to minimize makespan is easily seen to be solvable optimally in polynomial time using McNaughton’s rule [10], this problem turns out to be surprisingly challenging for mixed-criticality instances. Indeed, we have not yet been able to determine whether the problem is solvable in polynomial time or not; what we have instead is a polynomial-time approximation algorithm with approximation factor  $4/3$  for solving this problem<sup>1</sup>.

Given instance  $I$  and a desired makespan  $D$ , our strategy, which is based upon an algorithm called MC-Fluid [8, 1] for scheduling mixed-criticality sporadic tasks in order to meet all deadlines, is as follows. We will seek to determine a schedule for the jobs in  $\mathcal{J}$  upon the  $m$  unit-speed processors under a *fluid scheduling* model, which allows for schedules in which individual jobs may be assigned a fraction  $\leq 1$  of a processor (rather than an entire processor, or none) at each instant in time, subject to the constraint that the sum of the fractions assigned to all the jobs do not exceed  $m$  at any instant. That is, we will determine *execution rates*  $\phi_i^L$  and  $\phi_i^H$  for each task  $\tau_i$  such that the scheduling algorithm depicted in Figure 1 constitutes an MC-correct scheduling strategy for the jobs in  $\mathcal{J}$  upon  $m$  processors. (Standard techniques are known for converting such a fluid schedule to schedules in which there is no processor-sharing; see, e.g., [6, page 116] for details.)

We will now describe how the values for the  $\phi_i^L$  and  $\phi_i^H$  parameters are determined. We start out defining some additional notation:

- For each job  $J_i \in \mathcal{J}$ , let *flow rates*  $f_i^L$  and  $f_i^H$  be defined as follows:

$$\begin{aligned} f_i^L &\stackrel{\text{def}}{=} c_i^L / D \\ f_i^H &\stackrel{\text{def}}{=} c_i^H / D \end{aligned}$$

Intuitively, a fluid schedule in which  $J_i$  is executed at a constant rate  $f_i^L$  ( $f_i^H$ , respectively), over the interval  $[0, D]$  will complete at or before time-instant  $D$  in any LO-criticality behavior (HI-criticality behavior, resp.) of the instance. It should be evident that it is necessary that  $f_i^L$  be  $\leq 1$  for each job  $J_i$ , and that  $f_i^H$  be  $\leq 1$  for each HI-criticality job  $J_i$ , if we are to be able to guarantee a makespan  $D$ .

- Various *cumulative flow requirements* are defined for  $\mathcal{J}$  as follows – here,  $F_L^L$  denotes the cumulative LO-criticality flow rates of all LO-criticality jobs;  $F_H^L$  denotes the cumulative LO-criticality flow rates of all HI-criticality jobs; and  $F_H^H$  denotes the cumulative HI-criticality

<sup>1</sup> A trivial algorithm with approximation factor 2 can be obtained using McNaughton’s rule as follows. First schedule the HI-criticality jobs based on their HI-criticality WCET estimates in the time interval  $[0, D]$ , and then schedule the LO-criticality jobs based on their LO-criticality WCET estimates in the time interval  $[D, 2D]$ .

1. Define a *scaling factor*  $\rho$  as follows:

$$\rho \leftarrow \max \left\{ \left( \frac{F_L^L + F_H^L}{m} \right), \left( \frac{F_H^H}{m} \right), \max_{J_i \in \mathcal{J}_H} \{f_i^H\} \right\} \quad (1)$$

2. **If**  $\rho > 1$  **then** declare failure; **else** assign values to the execution-rate variables as follows:
 
$$\phi_i^H \leftarrow (f_i^H / \rho) \text{ for all } J_i \in \mathcal{J}_H \quad (2)$$

$$\phi_i^L \leftarrow \begin{cases} \frac{f_i^L}{\phi_i^H - (f_i^H - f_i^L)} \times \phi_i^H, & \text{if } J_i \in \mathcal{J}_H \\ f_i^L, & \text{else (i.e., if } J_i \in \mathcal{J}_L) \end{cases} \quad (3)$$

3. **If**

$$\sum_{J_i \in \mathcal{J}} \phi_i^L \leq m \quad (4)$$

**then** declare success **else** declare failure

■ **Figure 2** Computing execution rates.

flow rates of all HI-criticality jobs:

$$\begin{aligned} F_L^L &\stackrel{\text{def}}{=} \sum_{J_i \in \mathcal{J}_L} f_i^L \\ F_H^L &\stackrel{\text{def}}{=} \sum_{J_i \in \mathcal{J}_H} f_i^L \\ F_H^H &\stackrel{\text{def}}{=} \sum_{J_i \in \mathcal{J}_H} f_i^H \end{aligned}$$

The following observation directly follows from the definitions of  $F_L^L$ ,  $F_H^L$ , and  $F_H^H$ :

► **Observation 3.** *It is necessary that  $(F_L^L + F_H^L) \leq m$ , and  $F_H^H \leq m$ , if we are to be able to guarantee a makespan  $D$  for  $\mathcal{J}$  upon  $m$  processors.*

As stated in Figure 1, our run-time scheduling algorithm requires that values be assigned to the execution-rate variables  $\{\phi_i^L\}_{J_i \in \mathcal{J}} \cup \{\phi_i^H\}_{J_i \in \mathcal{J}_H}$  prior to run-time. In Figure 2 we describe, in pseudo-code form, the algorithm for computing the values of these execution-rate variables. Before proving the correctness of this algorithm (in Section 3.3), we first illustrate its application via an example.

**An example.** Consider the following collection of 4 mixed-criticality jobs, to be scheduled preemptively upon a 2-processor platform with a target makespan  $D \leftarrow 10$ .

	$\chi_i$	$c_i^L$	$c_i^H$
$J_1$	HI	3	8
$J_2$	HI	4	7
$J_3$	HI	1	1
$J_4$	LO	5	5

## 7:8 Mixed-Criticality Scheduling to Minimize Makespan

The flow rates for the jobs are obtained by dividing the corresponding WCET parameters by 10 (the value of  $D$ ); the cumulative flow requirements are then computed as follows:

$$\begin{aligned} F_L^L &= f_4^L = \mathbf{0.5} \\ F_H^L &= f_1^L + f_2^L + f_3^L = 0.3 + 0.4 + 0.1 = \mathbf{0.8} \\ F_H^H &= f_1^H + f_2^H + f_3^H = 0.8 + 0.7 + 0.1 = \mathbf{1.6} \end{aligned}$$

The scaling factor  $\rho$  is therefore

$$\begin{aligned} \rho &= \max\left\{\frac{0.5 + 0.8}{2}, \frac{1.6}{2}, \max\{0.8, 0.7, 0.1\}\right\} \\ &= \max\{1.3/2, 1.6/2, 0.8\} \\ &= \mathbf{0.8} \end{aligned}$$

The HI-criticality jobs  $J_1, J_2$ , and  $J_3$ , are assigned  $\phi_i^H$  values as follows:

$$\begin{aligned} \phi_1^H &= \frac{0.8}{0.8} = 1.0 \\ \phi_2^H &= \frac{0.7}{0.8} = 0.875 \\ \text{and } \phi_3^H &= \frac{0.1}{0.8} = 0.125 \end{aligned}$$

All the jobs are assigned  $\phi_i^L$  values as follows:

$$\begin{aligned} \phi_1^L &= \frac{1.0 \times 0.3}{1.0 - (0.8 - 0.3)} = 0.6 \\ \phi_2^L &= \frac{0.875 \times 0.4}{0.875 - (0.7 - 0.4)} = \frac{14}{23} < 0.61 \\ \phi_3^L &= \frac{0.125 \times 0.1}{0.125 - (0.1 - 0.1)} = 0.1 \\ \text{and } \phi_4^L &= 0.5 \end{aligned}$$

Since  $\sum_{i=1}^4 \phi_i^L < (0.6 + 0.61 + 0.1 + 0.5) = 1.81$ , which is  $\leq 2$  (the number of processors), our algorithm declares success.

### 3.3 Preemptive scheduling – proof of correctness

We will now show that the preemptive scheduling algorithm described above is correct: if the execution rates are computed as specified in Figure 2 without declaring failure for a given instance  $I$ , then the schedule resulting from using these execution rates in the manner described in Figure 1 does indeed constitute an MC-correct scheduling algorithm that always generates schedules of makespan  $\leq D$  upon all non-erroneous behaviors. Our proof proceeds in several steps.

1. We first prove, in Lemma 4 below, that the rate-assignment in Figure 2 is correct, by showing that the sum of the LO-criticality and the HI-criticality execution rates assigned to the jobs in Figure 2 do not exceed  $m$ , the number of available processors.
2. Next, we show in Lemma 5 that each execution rate is assigned a valid value in Figure 2: a non-negative real number that is no larger than one.
3. We then prove, in Lemma 6, correctness upon all LO-criticality behaviors, by showing that the  $\phi_i^L$  values assigned in Figure 2 are no smaller than the corresponding  $f_i^L$  values.

4. Finally, we prove correctness upon all HI-criticality behaviors by examining the actions of the scheduling algorithm in the event that some job does not signal completion despite having executed for up to its LO-criticality WCET (which indicates that the instance is exhibiting a HI-criticality behavior rather than a LO-criticality one).

► **Lemma 4.** *The sum of the LO-criticality execution rates assigned to all the jobs, and the sum of the HI-criticality rates assigned to all the HI-criticality jobs (i.e., the jobs in  $\mathcal{J}_H$ ), each does not exceed the number of processors  $m$ .*

**Proof.** It follows from Condition 4 of Figure 2 that our algorithm declares success only if the assigned LO-criticality execution rates sum to  $\leq m$ .

To show that the assigned HI-criticality rates also sum to no more than  $m$ , observe that

$$\sum_{J_i \in \mathcal{J}_H} \phi_i^H = \sum_{J_i \in \mathcal{J}_H} \frac{f_i^H}{\rho} \quad (\text{By Eqn 2}) = \frac{1}{\rho} F_H^H \quad (\text{By definition of } F_H^H)$$

By Equation 1,  $\rho \geq (F_H^H/m)$ ; hence

$$\frac{1}{\rho} F_H^H \leq \left(\frac{m}{F_H^H}\right) F_H^H = m$$

and the lemma is proved. ◀

► **Lemma 5.** *Each  $\phi_i^L$  and  $\phi_i^H$  is assigned a value  $\leq 1$  in the algorithm of Figure 2.*

**Proof.** Observe that Line 1 of Figure 2 assigns  $\rho$  a value  $\geq f_i^H$  for all  $J_i \in \mathcal{J}_H$ . Since Line 2 of Figure 2 assigns each  $\phi_i^H$  a value  $f_i^H/\rho$ , it follows that each such  $\phi_i^H$  has a value  $\leq 1$ , as required.

With regards to the  $\phi_i^L$ 's, the value assigned to  $\phi_i^L$  for each  $J_i \in \mathcal{J}_L$  is equal to  $f_i^L$  (and hence  $\leq 1$ ).

For each  $J_i \in \mathcal{J}_H$ , we will now show that  $\phi_i^L \leq \phi_i^H$ . It follows from the assignment of values to  $\phi_i^L$  (Equation 3 in Figure 2) that this will hold provided (Removed the justification in the last derivation step.)

$$\begin{aligned} \frac{f_i^L}{\phi_i^H - (f_i^H - f_i^L)} &\leq 1 \\ \Leftrightarrow f_i^L &\leq \phi_i^H - (f_i^H - f_i^L) \\ \Leftrightarrow f_i^H &\leq \phi_i^H \end{aligned}$$

which follows from the requirement that  $\rho$  be  $\leq 1$  (else, the algorithm in Figure 2 would declare failure in Step 2).

We have thus shown that  $\phi_i^L \leq \phi_i^H$  for each  $J_i \in \mathcal{J}_H$  (i.e., the execution rate guaranteed to each HI-criticality job does not decrease upon identification of HI-criticality behavior). Since we saw above that all such  $\phi_i^H$  values are  $\leq 1$ , it follows that the  $\phi_i^L$  variables are also assigned values  $\leq 1$ . ◀

► **Lemma 6.** *The instance is scheduled with a makespan  $\leq D$  in all LO-criticality behaviors.*

**Proof.** We will first prove that for each  $J_i \in \mathcal{J}$

$$\phi_i^L \geq f_i^L \tag{5}$$

Let us separately consider jobs in  $\mathcal{J}_L$  and  $\mathcal{J}_H$ . Observe that by the definition of  $\phi_i^L$  (Equation 3),

## 7:10 Mixed-Criticality Scheduling to Minimize Makespan

1. For each  $J_i \in \mathcal{J}_L$ ,  $\phi_i^L = f_i^L$ .
2. For each  $J_i \in \mathcal{J}_H$ ,

$$\begin{aligned}\phi_i^L &= f_i^L \times \frac{\phi_i^H}{\phi_i^H - (f_i^H - f_i^L)} \\ &\geq f_i^L \times \frac{\phi_i^H}{\phi_i^H} \quad (\text{Since } (f_i^H - f_i^L) \geq 0) \\ &= f_i^L\end{aligned}$$

These two cases together establish that  $\phi_i^L \geq f_i^L$  for all  $J_i \in \mathcal{J}$ ; it hence immediately follows that  $\phi_i^L \cdot D$ , the amount of execution that would be received by  $J_i$  if it were allowed to execute at a rate  $\phi_i^L$  over the entire duration  $[0, D)$  in any LO-criticality behavior of  $\mathcal{J}$ , is  $\geq c_i^L$ . From this we conclude that the makespan in any LO-criticality behavior is  $\leq D$ . ◀

► **Lemma 7.** *The instance is scheduled with a makespan  $\leq D$  in all HI-criticality behaviors.*

**Proof.** Consider any HI-criticality behavior of the instance, and let  $t_o$  denote the first time-instant at which some job does not signal completion despite having executed for its LO-criticality WCET. We will prove below that any HI-criticality job that is active (i.e., that has not yet completed execution) at time-instant  $t_o$  receives an amount of execution no smaller than its HI-criticality WCET by time-instant  $D$ .

Suppose that HI-criticality job  $J_i$  is active at time-instant  $t_o$ . Over the interval  $[0, t_o)$ , this job will have received an amount of execution equal to  $\phi_i^L \times t_o$ ; since the job is still active, it must be the case that

$$t_o \leq c_i^L / \phi_i^L \tag{6}$$

Henceforth job  $J_i$  will execute at a rate  $\phi_i^H$ . Hence for it to complete within a makespan  $D$ , it is sufficient that

$$\begin{aligned}t_o \phi_i^L + (D - t_o) \phi_i^H &\geq c_i^H \\ \Leftrightarrow D \phi_i^H - t_o(\phi_i^H - \phi_i^L) &\geq c_i^H \\ \Leftrightarrow D \phi_i^H - \frac{c_i^L}{\phi_i^L}(\phi_i^H - \phi_i^L) &\geq c_i^H \quad (\text{By Inequality 6}) \\ \Leftrightarrow D \phi_i^H &\geq \frac{c_i^L}{\phi_i^L}(\phi_i^H - \phi_i^L) + c_i^H \\ \Leftrightarrow D \phi_i^H &\geq \frac{c_i^L \phi_i^H}{\phi_i^L} - c_i^L + c_i^H \\ \Leftrightarrow D &\geq \frac{c_i^L}{\phi_i^L} + \left( \frac{c_i^H - c_i^L}{\phi_i^H} \right) \\ \Leftrightarrow 1 &\geq \frac{f_i^L}{\phi_i^L} + \left( \frac{f_i^H - f_i^L}{\phi_i^H} \right) \quad (\text{Dividing by } D, \text{ and applying definitions of } f_i^L, f_i^H) \quad (7)\end{aligned}$$

Also by Equation 3, for each  $J_i \in \mathcal{J}_H$  we have

$$\begin{aligned}\phi_i^L &= \frac{f_i^L \phi_i^H}{\phi_i^H - (f_i^H - f_i^L)} \\ \Leftrightarrow \frac{\phi_i^H - (f_i^H - f_i^L)}{\phi_i^H} &= \frac{f_i^L}{\phi_i^L}\end{aligned}$$



$$\begin{aligned} \Leftrightarrow 1 - \left( \frac{f_i^H - f_i^L}{\phi_i^H} \right) &= \frac{f_i^L}{\phi_i^L} \\ \Leftrightarrow 1 &= \frac{f_i^L}{\phi_i^L} + \left( \frac{f_i^H - f_i^L}{\phi_i^H} \right) \end{aligned}$$

thereby establishing Condition 7 and completing the proof of the lemma.  $\blacktriangleleft$

### 3.4 Preemptive scheduling – A 4/3’rds approximation Bound

We now prove that MC-correct scheduling algorithm described in Section 3.2 is a 4/3’rds approximate algorithm for preemptive scheduling to minimize makespan. Our approach towards showing this is as follows. A straightforward generalization of Observation 3 leads us to conclude that for mixed-criticality instance  $\mathcal{J}$  to be schedulable with makespan  $s \times D$  upon  $m$  processors, it is necessary that  $(F_L^L + F_H^L)$  and  $F_H^H$  for the instance both be  $\leq m \times s$ , and that in addition  $f_i^H \leq s$  for each  $J_i \in \mathcal{J}_H$  and  $f_i^L \leq s$  for each  $J_i \in \mathcal{J}_L$ . It therefore follows that the scaling factor  $\rho$  that is computed in Expression 1 of the algorithm of Figure 2 for such a system is  $\leq s$ . We will show below, in Lemma 9, that if  $\rho \leq 3/4$  and the  $\phi_i^H, \phi_i^L$  values are computed as specified in Expressions 2–3 of Figure 2, then the  $\phi_i^L$ ’s so computed are guaranteed to sum to  $\leq m$  and therefore satisfy Condition 4 of Figure 2 (which in turn means that the system is scheduled with makespan  $\leq D$  upon  $m$  processors). The approximation ratio follows, by observing that 4/3 is the multiplicative inverse of 3/4.

First, a technical lemma.

► **Lemma 8.** *Let  $c$  denote any positive constant. The function*

$$f(x) \stackrel{\text{def}}{=} \frac{x(c-x)}{\frac{c}{3} + x}$$

*is  $\leq \frac{c}{3}$  for all values of  $x \in [0, c]$ .*

**Proof.** (This lemma is easily proved rigorously using standard techniques from the calculus; we skip the details here in favor of a high-level outline.) Taking the derivative of  $f(x)$  with respect to  $x$ , we see that the only value of  $x \in [0, c]$  where this derivative equals zero is  $x \leftarrow c/3$ . We therefore conclude that  $f(x)$  takes on its maximum value over  $[0, c]$  for one of the values of  $x \in \{0, c/3, c\}$ . Explicit computation of  $f(x)$  at each of these values reveals that the value is maximized at  $x = c/3$ , where it takes on the value  $c/3$ .  $\blacktriangleleft$

► **Lemma 9.** *If  $\rho \leq 3/4$  and  $\phi_i^H, \phi_i^L$  values are computed as specified in Expressions 2–3 of Figure 2, then the  $\phi_i^L$  values so computed satisfy Condition 4.*

**Proof.** Let us first rewrite Condition 4 to an equivalent form expressed in Condition 8 below.

$$\begin{aligned} \sum_{J_i \in \mathcal{J}} \phi_i^L &\leq m \\ \Leftrightarrow \sum_{J_i \in \mathcal{J}_L} \phi_i^L + \sum_{J_i \in \mathcal{J}_H} \phi_i^L &\leq m \\ \Leftrightarrow F_L^L + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L \phi_i^H}{\phi_i^H - (f_i^H - f_i^L)} &\leq m \end{aligned}$$

## 7:12 Mixed-Criticality Scheduling to Minimize Makespan

$$\begin{aligned}
&\Leftrightarrow F_L^L + \sum_{J_i \in \mathcal{J}_H} f_i^L \left( 1 + \frac{(f_i^H - f_i^L)}{\phi_i^H - (f_i^H - f_i^L)} \right) \leq m \\
&\Leftrightarrow F_L^L + \sum_{J_i \in \mathcal{J}_H} f_i^L + \sum_{\tau_i \in \mathcal{J}_H} \frac{f_i^L (f_i^H - f_i^L)}{\phi_i^H - (f_i^H - f_i^L)} \leq m \\
&\Leftrightarrow F_L^L + F_H^L + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L (f_i^H - f_i^L)}{\phi_i^H - (f_i^H - f_i^L)} \leq m \tag{8}
\end{aligned}$$

We will show, in the remainder of this proof, that if  $\rho \leq 3/4$  then Condition 8 is satisfied; this will serve to establish the correctness of Lemma 9.

Let us assume henceforth that  $\rho \leq 3/4$ . From the definition of  $\rho$  (Expression 1), it follows that

$$F_L^L + F_H^L \leq \frac{3}{4}m \tag{9}$$

$$F_H^H \leq \frac{3}{4}m \tag{10}$$

$$\forall J_i \in \mathcal{J}_H \quad f_i^H \leq \frac{3}{4}m \tag{11}$$

Additionally, since  $\phi_i^H \leftarrow f_i^H / \rho$ , it must hold that

$$\forall J_i \in \mathcal{J}_H \quad \phi_i^H \geq \frac{4}{3}f_i^H \tag{12}$$

Let us use Inequalities 9–12 to further simplify Condition 8.

$$\begin{aligned}
&F_L^L + F_H^L + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L (f_i^H - f_i^L)}{\phi_i^H - (f_i^H - f_i^L)} \leq m \\
&\Leftrightarrow \frac{3}{4}m + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L (f_i^H - f_i^L)}{\phi_i^H - (f_i^H - f_i^L)} \leq m \text{ (By Ineq. 9)} \\
&\Leftrightarrow \frac{3}{4}m + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L (f_i^H - f_i^L)}{\frac{4}{3}f_i^H - (f_i^H - f_i^L)} \leq m \text{ (By Ineq. 12)} \\
&\Leftrightarrow \frac{3}{4}m + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L (f_i^H - f_i^L)}{\frac{f_i^H}{3} + f_i^L} \leq m \\
&\Leftrightarrow \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L (f_i^H - f_i^L)}{\frac{f_i^H}{3} + f_i^L} \leq \frac{m}{4} \\
&\Leftrightarrow \sum_{J_i \in \mathcal{J}_H} \frac{f_i^H}{3} \leq \frac{m}{4} \text{ (By Lemma 8)} \\
&\Leftrightarrow \frac{1}{3} \cdot (F_H^H) \leq \frac{m}{4} \\
&\Leftrightarrow \left( \frac{1}{3} \cdot \frac{3}{4}m \leq \frac{m}{4} \right) \text{ (By Inequality 10)} \\
&\Leftrightarrow \left( \frac{m}{4} \leq \frac{m}{4} \right)
\end{aligned}$$

and Lemma 9 is thereby proved. ◀

## 4 Summary and conclusions

Mixed-criticality scheduling is emerging as an increasingly important topic in the design, analysis, and implementation of safety-critical embedded systems. Most prior work on this topic has been restricted to uniprocessor scheduling; what little work has been done on multiprocessor scheduling has primarily focused upon recurrent (periodic and sporadic) workload models that are very different from the one we consider in this paper. We have adapted ideas from some such prior work, and have applied them to our problem of scheduling collections of independent jobs in order to minimize makespan. We have designed algorithms for both preemptive and non-preemptive scheduling of such workloads, but have not yet been able to classify the computational complexity of preemptive scheduling to minimize makespan – we leave this as an open problem.

We reiterate a point we had made earlier in this manuscript – although the particular problem we have presented here was obtained by applying a large number of simplifying assumptions to the actual application system under analysis, we hope that exposing this very interesting and important problem domain to the FST&TCS community will encourage members of this community to work upon more realistic, and more complex, variations.

---

### References

- 1 Sanjoy Baruah, Arvind Easwaran, and Zhishan Guo. MC-Fluid: simplified and optimally quantified. In *Real-Time Systems Symposium (RTSS), 2015 IEEE*, Dec 2015.
- 2 Sanjoy K. Baruah, Liliana Cucu-Grosjean, Robert I. Davis, and Claire Maiza. Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminar 15121). *Dagstuhl Reports*, 5(3):84–142, 2015. doi:10.4230/DagRep.5.3.84.
- 3 Alan Burns and Robert Davis. Mixed-criticality systems: A review. Available at <http://www-users.cs.york.ac.uk/~burns/review.pdf>, 2013.
- 4 Chandra Chekuri and Sanjeev Khanna. On multi-dimensional packing problems. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 185–194, January 1999.
- 5 Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, 2004. doi:10.1137/S0097539799356265.
- 6 Jr. E. Coffman and P. J. Denning. *Operating Systems Theory*. Prentice-Hall, Englewood Cliffs, NJ, 1973.
- 7 Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, and Lothar Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *International Conference on Embedded Software (EMSOFT)*, pages 17:1–17:15, Montreal, Oct 2013.
- 8 Lee Jaewoo, Kieu-My Phan, Xiaozhe Gu, Jiyeon Lee, A. Easwaran, Insik Shin, and Insup Lee. MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 41–52, Dec 2014.
- 9 Jing Li, David Ferry, Shaurya Ahuja, Kunal Agrawal, Christopher Gill, and Chenyang Lu. Mixed-criticality federated scheduling for parallel real-time tasks. In *Proceedings of the 22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2016.
- 10 R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.
- 11 Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the Real-Time Systems Symposium*, pages 239–243, Tucson, AZ, December 2007. IEEE Computer Society Press.
- 12 Tjark Vredeveld. Vector scheduling problems. In Min-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2014.



# Capacitated $k$ -Center Problem with Vertex Weights

Aounon Kumar

Indian Institute of Technology Delhi, India  
aounon.kumar.csy15@cse.iitd.ac.in

---

## Abstract

We study the capacitated  $k$ -center problem with vertex weights. It is a generalization of the well known  $k$ -center problem. In this variant each vertex has a weight and a capacity. The assignment cost of a vertex to a center is given by the product of the weight of the vertex and its distance to the center. The distances are assumed to form a metric. Each center can only serve as many vertices as its capacity. We show an  $n^{1-\epsilon}$ -approximation hardness for this problem, for any  $\epsilon > 0$ , where  $n$  is the number of vertices in the input. Both the capacitated and the weighted versions of the  $k$ -center problem individually can be approximated within a constant factor. Yet the common extension of both the generalizations cannot be approximated efficiently within a constant factor, unless  $P = NP$ . This problem, to the best of our knowledge, is the first facility location problem with metric distances known to have a super-constant inapproximability result. The hardness result easily generalizes to versions of the problem that consider the  $p$ -norm of the assignment costs (weighted distances) as the objective function. We give  $n^{1-1/p-\epsilon}$ -approximation hardness for this problem, for  $p > 1$ .

We complement the hardness result by showing a simple  $n$ -approximation algorithm for this problem. We also give a bi-criteria constant factor approximation algorithm, for the case of uniform capacities, which opens at most  $2k$  centers.

**1998 ACM Subject Classification** F.2 Analysis of Algorithms and Problem Complexity

**Keywords and phrases** approximation hardness,  $k$ -center, gadget reduction

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.8

## 1 Introduction

Resource location problems are a class of problems in which one is required to find a set of locations to open centers in order to serve clients (demands) placed in a metric space. The objective is to reduce the cost of opening the centers and/or the cost incurred to assign the clients to the centers. Various notions of distance/cost are used in different applications. The  $k$ -center problem is a very well known resource location problem in which a metric on  $n$  vertices is given. The objective is to open  $k$  centers and assign vertices (clients) to these centers such that the maximum distance between a vertex and its assigned center is minimized. This problem is NP-hard. It also has a  $(2 - \epsilon)$ -approximation hardness. [14] 2-approximation algorithms were given by Gonzalez [12] and Hochbaum and Shmoys [13].

Motivated by practical scenarios where each center has a limitation on the number of clients that it can serve, a generalization of this problem is the *capacitated*  $k$ -center problem. In this problem, each vertex has a capacity and a center opened at a vertex cannot serve more number of vertices than its capacity. Khuller and Sussmann [16] gave 5 and 6-approximation algorithms for uniform soft and hard capacities respectively. For non-uniform capacities, Cygan *et al.* [10] and An *et al.* [1] provide constant factor approximation algorithms using LP rounding.



© Aounon Kumar;

licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 8; pp. 8:1–8:14



Leibniz International Proceedings in Informatics

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

Another generalization of the  $k$ -center problem is one where vertices have *weights*. The assignment cost of a vertex to a center is given by the product of the weight of the vertex and its distance (*weighted distance*) to the center. This variant is motivated from scenarios where the clients are not treated equally. Some clients are more important than others and need to be kept closer to an open center. Weights can also be used to model the likelihood of clients demanding services. Wang and Cheng [20] provide a 2-approximation for the  $k$ -center problem with vertex weights. This is best possible as the  $k$ -center problem has  $(2 - \epsilon)$ -approximation hardness.

A common extension of the above two generalizations is the *capacitated  $k$ -center with vertex weights*. In this variant each vertex has a capacity and a weight. Each center can serve no more vertices than its capacity. The assignment cost of a vertex to a center is given by its weighted distance to the center. In this paper we study the approximability of this problem. We show an  $n^{1-\epsilon}$ -approximation hardness and provide an  $n$ -approximation algorithm. The hardness result easily generalizes to variants of the problem that consider the  $p$ -norm of the assignment costs (weighted distances) as the objective function. We give  $n^{1-\frac{1}{p}-\epsilon}$ -approximation hardness for the general  $p$ -norm, for  $p > 1$ . This immediately shows that for  $p > 1$ , the problem is hard to approximate within a constant factor. Although this generalization does not immediately provide an inapproximability result for the 1-norm which is the corresponding variant of the  $k$ -median problem, it provides insights into the capacitated (unweighted) version of the problem. The capacitated  $k$ -median problem is interesting as not much is known about its approximability. Constant factor approximation algorithms by either violating the capacity constraints or the cardinality constraints up to a constant factor are studied in [9], [4], [18], [5].

A vast body of work is available on various facility location problems. A variety of techniques like local search [17], [2], [6], LP rounding [7] and primal-dual method [15] have been studied. The capacitated facility location problem is well studied in [17], [19], [8], [3] and constant factor approximations are known.

### Our results and techniques

The main result of this paper is the approximation hardness of the capacitated  $k$ -center problem with vertex weights. We show that this problem cannot be efficiently approximated within a factor of  $n^{1-\epsilon}$ , unless  $P = NP$ , for any  $\epsilon > 0$ , where  $n$  is the number of vertices in the input. We give a reduction from the EXACT COVER BY 3-SETS, which is an NP-complete problem. It requires one to find a set cover from a family of sets, where each set has exactly three elements, such that each element of the universe is in exactly one of the sets in the set cover. This set cover variant was used by Cygan *et al.* in [10] to show a  $(3 - \epsilon)$ -approximation hardness for the capacitated  $k$ -center problem. The set gadget used in the reduction in [10] is designed for the unweighted case and does not generalize for the weighted case. In this paper, we introduce a novel set gadget that allows to create a polynomial factor gap between the solution cost of the *yes* and the *no* instances. It achieves this by allowing the vertices in a set gadget to be assigned to centers inside the gadget with *small* costs and making assignments to centers outside the gadget incur a *large* cost. Similarly, the vertices in an element gadget can only be assigned to centers in set gadgets corresponding to sets that it belongs to, with a small cost. Our reduction generates instances where the capacities are uniform and constant, showing that even this special case is hard to approximate within a constant factor.

An immediate consequence of the hardness result is an  $n^{1-\frac{1}{p}-\epsilon}$ -approximation hardness for the case where the objective function is a general  $p$ -norm of the assignment costs, for  $p > 1$ . The  $k$ -center problem is a special case where the objective function is the  $\infty$ -norm of

the assignment costs. This shows that interesting variants of the problem which consider a  $p$ -norm for  $p > 1$  are hard to approximate within a constant factor.

We complement the hardness result by showing a simple  $n$ -approximation algorithm. For this algorithm, we use the standard *thresholding* technique modified to handle weights. We create threshold graphs corresponding to each distinct weight in decreasing order and open as many centers, in decreasing order of capacities, in each connected component as required to cover all the vertices in it.

Next, we relax the cardinality constraint on the set of centers. We consider the variant with uniform capacities where we show that if we are allowed to open twice the number of centers then we can output a solution with cost within a constant factor of the optimum cost. This simply modifies the 2-approximation by Wang and Cheng [20] by opening as many capacitated centers required in place of each uncapacitated center to serve all the vertices assigned to it.

## 2 Problem statement

The input for the capacitated  $k$ -center problem with vertex weights (CkCW) is a set of vertices  $V$ , a metric distance  $d : V \times V \rightarrow \mathbb{R}_{\geq 0}$  on  $V$ , an integer  $k$ , a capacity function  $L : V \rightarrow \mathbb{Z}_{\geq 0}$  and a weight function  $W : V \rightarrow \mathbb{R}_{\geq 0}$ . The output is a set  $S \subseteq V$  of  $k$  vertices called *centers* and an *assignment map*  $h : V \rightarrow S$  such that  $|\{j \in V \mid h(j) = i\}| \leq L(i), \forall i \in S$ . The *assignment cost* of a vertex  $j \in V$  to a center  $i \in S$  is given by  $W(j)d(i, j)$ . The goal is to minimize the maximum assignment cost of a vertex to its assigned center. Formally, the cost of the solution is given by  $\max_{j \in V} W(j)d(h(j), j)$ . Let  $|V| = n$ .

The metric distance  $d$  satisfies the following properties for  $i, j, u \in V$ :

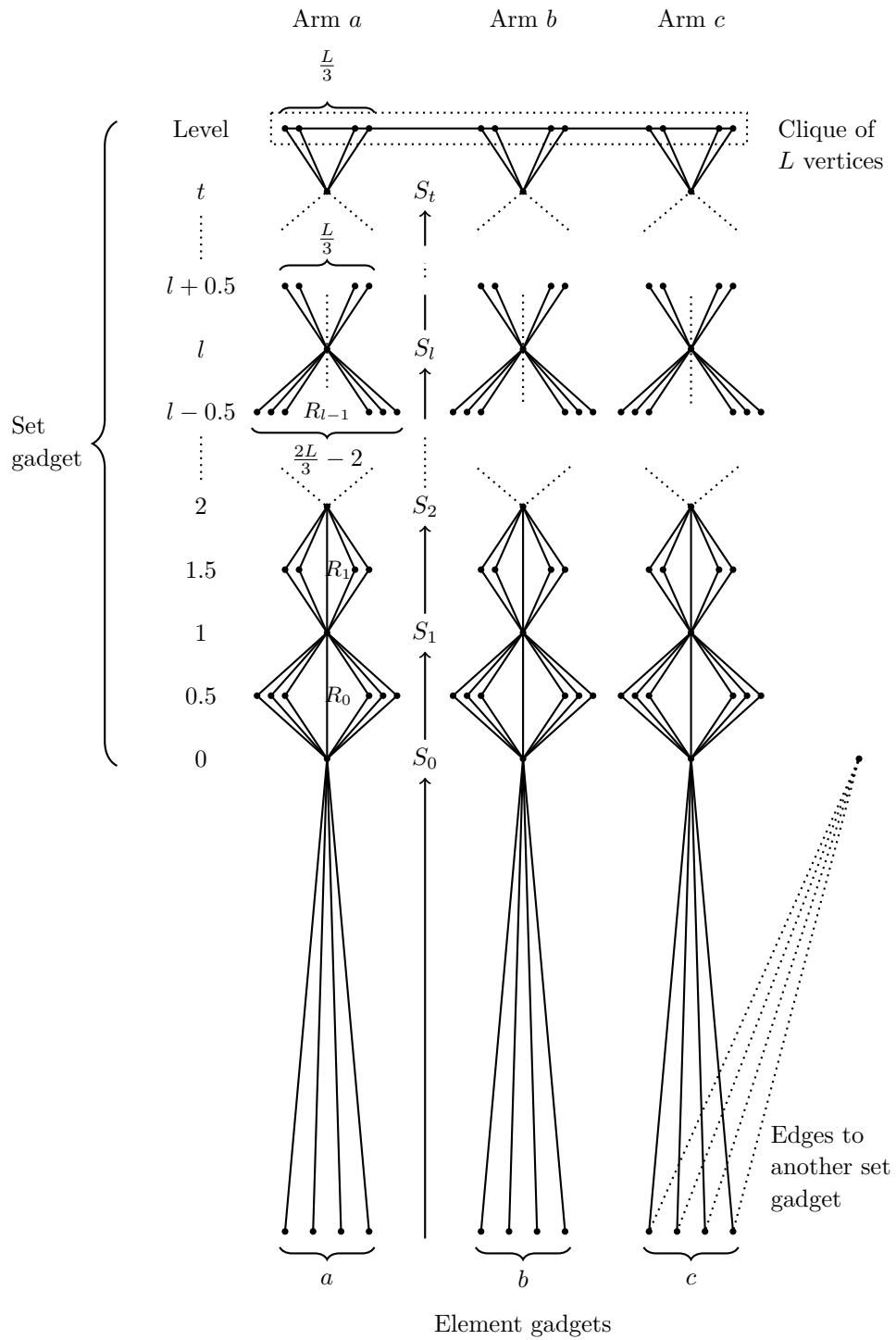
1.  $d(i, j) \geq 0$
2.  $d(j, j) = 0$
3.  $d(i, j) = d(j, i)$
4.  $d(i, j) \leq d(i, u) + d(u, j)$

## 3 Hardness of approximation

In this section we show that the above problem cannot be approximated within a constant factor. We give a reduction from the EXACT COVER BY 3-SETS (EC3S), which is an NP-complete problem. This problem is used in [10] to show a  $(3 - \epsilon)$ -approximation hardness for the  $k$ -center problem (unweighted) with non-uniform capacities. The input of the problem is a set system  $(\mathcal{F}, \mathcal{U})$ , where each set in  $\mathcal{F}$  has exactly 3 elements. The goal is to decide whether there exists a subset  $\mathcal{F}' \subseteq \mathcal{F}$ , such that each element of  $\mathcal{U}$  belongs to exactly one set in  $\mathcal{F}'$ . For such a set cover to exist,  $|\mathcal{U}|$  must be a multiple of three.

An instance of the EC3S problem can be viewed as a bipartite graph  $(\mathcal{F} \cup \mathcal{U}, \mathcal{E})$  where the edge set  $\mathcal{E}$  encodes the membership of the elements of  $\mathcal{U}$  in the elements of  $\mathcal{F}$ . In our reduction, we encode this bipartite graph into an instance  $\mathcal{I}$  of CkCW, with each vertex having a uniform capacity of  $L$ . We replace each vertex of  $\mathcal{F}$  with the corresponding *set gadget* and that of  $\mathcal{U}$  with the corresponding *element gadget*.

Figure 1 illustrates these *gadgets*. The *set gadget* consists of three *long* arms, one for each of the three elements in the set, joined together with a clique at the top. Each arm is divided into *integral* levels  $0, 1, \dots, t$  and *fractional* levels  $0.5, 1.5, \dots, t + 0.5$ , where  $t$  is an odd integer which we will fix later in this construction. An integral level  $l$  consists of a vertex of weight  $W_l$ . A fractional level  $l + 0.5$  contains  $\frac{L}{3}$  vertices of weight  $W_l$  if  $l$  is odd and  $\frac{2L}{3} - 2$



■ **Figure 1** Gadgets for reduction.



vertices of weight  $W_l$  if  $l$  is even. The two vertices in levels  $l$  and  $l + 1$  are connected to each other and to the vertices in level  $l + 0.5$  by edges of length  $R_l$ . The  $\frac{L}{3}$  vertices in level  $t + 0.5$  (the highest level) from each of the arms are all connected to each other by edges of length  $R_t$ , forming a clique of size  $L$ . The *element gadget* is a collection of  $\frac{L}{3}$  vertices of unit weight connected to the level 0 vertex of the corresponding arm of the set gadget of each of the sets that it belongs to. The length of each of these connecting edges is  $S_0$ .

Let  $S_l$  denote the shortest distance of a level  $l$  vertex from the vertices in the element gadget connected to the corresponding arm (refer to Figure 1). Then we have the following relation:

$$S_l = R_{l-1} + S_{l-1} \quad (1)$$

We would like to set the parameters of the construction in such a way that any solution with cost  $< w^2$  must assign the vertices in a set gadget to centers in the same gadget. It must also assign the vertices of an element gadget to centers in set gadgets corresponding to the sets it belongs to. So, we want the following relations to hold:

$$W_0 = w \quad (2)$$

$$W_l R_l = w \quad (3)$$

$$W_l S_l = w^2 \quad (4)$$

where  $w$  is some parameter. From equations 1, 3 and 4 we get:

$$\begin{aligned} S_l &= S_{l-1} \left( \frac{1}{w} + 1 \right) \\ &= S_0 \left( \frac{1}{w} + 1 \right)^l = w \left( \frac{1}{w} + 1 \right)^l \end{aligned} \quad (\text{from equations 2 and 4})$$

We fix  $t$  such that:

$$\begin{aligned} S_t &= w \left( \frac{1}{w} + 1 \right)^t \geq w^2 \\ t &\geq \frac{\log(w)}{\log\left(1 + \frac{1}{w}\right)} \leq 2w \log w \end{aligned}$$

We set  $t$  to be an odd integer just greater than  $2w \log w$  and  $k$  to be  $3 \left( \frac{t+1}{2} \right) |\mathcal{F}| + \frac{|\mathcal{U}|}{3}$ . The distance metric  $d$  is given by the shortest distance metric.

► **Lemma 1.** *If there exists a solution to the EC3S instance, then there exists a solution to the instance  $\mathcal{I}$  with cost  $w$ .*

**Proof.** Let  $\mathcal{F}' \subseteq \mathcal{F}$  be the solution of the EC3S instance. Note that,  $|\mathcal{F}'| = \frac{|\mathcal{U}|}{3}$ . For a sets  $A \in \mathcal{F}'$  place a center on each of the three vertices at even levels  $0, 2, \dots, t-1$  in the corresponding set gadget and one center on a vertex at level  $t + 0.5$ . Assign the vertices of the element gadget corresponding to the elements in  $A$  and the vertices in levels  $0, 0.5$  and  $1$  to the centers at level  $0$ . For a level  $l \in \{2, 4, \dots, t-1\}$ , assign all the vertices at levels  $l-0.5, l, l+0.5$  and  $l+1$  to the centers at level  $l$ . Assign all the vertices in the clique at level  $t+0.5$  to the center opened at this level. For all sets not in  $\mathcal{F}'$ , place centers similarly at odd levels  $1, 3, \dots, t$ . For a level  $l \in \{1, 3, \dots, t\}$ , assign all the vertices at levels  $l-1, l-0.5, l$  and  $l+0.5$  to the centers at level  $l$ . This is an assignment with cost  $w$ . The total number of centers opened is  $\sum_{A \in \mathcal{F}'} \left( 3 \left( \frac{t+1}{2} \right) + 1 \right) + \sum_{A \notin \mathcal{F}'} 3 \left( \frac{t+1}{2} \right) = 3 \left( \frac{t+1}{2} \right) |\mathcal{F}| + \frac{|\mathcal{U}|}{3} = k$ . ◀

Now consider a solution  $\mathcal{S}$  to the instance  $\mathcal{I}$  with maximum assignment cost  $< w^2$ . Note that each center must serve  $L$  vertices as  $|V| = 3 \left(\frac{t+1}{2}\right) |\mathcal{F}|L + |\mathcal{U}|\frac{L}{3} = kL$ .

► **Lemma 2.**  $\mathcal{S}$  does not have a center in any of the element gadgets.

**Proof.** Consider the vertex set  $g_a$  of the element gadget for an element  $a \in \mathcal{U}$ . From the construction of the gadget we can say that any  $j \notin g_a$  and  $i \in g_a$ ,  $W(j)d(i, j) \geq w^2$ . Therefore, only the vertices in  $g_a$  can be assigned to a center in  $g_a$ . But,  $|g_a| = \frac{L}{3} < L$ . ◀

► **Lemma 3.** Each set gadget in  $\mathcal{S}$  has at least  $3 \left(\frac{t+1}{2}\right)$  and at most  $3 \left(\frac{t+1}{2}\right) + 1$  open centers in it.

**Proof.** Consider the vertex set  $g_A$  of the set gadget for a set  $A \in \mathcal{F}$ . For any  $j \in g_A$  and  $i \notin g_A$ ,  $W(j)d(i, j) \geq w^2$ . Thus, all the vertices in  $g_A$  must be assigned to centers in  $g_A$ . Therefore, the number of centers in  $g_A \geq \lceil |g_A|/L \rceil = 3 \left(\frac{t+1}{2}\right)$ .

Assume, for contradiction, that the number of centers in  $g_A > 3 \left(\frac{t+1}{2}\right) + 1$ . Let  $a, b$  and  $c$  be the elements of set  $A$  and let  $g_a, g_b$  and  $g_c$  be the vertex sets of their respective gadgets. For any  $j \notin g_A \cup g_a \cup g_b \cup g_c$  and  $i \in g_A$ ,  $W(j)d(i, j) \geq w^2$ . Thus the number of vertices that the centers in  $g_A$  can serve  $\leq |g_A| + |g_a| + |g_b| + |g_c| = (3 \left(\frac{t+1}{2}\right) + 1)L$ . Therefore, at least one of the centers in  $g_A$  must be serving less than  $L$  vertices. ◀

► **Lemma 4.** In  $\mathcal{S}$ , gadgets corresponding to any two sets in  $\mathcal{F}$  sharing a common element cannot have  $3 \left(\frac{t+1}{2}\right) + 1$  open centers in each one of them.

**Proof.** Assume, for contradiction, that there exist two sets  $A, B \in \mathcal{F}$  having at least one element in common such that the vertex sets  $g_A$  and  $g_B$  of the corresponding gadgets each have  $3 \left(\frac{t+1}{2}\right) + 1$  centers. As shown in the proof of Lemma 3, the vertices that can be assigned to a center in the gadget of a set  $C = \{d, e, f\}$  are only those in  $g_C \cup g_d \cup g_e \cup g_f$ . Thus, the number of vertices that can be assigned to centers in  $g_A$  and  $g_B \leq 2 \times (3 \left(\frac{t+1}{2}\right) + 1)L - \frac{L}{3}$  (since at least one element is common in  $A$  and  $B$ ). Therefore, at least one of the centers in  $g_A$  or  $g_B$  must be serving less than  $L$  vertices. ◀

► **Lemma 5.** If there exists a weighted  $k$ -center solution with cost  $R < w^2$ , then there exists a solution to the EC3S instance.

**Proof.** From Lemmas 2, 3 and 4, there are  $\frac{|\mathcal{U}|}{3}$  set gadgets each of which have  $3 \left(\frac{t+1}{2}\right) + 1$  centers and the corresponding sets are all disjoint. These  $\frac{|\mathcal{U}|}{3}$  sets form the solution set  $\mathcal{F}'$ . ◀

► **Theorem 6.** The weighted  $k$ -center solution cannot be approximated within a factor of  $n^{1-\epsilon}$  for any  $\epsilon > 0$ , unless  $P = NP$ .

**Proof.** From Lemmas 1 and 5, an  $\alpha$ -approximation is not possible for  $\alpha < w$ , unless  $P = NP$ .

Now, we show a lower bound on  $w$  in terms of  $n$ . In the construction, the number of vertices is given by:

$$\begin{aligned} n = kL &= \left(3 \left(\frac{t+1}{2}\right) |\mathcal{F}| + \frac{|\mathcal{U}|}{3}\right) L \\ &\leq \text{constant} \times w \log w |\mathcal{F}| && (|\mathcal{F}| \geq \frac{|\mathcal{U}|}{3}, t \sim 2w \log w \text{ and } L \text{ is constant}) \\ &\leq \text{constant} \times w^{1+\frac{1}{q}} \log w && (\text{setting } w = |\mathcal{F}|^q, q > 0) \\ &\leq \text{constant} \times w^{1+\frac{2}{q}} \\ w &\geq \text{constant} \times n^{\frac{1}{1+\frac{2}{q}}} > n^{1-\epsilon} && (\text{for sufficiently large } n \text{ and } q > \frac{2}{\epsilon}) \end{aligned}$$

◀

► **Remark.** The capacity  $L$  of each vertex does not depend on the input of the reduction. Thus,  $L$  can be fixed to be a sufficiently large constant. In Appendix A, we show that the known hardness results of  $(3 - \epsilon)$  for the  $\{0, L\}$  capacitated version [10] and  $(2 - \epsilon)$  for the uniform  $L$  capacitated version (which follows from the  $(2 - \epsilon)$ -approximation hardness of the uncapacitated problem [14]) of  $k$ -center problem hold even when  $L$  is a constant. Note, that for  $L = 1$  the problem can be solved trivially.

### Generalizing to other cost functions

In the  $k$ -center problem the goal is to minimize the maximum assignment cost, that is, to minimize the infinity norm of the assignment costs. Now we generalize the hardness result for any  $p$ -norm as the objective function. The objective function is given by:

$$\left( \sum_{j \in V} (W(j)d(h(j), j))^p \right)^{\frac{1}{p}}.$$

Consider the instance  $\mathcal{I}$  generated by the reduction. If there exists a solution to the EC3S instance, there exists a solution to the instance  $\mathcal{I}$  with cost at most  $n^{\frac{1}{p}}w$  and if there is no solution to the EC3S instance then any solution to  $\mathcal{I}$  must have a cost at least  $w^2$ . Thus an approximation factor of  $w/n^{\frac{1}{p}}$  or,  $n^{1-\frac{1}{p}-\epsilon}$  cannot be achieved, unless  $P = NP$ . This gives a super-constant inapproximability result for,  $p > 1$ .

## 4 $n$ -approximation algorithm

In this section, we present a simple  $n$ -approximation algorithm for the capacitated  $k$ -center problem with vertex weights. It guesses through all possible values  $R$  of the optimal solution cost in increasing order. The number of possible values can be at most  $|V|^2$  as each value must be equal to  $W(j)d(i, j)$  for some  $i, j \in V$ . For each  $R$ , consider the distinct values of the weights  $w_1 > w_2 > \dots > w_m$  in decreasing order, where  $m$  is the number of distinct weights. For each distinct weight  $w_i$ , it creates the undirected graph  $G_{r_i} = (V, E_{r_i})$  where  $V$  is the input set of vertices and  $E_{r_i} = \{(i, j) \mid d(i, j) \leq r_i = R/w_i\}$ . Note that if  $R$  is the optimal solution cost then the optimal solution cannot assign a vertex  $j$  to a center  $i$  such that  $d(i, j) > R/W(j)$ . Let  $\Gamma_i$  be the set of connected components of  $G_{r_i}$  which have at least one vertex of weight at least  $w_i$ . For a component  $\gamma \in \Gamma_i$ , let  $\mathcal{H}_i^\gamma = \{v \in \gamma \mid W(v) \geq w_i\}$  be the set of *heavy* vertices and  $\mathcal{P}_\gamma$  be the set of open centers in  $\gamma$ .  $\mathcal{P}_\gamma$  for  $\gamma \in \Gamma_i$ , initially consists of the centers opened at vertices in  $\gamma$  up till iteration  $i - 1$ . We say a center in  $\mathcal{P}_\gamma$  is *unsaturated* if the number of vertices assigned to it is less than its capacity. The algorithm, in iteration  $i$ , assigns vertices from  $\mathcal{H}_i^\gamma$  for each component  $\gamma$  to unsaturated centers in  $\mathcal{P}_\gamma$  till their capacities are exhausted and then adds new centers to serve all the remaining vertices in  $\mathcal{H}_i^\gamma$ . After  $m$  iterations, if the number of open centers is at most  $k$  it returns the set as the solution. Algorithm 1 illustrates this procedure.

► **Lemma 7.** *The assignment cost of each vertex is at most  $nR$ .*

**Proof.** Note that in each iteration  $i$ , the algorithm assigns all the vertices of weight  $w_i$  to some center. Also, each vertex in a component is assigned to some center in the same component. Thus the assignment cost is at most  $w_i n r_i = w_i n R / w_i = nR$ . ◀

Consider an optimal solution  $\mathcal{S}^*$ . Let  $R^*$  be the optimal solution cost. For a center  $u$  in the optimal solution, let  $\sigma(u)$  be the number of vertices assigned to it. Now consider the iteration of Algorithm 1 when  $R = R^*$ .

**Algorithm 1**  $n$ -approximation algorithm

---

```

for each guess  $R$  of the optimal solution cost in increasing order do
  Order the weights  $w_1 > w_2 > \dots > w_m$ 
  for each  $w_i, i \in \{1, 2, \dots, m\}$  do
    Construct  $G_{r_i}$ 
    Construct the set  $\Gamma_i$  for  $G_{r_i}$ 
    for each component  $\gamma \in \Gamma_i$  do
      Construct  $\mathcal{P}_\gamma$ 
      while  $\exists$  unassigned vertex  $v \in \mathcal{H}_i^\gamma$  do
        if  $\exists$  unsaturated center  $u \in \mathcal{P}_\gamma$  then
          assign  $v$  to  $u$ 
        else
           $\mathcal{P}_\gamma \rightarrow \mathcal{P}_\gamma \cup \{u\}$ , where  $u \in \gamma \setminus \mathcal{P}_\gamma$ , such that  $L(u) = \max\{L(v) \mid v \in \gamma \setminus \mathcal{P}_\gamma\}$ 
        end if
      end while
    end for
  end for
  if  $|\bigcup_{\gamma \in \Gamma_m} \mathcal{P}_\gamma| \leq k$  then
    return set of open centers and vertex assignment map
  end if
end for

```

---

► **Lemma 8.** *For a component  $\gamma \in \Gamma_i$  of any  $G_{r_i}$ , there exists a set of centers  $\chi_\gamma$  opened by  $\mathcal{S}^*$  in component  $\gamma$  with the following properties:*

1.  $|\chi_\gamma| = |\mathcal{P}_\gamma| = \kappa_\gamma$
2. *Order the elements  $u_i$  of  $\chi_\gamma$  in decreasing order of the value of  $\sigma(u_i)$  and the elements  $p_i$  of  $\mathcal{P}_\gamma$  in decreasing order of their capacities  $L(p_i)$ . For  $i \in \{1, 2, \dots, \kappa_\gamma\}$ ,  $\sigma(u_i) \leq L(p_i)$ .*

**Proof.** We prove this by induction on  $i$ . The lemma holds for  $\Gamma_1$  since the algorithm opens centers in decreasing order of capacities in each of the components. Assume it holds for  $\Gamma_i$  for some  $i$ . Note that, from the construction of a component in  $\Gamma_i$ , we can say that each component in  $\Gamma_i$  is disjoint from other components in  $\Gamma_i$  and is a subset of some component in  $\Gamma_{i+1}$ . Now consider a component  $\gamma \in \Gamma_{i+1}$ . Let  $\gamma_1, \gamma_2, \dots, \gamma_z$  be the components in  $\Gamma_i$  which are subsets of  $\gamma$ . As long as there is an unsaturated center from iteration  $i$ , the algorithm assigns vertices to that center. If all the vertices in  $\gamma$  are assigned to some center from iteration  $i$ , then the lemma holds for  $\Gamma_{i+1}$ . The corresponding  $\mathcal{P}_\gamma$  and  $\chi_\gamma$  would be  $\mathcal{P}_{\gamma_1} \cup \mathcal{P}_{\gamma_2} \cup \dots \cup \mathcal{P}_{\gamma_z}$  and  $\chi_{\gamma_1} \cup \chi_{\gamma_2} \cup \dots \cup \chi_{\gamma_z}$  respectively.

Now consider the case when all the centers from iteration  $i$  are saturated.  $\mathcal{P}_\gamma = \mathcal{P}_{\gamma_1} \cup \mathcal{P}_{\gamma_2} \cup \dots \cup \mathcal{P}_{\gamma_z}$  and  $\chi_\gamma = \chi_{\gamma_1} \cup \chi_{\gamma_2} \cup \dots \cup \chi_{\gamma_z}$  satisfy the conditions of the lemma. Arrange all the vertices in  $\gamma$  in decreasing order of capacities. Let  $q$  be the smallest index in this ordering such that the algorithm has not opened a center at the  $q^{\text{th}}$  vertex. Replace the first  $q - 1$  centers in  $\chi_\gamma$  with the highest  $q - 1$  centers opened in  $\gamma$  by  $\mathcal{S}^*$ , according to the number of vertices served. The new  $\chi_\gamma$  and  $\mathcal{P}_\gamma$  also satisfy both the conditions of the lemma. Now, if there are unassigned vertices even after all centers in  $\mathcal{P}_\gamma$  are saturated, the algorithm opens center at the vertex at index  $q$  and adds it to  $\mathcal{P}_\gamma$ . The optimum solution must also have an open center  $u \notin \chi_\gamma$  as the centers in  $\chi_\gamma$  do not serve all the vertices in  $\gamma$ .  $\sigma(u)$  can be at most the number of vertices served by the  $q^{\text{th}}$  maximum center in the optimum solution which is at most the capacity of the newly opened center. We compute  $q$

again and replace the  $q - 1$  centers in  $\chi_\gamma$  as previously. This shows that both the conditions of the lemma hold when each new center is added by the algorithm. Hence, the lemma holds for  $\Gamma_{i+1}$ .  $\blacktriangleleft$

► **Theorem 9.** *Algorithm 1 is an  $n$ -approximation algorithm for the capacitated  $k$ -center problem with vertex weights.*

**Proof.** When  $R = R^*$ , consider  $\mathcal{P}_\gamma$  for  $\gamma \in \Gamma_m$  after the algorithm has iterated through all the distinct weights. At this point, each vertex is assigned to some open center. From Lemma 8, there exists a set of centers  $\chi_\gamma$  opened by  $\mathcal{S}^*$  in component  $\gamma$  such that  $|\chi_\gamma| = |\mathcal{P}_\gamma| = \kappa_\gamma$ . Since all the components are disjoint, the number of centers opened by the algorithm is  $\sum_{\gamma \in \Gamma_m} |\mathcal{P}_\gamma| = \sum_{\gamma \in \Gamma_m} |\chi_\gamma| \leq k$ . Also, from Lemma 7, each assignment cost is at most  $nR^*$ .  $\blacktriangleleft$

## 5 Relaxing the number of centers

In this section we present a greedy  $(2, 2)$ -approximation algorithm<sup>1</sup> for the uniform soft capacitated  $k$ -center problem with vertex weights. In the soft capacitated version, the solution is allowed to have multiple centers at a vertex. All vertices have equal capacities of  $L$ . The algorithm uses the greedy clustering technique used by Wang and Cheng in [20] to produce a solution for the uncapacitated version of the problem. It then replaces the open uncapacitated centers with the required number of capacitated ones.

For an input instance  $\mathcal{I}$  and a solution cost  $R$ , we can construct a digraph  $G_R = (V, E_R)$ , where  $V$  is the set of vertices in  $\mathcal{I}$  and  $E_R = \{(j, i) \mid W(j)d(i, j) \leq R\}$  is the set of edges that a solution with cost  $R$  can potentially use to assign vertices to centers. Thus, a directed edge  $(j, i) \in E_R$  if  $j$  can be assigned to  $i$  within cost  $R$ . It is easy to verify that there exists a solution to  $\mathcal{I}$  with cost  $R$  if and only if there exists a set  $S \subseteq V, |S| = k$  and an assignment map  $h : V \rightarrow S$  assigning vertices to centers respecting the capacity constraint and using only the edges in  $E_R$ , that is,  $h(j) = i \implies (j, i) \in E_R$ .

Given an instance  $\mathcal{I}$  of the problem, the algorithm goes through all possible values  $R$  (which can be at most  $|V|^2$ ) of the optimal solution cost, in increasing order. It constructs the graph  $G_R$  and for each vertex  $v \in V$ , computes its neighbourhood  $N(v)$  as:

$$N(v) = \{v\} \cup \{u \mid (u, v) \in E_R\} \cup \{u \mid \exists x \in V, (v, x), (u, x) \in E_R\}$$

It then select a set of vertices  $S$  greedily according to weight and clusters  $(C_v)$  the vertices in the neighbourhood of each vertex  $v \in S$ . It opens sufficient number centers at the vertices in  $S$  (with multiple centers at a vertex if required) such that all the vertices can be assigned to some center with cost at most  $2R$ , respecting the capacity constraint. Algorithm 2 formally defines this greedy procedure.

► **Lemma 10.** *For any vertex in a cluster  $C_v, \forall v \in S$ , its cost of assignment to an open center at  $v$  is at most  $2R$ . Formally,*

$$d(v, j)W(j) \leq 2R, \forall j \in C_v.$$

**Proof.** The lemma holds trivially for  $v$ . All other vertices  $j \in C_v$  are of the following two types:

<sup>1</sup> An  $(\alpha, \beta)$ -approximation algorithm outputs a solution with cost at most  $\alpha R$  by opening at most  $\beta k$  centers

**Algorithm 2** Greedy algorithm

---

```

for each guess  $R$  of the solution cost in increasing order do
  Construct  $G_R$ .
  for each  $v \in V$  do
    Construct  $N(v)$ 
  end for
   $\mathcal{X} \leftarrow V$ 
   $S \leftarrow \phi$ 
  while  $\mathcal{X}$  is not empty do
    select  $v \in \mathcal{X}$  such that  $W(v) = \max\{W(v) \mid v \in \mathcal{X}\}$ 
     $S \leftarrow S \cup \{v\}$ 
    Assign  $\mathcal{X} \cap N(v)$  to cluster  $C_v$ 
    Open  $\lceil |C_v|/L \rceil$  centers at  $v$ 
     $\mathcal{X} \leftarrow \mathcal{X} \setminus N(v)$ 
  end while
  if number of open centers  $\leq 2k$  then
    return set of open centers
  end if
end for

```

---

**Type 1:**  $(j, v) \in E_R$ . In this case, by definition of  $E_R$  we have:

$$W(j)d(v, j) \leq R \leq 2R.$$

**Type 2:**  $\exists x \in V, (v, x), (j, x) \in E_R$ . Algorithm 2 in its while loop selects the maximum weight vertex  $v$  from the set  $\mathcal{X}$  in a given iteration. Since,  $C_v \subseteq \mathcal{X}$ , therefore,  $W(j) \leq W(v)$ .

$$\begin{aligned}
W(j)d(v, j) &\leq W(j)(d(v, x) + d(x, j)) && \text{(using triangle inequality)} \\
&\leq W(j) \left( \frac{R}{W(v)} + d(x, j) \right) && ((v, x) \in E_R) \\
&\leq W(j) \left( \frac{R}{W(j)} + d(x, j) \right) && (W(j) \leq W(v)) \\
&\leq R + W(j)d(x, j) \leq 2R && ((j, x) \in E_R)
\end{aligned}$$

◀

Let  $R^*$  be the optimal solution cost. Now consider the iteration of Algorithm 2 when  $R = R^*$ .

► **Lemma 11.** *Algorithm 2 opens at most  $2k$  centers and every vertex in cluster  $C_v$  can be assigned to some open center at  $v$ .*

**Proof.** Algorithm 2 opens  $\lceil |C_v|/L \rceil$  centers at vertex  $v$  in cluster  $C_v$ , which is sufficient to serve all vertices in  $C_v$ . Also, no two vertices in  $S$  can be served by the same center in the optimal solution, otherwise one of them must be in the neighbourhood of the other. Thus,  $k \geq |S|$ . The total number of centers  $k'$  opened by Algorithm 2 follows,

$$k' = \sum_{v \in S} \lceil |C_v|/L \rceil = |S| + \sum_{v \in S} \lfloor |C_v|/L \rfloor \leq |S| + \lfloor |V|/L \rfloor \leq 2k \quad (\text{all clusters are disjoint})$$

◀

► **Theorem 12.** *Algorithm 2 is a  $(2, 2)$ -approximation algorithm for the uniform soft capacitated  $k$ -center problem with vertex weights.*

**Proof.** Follows from Lemmas 10 and 11. ◀

► **Remark.** Algorithm 2 can be modified to a  $(4, 2)$ -approximation for the uniform hard capacitated  $k$ -center problem with vertex weights. In the hard capacitated version, multiple centers are not allowed to be opened at the same location. So, instead of opening all the centers in a cluster at one vertex we open one center at each of the top  $\lceil |C_v|/L \rceil$  vertices in  $C_v$  in decreasing order of weight. The cost of assigning a vertex with a lower weight to a center with higher weight is at most  $4R$ .

## 6 Conclusion and open problems

In this paper we make progress towards showing approximation hardness for capacitated facility location problems with vertex weights. To the best of our knowledge, this is the first facility location problem known to be hard to approximate within a constant factor. This provides insight into other variants, for many of which not much is known about their approximabilities. It would be interesting to extend our result for the  $k$ -median problem.

Other directions for future work would be to reduce the gap between the lower bound of  $n^{1-\epsilon}$  and the upper bound of  $n$  presented in this paper and to design algorithms that achieve a constant factor on the solution cost by relaxing the cardinality or capacity constraints up to a constant smaller than 2.

**Acknowledgements.** We thank Naveen Garg and Amit Kumar for many helpful discussions. We also thank anonymous reviewers for their valuable comments.

---

### References

- 1 Hyung-Chan An, Aditya Bhaskara, Chandra Chekuri, Shalmoli Gupta, Vivek Madan, and Ola Svensson. Centrality of trees for capacitated  $k$ -center. In *Integer Programming and Combinatorial Optimization*, pages 52–63. Springer, 2014.
- 2 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristic for  $k$ -median and facility location problems. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC'01, pages 21–29, New York, NY, USA, 2001. ACM. doi:10.1145/380752.380755.
- 3 Manisha Bansal, Naveen Garg, and Neelima Gupta. A 5-approximation for capacitated facility location. In *Proceedings of the 20th Annual European Conference on Algorithms*, ESA'12, pages 133–144, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-33090-2\_13.
- 4 Jarosław Byrka, Krzysztof Fleszar, Bartosz Rybicki, and Joachim Spoerhase. Bi-factor approximation algorithms for hard capacitated  $k$ -median problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'15, pages 722–736, Philadelphia, PA, USA, 2015. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=2722129.2722178>.
- 5 Jarosław Byrka, Bartosz Rybicki, and Sumedha Uniyal. An approximation algorithm for uniform capacitated  $k$ -median problem with  $(1 + \epsilon)$  capacity violation. In *Proceedings of the 18th International Conference on Integer Programming and Combinatorial Optimization (IPCO'16)*, volume 9682 of *LNCS*, pages 262–274, New York, NY, USA, 2016. Springer-Verlag New York, Inc. doi:10.1007/978-3-319-33461-5\_22.

- 6 Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and  $k$ -median problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science, FOCS'99*, pages 378–, Washington, DC, USA, 1999. IEEE Computer Society. URL: <http://dl.acm.org/citation.cfm?id=795665.796483>.
- 7 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the  $k$ -median problem (extended abstract). In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing, STOC'99*, pages 1–10, New York, NY, USA, 1999. ACM. doi:10.1145/301250.301257.
- 8 A. Fabián Chudak and P. David Williamson. Improved approximation algorithms for capacitated facility location problems. *Mathematical Programming*, 102(2):207–222, 2005. doi:10.1007/s10107-004-0524-9.
- 9 Julia Chuzhoy and Yuval Rabani. Approximating  $k$ -median with non-uniform capacities. In *In SODA'05*, pages 952–958, 2005.
- 10 Marek Cygan, MohammadTaghi Hajiaghayi, and Samir Khuller. Lp rounding for  $k$ -centers with non-uniform hard capacities. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 273–282. IEEE, 2012.
- 11 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
- 12 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.
- 13 Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the  $k$ -center problem. *Mathematics of operations research*, 10(2):180–184, 1985.
- 14 Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.
- 15 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and  $k$ -median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, March 2001. doi:10.1145/375827.375845.
- 16 Samir Khuller and Yoram J. Sussmann. The capacitated  $k$ -center problem. In *In Proceedings of the 4th Annual European Symposium on Algorithms, Lecture Notes in Computer Science 1136*, pages 152–166. Springer, 1996.
- 17 Madhukar R. Korupolu, C.Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of Algorithms*, 37(1):146–188, 2000. doi:10.1006/jagm.2000.1100.
- 18 Shi Li. On uniform capacitated  $k$ -median beyond the natural lp relaxation. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'15*, pages 696–707, Philadelphia, PA, USA, 2015. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=2722129.2722176>.
- 19 Martin Pal, Eva Tardos, and Tom Wexler. Facility location with nonuniform hard capacities. In *Proceedings of the 42nd IEEE Symposium on the Foundations of Computer Science*, pages 329–338, 2001.
- 20 Qingzhou Wang and Kam Hoi Cheng. A heuristic algorithm for the  $k$ -center problem with vertex weight. In *Proceedings of the International Symposium on Algorithms, SIGAL'90*, pages 388–396, New York, NY, USA, 1990. Springer-Verlag New York, Inc. URL: <http://dl.acm.org/citation.cfm?id=88307.88459>.

## **A** Hardness of unweighted capacitated $k$ -center problem

The hardness result for the capacitated  $k$ -center problem by Cygan *et al.* in [10] also holds for bounded capacities. The reduction in [10] uses the EC3S problem in which there is no bound on the number of sets in  $\mathcal{F}$  an element of  $\mathcal{U}$  may belong to. This requires  $L$  to be



$\Theta(|\mathcal{F}|)$ . A different version of the EC3S problem in which each element of  $\mathcal{U}$  can belong to at most three sets in  $\mathcal{F}$  is also NP-complete [11] [12]. We use the same reduction as in [10]. Figure 2 illustrates the gadgets used in the reduction. Each vertex has a uniform capacity of  $L$  and  $k = |\mathcal{F}| + \frac{|\mathcal{U}|}{3}$ . All edges are of unit length.

► **Lemma 13.** *If there exists a solution to the EC3S instance, then there exist a capacitated  $k$ -center solution with cost  $\leq 1$ .*

**Proof.** Let  $\mathcal{F}' \subseteq \mathcal{F}$  be the solution of the EC3S instance. Note that,  $|\mathcal{F}'| = \frac{|\mathcal{U}|}{3}$ . For each set  $A \in \mathcal{F}$ , place a center at the vertex  $x_A$  in the corresponding set gadget. For each set  $A \in \mathcal{F}'$ , place a center at the vertex  $A$  in the corresponding set gadget. Thus, the vertices in each set gadget  $g_A$  is served by the center at vertex  $x_A$  and the vertices in the element gadget of the elements in a set  $A \in \mathcal{F}'$  are served by the center at  $A$ . The number of centers used is  $|\mathcal{F}| + |\mathcal{F}'| = |\mathcal{F}| + \frac{|\mathcal{U}|}{3} = k$ . ◀

Now consider a solution  $\mathcal{S}$  of the capacitated  $k$ -center instance with cost  $< 2$ . Note that each center must serve  $L$  vertices as  $|V| = kL$ .

► **Lemma 14.**  *$\mathcal{S}$  does not have a center in any of the element gadgets.*

**Proof.** Consider an element  $a \in \mathcal{U}$ . The vertices with distance  $< 2$  to a vertex in the element gadget  $g_a$  are the vertex itself and the vertices  $x_A$  for each set  $A \in \mathcal{F}$  that it belongs to. Since, each element can belong to at most three sets in  $\mathcal{F}$ , the number of vertices that can be assigned to a center in an element gadget is bounded by a constant. For sufficiently large but constant  $L$ , the center will not be able to serve  $L$  vertices. ◀

► **Lemma 15.** *Each set gadget in  $\mathcal{S}$  has at least one and at most two open centers in it.*

**Proof.** Consider the vertex set  $g_A$  of the set gadget for a set  $A \in \mathcal{F}$ . The  $L - 2$  pendant vertices in  $g_A$  cannot be served by a center outside  $g_A$ . Thus,  $g_A$  has at least one center in it.

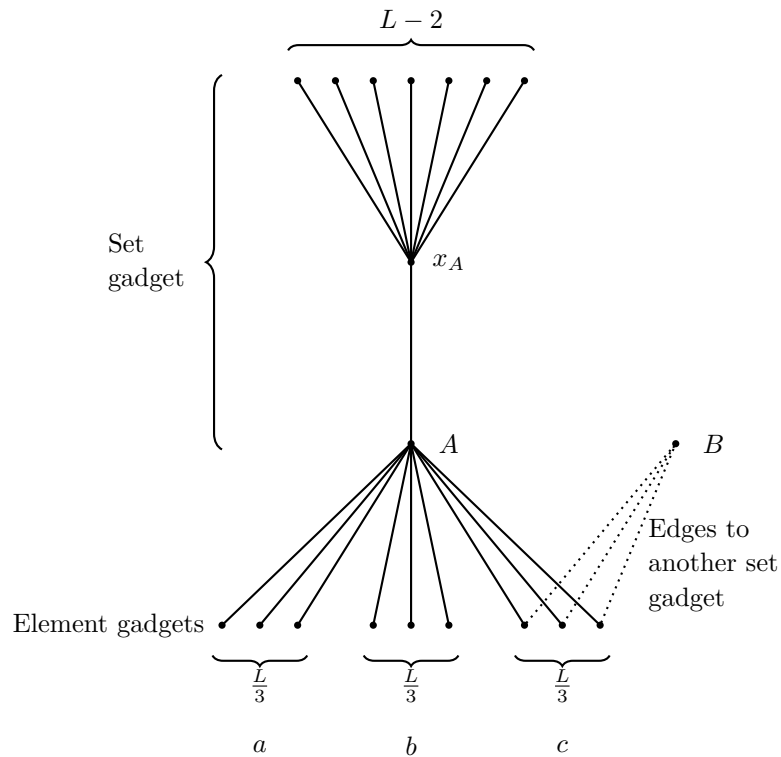
Assume, for contradiction, that the number of centers in  $g_A > 2$ . Let  $a, b$  and  $c$  be the elements in set  $A$  and let  $g_a, g_b$  and  $g_c$  be the vertex sets of their respective gadgets. The vertices that are at a distance  $< 2$  from some vertex  $g_A$  are the ones in  $g_A, g_a, g_b, g_c$ . Thus, the number of vertices that the centers in  $g_A$  can serve  $\leq |g_A| + |g_a| + |g_b| + |g_c| = 2L$ . Therefore, at least one of the centers in  $g_A$  must be serving less than  $L$  vertices. ◀

► **Lemma 16.** *In  $\mathcal{S}$ , gadgets corresponding to any two sets in  $\mathcal{F}$  sharing a common element cannot have two open centers in each one of them.*

**Proof.** Assume, for contradiction, that there exist two sets  $A, B \in \mathcal{F}$  having at least one element in common such that the vertex sets  $g_A$  and  $g_B$  of the corresponding gadgets each have 2 centers. As shown in the proof of Lemma 15, the vertices that can be assigned to a center in the gadget of a set  $C = \{d, e, f\}$  are those in  $g_C \cup g_d \cup g_e \cup g_f$ . Thus, the number of vertices that can be assigned to centers in  $g_A$  and  $g_B \leq 4L - \frac{L}{3}$  (since at least one element is common in  $A$  and  $B$ ). Therefore, at least one of the centers in  $g_A$  or  $g_B$  must be serving less than  $L$  vertices. ◀

► **Lemma 17.** *If there exists a capacitated  $k$ -center solution with cost  $R < 2$ , then there exists a solution to the EC3S instance.*

**Proof.** From Lemmas 14, 15 and 16, there are  $\frac{|\mathcal{U}|}{3}$  set gadgets each of which have 2 centers and the corresponding sets are all disjoint. These  $\frac{|\mathcal{U}|}{3}$  sets form the solution set  $\mathcal{F}'$ . ◀



■ **Figure 2** Gadgets for reduction (for the capacitated  $k$ -center problem).

► **Theorem 18.** *An  $\alpha$ -approximation is not possible for the uniform capacitated  $k$ -center problem for  $\alpha < 2$ , unless  $P = NP$ .*

**Proof.** Follows from Lemmas 13 and 17. ◀

► **Remark.** Using the same reduction and allowing capacities of  $L$  at vertices  $x_A$  and  $A$  in  $g_A$  for each set  $A \in \mathcal{F}$  and a capacity of zero at every other vertex, it can be shown that the  $\{0, L\}$ -capacitated  $k$ -center problem is hard to approximate within a factor of  $(3 - \epsilon)$  for a constant  $L$ .

# Improved Pseudo-Polynomial-Time Approximation for Strip Packing\*

Waldo Gálvez<sup>1</sup>, Fabrizio Grandoni<sup>2</sup>, Salvatore Ingala<sup>3</sup>, and Arindam Khan<sup>4</sup>

- 1 IDSIA, USI-SUPSI, Manno TI, Switzerland  
waldo@idsia.ch
- 2 IDSIA, USI-SUPSI, Manno TI, Switzerland  
fabrizio@idsia.ch
- 3 IDSIA, USI-SUPSI, Manno TI, Switzerland  
salvatore@idsia.ch
- 4 IDSIA, USI-SUPSI, Manno TI, Switzerland  
arindam@idsia.ch

---

## Abstract

We study the *strip packing* problem, a classical packing problem which generalizes both *bin packing* and *makespan minimization*. Here we are given a set of axis-parallel rectangles in the two-dimensional plane and the goal is to pack them in a vertical strip of fixed width such that the height of the obtained packing is minimized. The packing must be *non-overlapping* and the rectangles cannot be *rotated*.

A reduction from the *partition* problem shows that no approximation better than  $3/2$  is possible for strip packing in polynomial time (assuming  $P \neq NP$ ). Nadiradze and Wiese [SODA16] overcame this barrier by presenting a  $(\frac{7}{5} + \epsilon)$ -approximation algorithm in pseudo-polynomial-time (PPT). As the problem is strongly NP-hard, it does not admit an exact PPT algorithm (though a PPT approximation scheme might exist).

In this paper we make further progress on the PPT approximability of strip packing, by presenting a  $(\frac{4}{3} + \epsilon)$ -approximation algorithm. Our result is based on a non-trivial repacking of some rectangles in the *empty space* left by the construction by Nadiradze and Wiese, and in some sense pushes their approach to its limit.

Our PPT algorithm can be adapted to the case where we are allowed to rotate the rectangles by  $90^\circ$ , achieving the same approximation factor and breaking the polynomial-time approximation barrier of  $3/2$  for the case with rotations as well.

**1998 ACM Subject Classification** F.2 Analysis of Algorithms and Problem Complexity

**Keywords and phrases** approximation algorithms, strip packing, rectangle packing, cutting-stock problem

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.9

## 1 Introduction

In this paper, we consider the *strip packing* problem, a well-studied classical two-dimensional packing problem [6, 14, 28]. Here we are given a collection of rectangles, and an infinite vertical strip of width  $W$  in the two dimensional (2-D) plane. We need to find an axis-parallel

---

\* Partially supported by ERC Starting Grant NEWNET 279352 and SNSF Grant APXNET 200021\_159697/1.



© Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, and Arindam Khan;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 9; pp. 9:1–9:14



Leibniz International Proceedings in Informatics

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

embedding of the rectangles *without rotations* inside the strip so that no two rectangles overlap (feasible *packing*). Our goal is to minimize the total height of this packing.

More formally, we are given a parameter  $W \in \mathbb{N}$  and a set  $\mathcal{R} = \{R_1, \dots, R_n\}$  of rectangles, each one characterized by a width  $w_i \in \mathbb{N}$ ,  $w_i \leq W$ , and a height  $h_i \in \mathbb{N}$ . A packing of  $\mathcal{R}$  is a pair  $(x_i, y_i) \in \mathbb{N} \times \mathbb{N}$  for each  $R_i$ , with  $0 \leq x_i \leq W - w_i$ , meaning that the left-bottom corner of  $R_i$  is placed in position  $(x_i, y_i)$  and its right-top corner in position  $(x_i + w_i, y_i + h_i)$ . This packing is feasible if the interior of rectangles is disjoint in this embedding (or equivalently rectangles are allowed to overlap on their boundary only). Our goal is to find a feasible packing of minimum *height*  $\max_i \{y_i + h_i\}$ .

Strip packing is a natural generalization of *one-dimensional bin packing* [13] (when all the rectangles have the same height) and *makespan minimization* [12] (when all the rectangles have the same width). The problem has lots of applications in industrial engineering and computer science, specially in cutting stock, logistics and scheduling [28, 20]. Recently, there have been a lot of applications of strip packing in electricity allocation and peak demand reductions in smart-grids [36, 27, 32].

A simple reduction from the *partition* problem shows that the problem cannot be approximated within a factor  $\frac{3}{2} - \varepsilon$  for any  $\varepsilon > 0$  in polynomial-time unless  $P=NP$ . This reduction relies on exponentially large (in  $n$ ) rectangle widths.

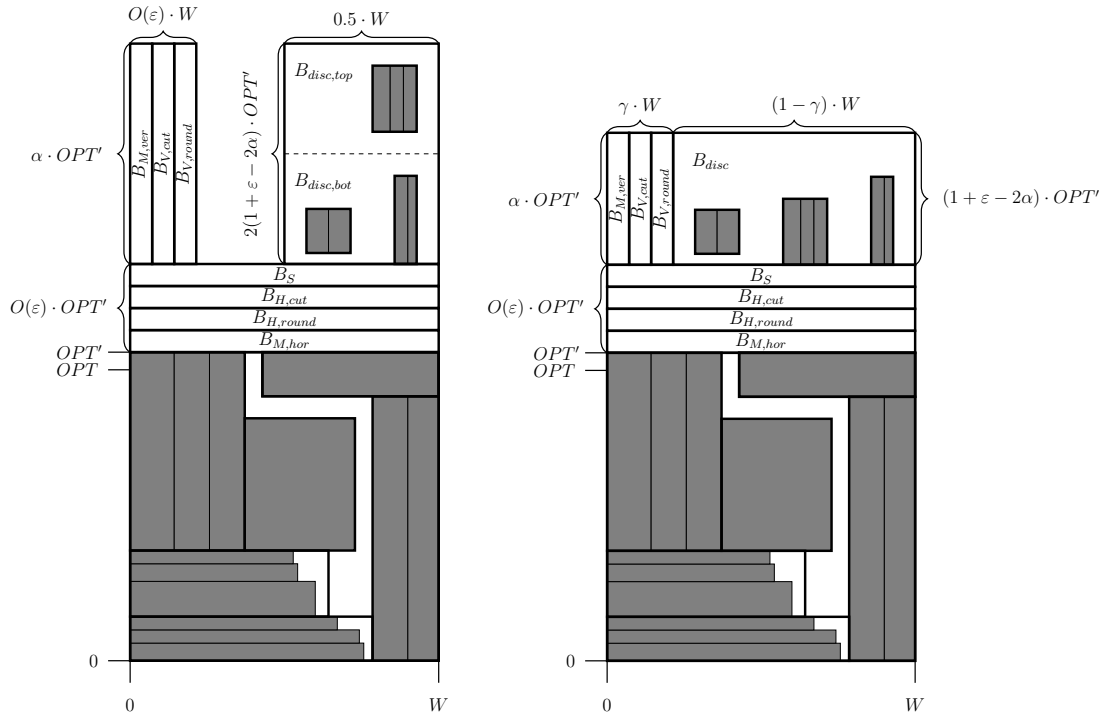
Let  $OPT = OPT(\mathcal{R})$  denote the optimal height for the considered strip packing instance  $(\mathcal{R}, W)$ , and  $h_{\max} = h_{\max}(\mathcal{R})$  (resp.  $w_{\max} = w_{\max}(\mathcal{R})$ ) be the largest height (resp. width) of any rectangle in  $\mathcal{R}$ . Observe that trivially  $OPT \geq h_{\max}$ . W.l.o.g. we can assume that  $W \leq nw_{\max}$ . The first non-trivial approximation algorithm for strip packing, with approximation ratio 3, was given by Baker, Coffman and Rivest [6]. The First-Fit-Decreasing-Height algorithm (FFDH) by Coffman et al. [14] gives a 2.7 approximation. Sleator [34] gave an algorithm that generates packing of height  $2OPT + \frac{h_{\max}}{2}$ , hence achieving a 2.5 approximation. Afterwards, Steinberg [35] and Schiermeyer [33] independently improved the approximation ratio to 2. Harren and van Stee [21] first broke the barrier of 2 with their 1.9396 approximation. The present best  $(\frac{5}{3} + \varepsilon)$ -approximation is due to Harren et al. [20].

Nadiradze and Wiese [31] overcame the  $\frac{3}{2}$ -inapproximability barrier by presenting a  $(\frac{7}{5} + \varepsilon)$ -approximation algorithm running in pseudo-polynomial-time (PPT). More specifically, the running time of their algorithm is  $O((Nn)^{O(1)})$ , where  $N = \max\{w_{\max}, h_{\max}\}$ <sup>1</sup>. As strip packing is strongly NP-hard [17], it does not admit an exact PPT algorithm. However, the existence of a PPT approximation scheme is currently not excluded.

## 1.1 Our contribution and techniques

In this paper, we make progress on the PPT approximability of strip packing, by presenting an improved  $(\frac{4}{3} + \varepsilon)$  approximation. Our approach refines the technique of Nadiradze and Wiese [31], that modulo several technical details works as follows. Let  $\alpha \in [1/3, 1/2)$  be a proper constant parameter, and define a rectangle  $R_i$  to be *tall* if  $h_i > \alpha \cdot OPT$ . They prove that the optimal packing can be structured into a constant number of axis-aligned rectangular regions (*boxes*), that occupy a total height of  $OPT' \leq (1 + \varepsilon)OPT$  inside the vertical strip. Some rectangles are not fully contained into one box (they are *cut* by some box). Among them, tall rectangles remain in their original position. All the other cut rectangles are repacked on top of the boxes: part of them in a horizontal box of size  $W \times O(\varepsilon)OPT$ ,

<sup>1</sup> For the case without rotations, the polynomial dependence on  $h_{\max}$  can indeed be removed with standard techniques.



(a) Final packing obtained by Nadiradze & Wiese [31].

(b) Final packing obtained in this work. Here  $\gamma$  is a small constant depending on  $\varepsilon$ .

■ **Figure 1** Comparison of final solutions.

and the remaining ones in a vertical box of size  $O(\varepsilon W) \times \alpha OPT$  (that we next imagine as placed on the top-left of the packing under construction).

Some of these boxes contain only relatively high rectangles (including tall ones) of relatively small width. The next step is a rearrangement of the rectangles inside one such *vertical* box  $\bar{B}$  (see Figure 3a), say of size  $\bar{w} \times \bar{h}$ : they first slice non-tall rectangles into unit width rectangles (this slicing can be finally avoided with standard techniques). Then they shift tall rectangles to the top/bottom of  $\bar{B}$ , shifting sliced rectangles consequently (see Figure 3b). Now they discard all the (sliced) rectangles completely contained in a central horizontal region of size  $\bar{w} \times (1 + \varepsilon - 2\alpha)\bar{h}$ , and they *nicely rearrange* the remaining rectangles into a constant number of *sub-boxes* (excluding possibly a few more non-tall rectangles, that can be placed in the additional vertical box).

These discarded rectangles can be packed into 2 extra boxes of size  $\frac{\bar{w}}{2} \times (1 + \varepsilon - 2\alpha)\bar{h}$  (see Figure 3d). In turn, the latter boxes can be packed into two *discarded* boxes of size  $\frac{W}{2} \times (1 + \varepsilon - 2\alpha)OPT'$ , that we can imagine as placed, one on top of the other, on the top-right of the packing. See Figure 1a for an illustration of the final packing. This leads to a total height of  $(1 + \max\{\alpha, 2(1 - 2\alpha)\} + O(\varepsilon)) \cdot OPT$ , which is minimized by choosing  $\alpha = \frac{2}{5}$ .

Our main technical contribution is a repacking lemma that allows one to repack a small fraction of the discarded rectangles of a given box inside the free space left by the corresponding sub-boxes (while still having  $O_\varepsilon(1)$  many sub-boxes in total). This is illustrated in Figure 3e. This way we can pack all the discarded rectangles into a *single* discarded box of size  $(1 - \gamma)W \times (1 + \varepsilon - 2\alpha)OPT'$ , where  $\gamma$  is a small constant depending on  $\varepsilon$ , that we

can place on the top-right of the packing. The vertical box where the remaining rectangles are packed still fits to the top-left of the packing, next to the discarded box. See Figure 1b for an illustration. Choosing  $\alpha = 1/3$  gives the claimed approximation factor.

We remark that the basic approach by Nadiradze and Wiese strictly requires that at most 2 tall rectangles can be packed one on top of the other in the optimal packing, hence imposing  $\alpha \geq 1/3$ . Thus in some sense we pushed their approach to its limit.

The algorithm by Nadiradze and Wiese [31] is not directly applicable to the case when  $90^\circ$  rotations are allowed. In particular, they use a linear program to pack some rectangles. When rotations are allowed, it is unclear how to decide which rectangles are packed by the linear program. We use a combinatorial *container*-based approach to circumvent this limitation, which allows us to pack all the rectangles using dynamic programming. This way we achieve a PPT  $(4/3 + \varepsilon)$ -approximation for strip packing with rotations, breaking the polynomial-time approximation barrier of  $3/2$  for that variant as well.

## 1.2 Related work

For packing problems, many pathological lower bound instances occur when  $OPT$  is small. Thus it is often insightful to consider the *asymptotic approximation ratio*. Coffman et al. [14] described two *level-oriented* algorithms, Next-Fit-Decreasing-Height (NFDH) and First-Fit-Decreasing-Height (FFDH), that achieve asymptotic approximations of 2 and 1.7, respectively. After a sequence of improvements [18, 5], the seminal work of Kenyon and Rémila [28] provided an asymptotic polynomial-time approximation scheme (APTAS) with an additive term  $O(\frac{h_{max}}{\varepsilon^2})$ . The latter additive term was subsequently improved to  $h_{max}$  by Jansen and Solis-Oba [24].

In the variant of strip packing *with rotations*, we are allowed to rotate the input rectangles by  $90^\circ$  (in other terms, we are free to swap the width and height of an input rectangle). The case with rotations is much less studied in the literature. It seems that most techniques that work for the case without rotations can be extended to the case with rotations, however this is not always a trivial task. In particular, it is not hard to achieve a  $2 + \varepsilon$  approximation, and the  $3/2$  hardness of approximation extends to this case as well [24]. In terms of asymptotic approximation, Miyazawa and Wakabayashi [30] gave an algorithm with asymptotic performance ratio of 1.613. Later, Epstein and van Stee [16] gave a  $\frac{3}{2}$  asymptotic approximation. Finally, Jansen and van Stee [25] achieved an APTAS for the case with rotations.

Strip packing has also been well studied for higher dimensions. The present best asymptotic approximation for 3-D strip packing is due to Jansen and Prädél [23] who gave 1.5-approximation extending techniques from 2-D bin packing.

There are many other related geometric packing problems. For example, in the *independent set of rectangles* problems we are given a collection of axis-parallel rectangles embedded in the plane, and we need to find a maximum cardinality/weight subset of non-overlapping rectangles [1, 10, 11]. Interesting connections between this problems and *unsplittable flow on a path* were recently discovered [3, 4, 7, 9, 19]. In the *geometric knapsack* problem we wish to pack a maximum cardinality/profit subset of the rectangles in a given square knapsack [2, 26]. One can also consider a natural geometric version of bin packing, where one needs to pack a given set of rectangles in the smallest possible number of square bins [8]. We refer the readers to [29] for a survey on geometric packing problems.

### 1.3 Organization of the paper

First, we discuss some preliminaries and notations in Section 2. Section 3 contains our main technical contribution, our *repacking lemma*. There we also discuss a refined structural result leading to a packing into  $O_\varepsilon(1)$  many *containers*. In Section 4, we describe our algorithm to pack the rectangles. Then in Section 5, we extend our algorithm to the case with rotations. Finally, in Section 6, we conclude with some observations.

Due to space constraints, some proofs are omitted from this extended abstract and will appear in the full version of the paper.

## 2 Preliminaries and notations

Throughout the present work, we will follow the notation from [31], which will be explained as it is needed.

Recall that  $OPT \in \mathbb{N}$  denotes the height of the optimal packing for instance  $\mathcal{R}$ . By trying all the pseudo-polynomially many possibilities, we can assume that  $OPT$  is known to the algorithm. Given a set  $\mathcal{M} \subseteq \mathcal{R}$  of rectangles,  $a(\mathcal{M})$  will denote the total area of rectangles in  $\mathcal{M}$ , i.e.,  $a(\mathcal{M}) = \sum_{R_i \in \mathcal{M}} h_i \cdot w_i$ , and  $h_{\max}(\mathcal{M})$  (resp.  $w_{\max}(\mathcal{M})$ ) denotes the maximum height (resp. width) of rectangles in  $\mathcal{M}$ . Throughout this work, a *box* of size  $a \times b$  means an axis-aligned rectangular region of width  $a$  and height  $b$ .

In order to lighten the notation, we sometimes interpret a rectangle/box as the corresponding region inside the strip according to some given embedding. The latter embedding will not be specified when clear from the context. Similarly, we sometimes describe an embedding of some rectangles inside a box, and then embed the box inside the strip: the embedding of the considered rectangles is shifted consequently in that case.

A vertical (resp. horizontal) *container* is an axis-aligned rectangular region where we implicitly assume that rectangles are packed one next to the other from left to right (resp., bottom to top), i.e., any vertical (resp. horizontal) line intersects only one packed rectangle (see Figure 2b). Container-like packings will turn out to be particularly useful since they naturally induce a (one-dimensional) knapsack instance.

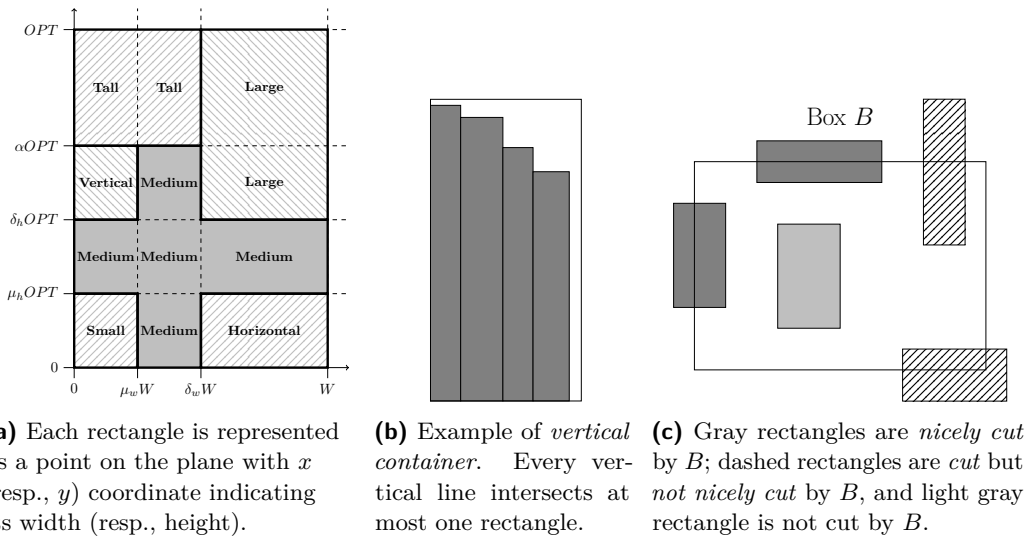
### 2.1 Classification of rectangles

Let  $0 < \varepsilon < \alpha$ , and assume for simplicity that  $\frac{1}{\varepsilon} \in \mathbb{N}$ . We first classify the input rectangles into six groups according to parameters  $\delta_h, \delta_w, \mu_h, \mu_w$  satisfying  $\varepsilon \geq \delta_h > \mu_h > 0$  and  $\varepsilon \geq \delta_w > \mu_w > 0$ , whose values will be chosen later (see also Figure 2a). A rectangle  $R_i$  is

- *Large* if  $h_i \geq \delta_h OPT$  and  $w_i \geq \delta_w W$ .
- *Tall* if  $h_i > \alpha OPT$  and  $w_i < \delta_w W$ .
- *Vertical* if  $h_i \in [\delta_h OPT, \alpha OPT]$  and  $w_i \leq \mu_w W$ ,
- *Horizontal* if  $h_i \leq \mu_h OPT$  and  $w_i \geq \delta_w W$ ,
- *Small* if  $h_i \leq \mu_h OPT$  and  $w_i \leq \mu_w W$ ;
- *Medium* in all the remaining cases, i.e., if  $h_i \in (\mu_h OPT, \delta_h OPT)$ , or  $w_i \in (\mu_w W, \delta_w W)$  and  $h_i \leq \alpha OPT$ .

We use  $L, T, V, H, S$ , and  $M$  to denote large, tall, vertical, horizontal, small, and medium rectangles, respectively. We remark that, differently from [31], we need to allow  $\delta_h \neq \delta_w$  and  $\mu_h \neq \mu_w$  due to some additional constraints in our construction (see Section 4).

Notice that according to this classification, every vertical line across the optimal packing intersects at most two tall rectangles. The following lemma allows us to choose  $\delta_h, \delta_w, \mu_h$



■ **Figure 2** Illustration of some of the definitions used in this paper.

and  $\mu_w$  in such a way that  $\delta_h$  and  $\mu_h$  ( $\delta_w$  and  $\mu_w$ , respectively) differ by a large factor, and medium rectangles have small total area.

► **Lemma 1.** *Given a polynomial-time computable function  $f : (0, 1) \rightarrow (0, 1)$ , with  $f(x) < x$ , any constant  $\varepsilon \in (0, 1)$ , and any positive integer  $k$ , we can compute in polynomial time a set  $\Delta$  of  $T = 2(\frac{1}{\varepsilon})^k$  many positive real numbers upper bounded by  $\varepsilon$ , such that there is at least one number  $\delta_h \in \Delta$  so that  $a(M) \leq \varepsilon^k \cdot OPT \cdot W$  by choosing  $\mu_h = f(\delta_h)$ ,  $\mu_w = \frac{\varepsilon \mu_h}{12}$ , and  $\delta_w = \frac{\varepsilon \delta_h}{12}$ .*

Function  $f$  and constant  $k$  will be chosen later. From now on, assume that  $\delta_h, \delta_w, \mu_h$  and  $\mu_w$  are chosen according to Lemma 1.

## 2.2 Overview of the algorithm

We next overview some of the basic results in [31] that are needed in our result. We define the constant  $\gamma := \frac{\varepsilon \delta_h}{2}$ , and w.l.o.g. assume  $\gamma \cdot OPT \in \mathbb{N}$ .

Let us forget for a moment small rectangles  $S$ . We will pack all the remaining rectangles  $L \cup H \cup T \cup V \cup M$  into a sufficiently small number of boxes embedded into the strip. By standard techniques, as in [31], it is then possible to pack  $S$  (essentially using NFDH in a proper grid defined by the above boxes) while increasing the total height at most by  $O(\varepsilon)OPT$ . See Section 4.1 for more details on packing of small rectangles.

The following lemma from [31] allows one to round the heights and positions of rectangles of large enough height, without increasing much the height of the packing.

► **Lemma 2** ([31]). *There exists a feasible packing of height  $OPT' \leq (1 + \varepsilon)OPT$  where: (1) the height of each rectangles in  $L \cup T \cup V$  is rounded up to the closest integer multiple of  $\gamma \cdot OPT$  and (2) their  $x$ -coordinates are as in the optimal solution and their  $y$ -coordinates are integer multiples of  $\gamma \cdot OPT$ .*

We next focus on rounded rectangle heights (i.e., implicitly replace  $L \cup T \cup V$  by their rounded version) and on this slightly suboptimal solution of height  $OPT'$ .

The following lemma helps us to pack rectangles in  $M$ .



► **Lemma 3.** *If  $k$  in Lemma 1 is chosen sufficiently large, all the rectangles in  $M$  can be packed in polynomial time into a box  $B_{M,hor}$  of size  $W \times O(\varepsilon)OPT$  and a box  $B_{M,ver}$  of size  $(\frac{\gamma}{3}W) \times (\alpha OPT)$ . Furthermore, there is one such packing using  $\frac{3\varepsilon}{\mu_h}$  vertical containers in  $B_{M,hor}$  and  $\frac{\gamma}{3\mu_w}$  horizontal containers in  $B_{M,ver}$ .*

We say that a rectangle  $R_i$  is *cut* by a box  $B$  if both  $R_i \setminus B$  and  $B \setminus R_i$  are non-empty (considering both  $R_i$  and  $B$  as open regions with an implicit embedding on the plane). We say that a rectangle  $R_i \in H$  (resp.  $R_i \in T \cup V$ ) is *nicely cut* by a box  $B$  if  $R_i$  is cut by  $B$  and their intersection is a rectangular region of width  $w_i$  (resp. height  $h_i$ ). Intuitively, this means that an edge of  $B$  cuts  $R_i$  along its longest side (see Figure 2c).

Now it remains to pack  $L \cup H \cup T \cup V$ : The following lemma, taken from [31] modulo minor technical adaptations, describes an almost optimal packing of those rectangles.

► **Lemma 4.** *There is an integer  $K_B = (\frac{1}{\varepsilon})(\frac{1}{\delta_w})^{O(1)}$  such that, assuming  $\mu_h \leq \frac{\varepsilon\delta_w}{K_B}$ , there is a partition of the region  $B_{OPT'} := [0, W] \times [0, OPT']$  into a set  $\mathcal{B}$  of at most  $K_B$  boxes and a packing of the rectangles in  $L \cup T \cup V \cup H$  such that:*

- each box has size equal to the size of some  $R_i \in L$  (large box), or has height at most  $\delta_h OPT'$  (horizontal box), or has width at most  $\delta_w W$  (vertical box);
- each  $R_i \in L$  is contained into a large box of the same size;
- each  $R_i \in H$  is contained into a horizontal box or is cut by some box. Furthermore, the total area of horizontal cut rectangles is at most  $W \cdot O(\varepsilon)OPT'$ ;
- each  $R_i \in T \cup V$  is contained into a vertical box or is nicely cut by some vertical box.

We denote the sets of vertical, horizontal, and large boxes by  $\mathcal{B}_V, \mathcal{B}_H$  and  $\mathcal{B}_L$ , respectively. Observe that  $\mathcal{B}$  can be guessed in PPT. We next use  $T_{cut} \subseteq T$  and  $V_{cut} \subseteq V$  to denote tall and vertical cut rectangles in the above lemma, respectively. Let us also define  $T_{box} = T \setminus T_{cut}$  and  $V_{box} = V \setminus V_{cut}$ .

Using standard techniques (see e.g. [31]), we can pack all the rectangles excluding the ones contained in vertical boxes in a convenient manner. This is summarized in the following lemma.

► **Lemma 5.** *Given  $\mathcal{B}$  as in Lemma 4 and assuming  $\mu_w \leq \frac{\gamma\delta_h}{6K_B(1+\varepsilon)}$ , there exists a packing of  $L \cup H \cup T \cup V$  such that:*

1. all the rectangles in  $L$  are packed in  $\mathcal{B}_L$ ;
2. all the rectangles in  $H$  are packed in  $\mathcal{B}_H$  plus an additional box  $B_{H,cut}$  of size  $W \times O(\varepsilon)OPT$ ;
3. all the rectangles in  $T_{cut} \cup T_{box} \cup V_{box}$  are packed as in Lemma 4;
4. all the rectangles in  $V_{cut}$  are packed in an additional vertical box  $B_{V,cut}$  of size  $(\frac{\gamma}{3}W) \times (\alpha OPT)$ .

We will pack all the rectangles (essentially) as in [31], with the exception of  $T_{box} \cup V_{box}$  where we exploit a refined approach. This is the technical heart of this paper, and it is discussed in the next section.

### 3 A repacking lemma

We next describe how to pack rectangles in  $T_{box} \cup V_{box}$ . In order to highlight our contribution, we first describe how the approach by Nadiradze and Wiese [31] works.

It is convenient to assume that all the rectangles in  $V_{box}$  are sliced vertically into sub-rectangles of width 1 each<sup>2</sup>. Let  $V_{sliced}$  be such *sliced* rectangles. We will show how to pack all the rectangles in  $T_{box} \cup V_{sliced}$  into a constant number of sub-boxes. Using standard techniques it is then possible to pack  $V_{box}$  into the space occupied by  $V_{sliced}$  plus an additional box  $B_{V,round}$  of size  $(\frac{\gamma}{3}W) \times \alpha OPT$ .

We next focus on a specific vertical box  $\bar{B}$ , say of size  $\bar{w} \times \bar{h}$  (see Figure 3a). Let  $\bar{T}_{cut}$  be the tall rectangles cut by  $\bar{B}$ . Observe that there are at most 4 such rectangles (2 on the left/right side of  $\bar{B}$ ). The rectangles in  $\bar{T}_{cut}$  are packed as in Lemma 5. Let also  $\bar{T}$  and  $\bar{V}$  be the tall rectangles and sliced vertical rectangles, respectively, originally packed completely inside  $\bar{B}$ .

They show that it is possible to pack  $\bar{T} \cup \bar{V}$  into a constant size set  $\bar{S}$  of sub-boxes contained inside  $\bar{B} - \bar{T}_{cut}$ , plus an additional box  $\bar{D}$  of size  $\bar{w} \times (1 + \varepsilon - 2\alpha)\bar{h}$ . Here  $\bar{B} - \bar{T}_{cut}$  denotes the region inside  $\bar{B}$  not contained in  $\bar{T}_{cut}$ . In more detail, they start by considering each rectangle  $R_i \in \bar{T}$ . Since  $\alpha \geq \frac{1}{3}$  by assumption, one of the regions above or below  $R_i$  cannot contain another tall rectangle in  $\bar{T}$ , say the first case applies (the other one being symmetric). Then we move  $R_i$  up so that its top side overlaps with the top side of  $\bar{B}$ . The sliced rectangles in  $\bar{V}$  that are covered this way are shifted right below  $R$  (note that there is enough free space by construction). At the end of the process all the rectangles in  $\bar{T}$  touch at least one of the top and bottom side of  $\bar{B}$  (see Figure 3b). Note that no rectangle is discarded up to this point.

Next, we partition the space inside  $\bar{B} - (\bar{T} \cup \bar{T}_{cut})$  into maximal height unit-width vertical stripes. We call each such stripe a *free rectangle* if both its top and bottom side overlap with the top or bottom side of some rectangle in  $\bar{T} \cup \bar{T}_{cut}$ , and otherwise a *pseudo rectangle* (see Figure 3c). We define the  $i$ -th free rectangle to be the free rectangle contained in stripe  $[i - 1, i] \times [0, \bar{h}]$ .

Note that all the free rectangles are contained in a horizontal region of width  $\bar{w}$  and height at most  $\bar{h} - 2\alpha OPT \leq \bar{h} - 2\alpha \frac{OPT'}{1+\varepsilon} \leq \bar{h}(1 - \frac{2\alpha}{1+\varepsilon}) \leq \bar{h}(1 + \varepsilon - 2\alpha)$  contained in the central part of  $\bar{B}$ . Let  $\bar{V}_{disc}$  be the set of (sliced vertical) rectangles contained in the free rectangles. Rectangles in  $\bar{V}_{disc}$  can be obviously packed inside  $\bar{D}$ . For each corner  $Q$  of the box  $\bar{B}$ , we consider the maximal rectangular region that has  $Q$  as a corner and only contains pseudo rectangles whose top/bottom side overlaps with the bottom/top side of a rectangle in  $\bar{T}_{cut}$ ; there are at most 4 such non-empty regions, and for each of them we define a *corner sub-box*, and we call the set of such sub-boxes  $\bar{B}_{corn}$  (see Figure 3c). The final step of the algorithm is to rearrange horizontally the pseudo/tall rectangles so that pseudo/tall rectangles of the same height are grouped together *as much as possible* (modulo some technical details). The rectangles in  $\bar{B}_{corn}$  are not moved. The *sub-boxes* are induced by maximal consecutive subsets of pseudo/tall rectangles of the same height touching the top (resp., bottom) side of  $\bar{B}$  (see Figure 3d). We crucially remark that, by construction, the height of each sub-box (and of  $\bar{B}$ ) is a multiple of  $\gamma OPT$ .

By splitting each discarded box  $\bar{D}$  into two halves  $\bar{B}_{disc,top}$  and  $\bar{B}_{disc,bot}$ , and replicating the packing of boxes inside  $B_{OPT'}$ , it is possible to pack all the discarded boxes into two boxes  $B_{disc,top}$  and  $B_{disc,bot}$ , both of size  $\frac{W}{2} \times (1 + \varepsilon - 2\alpha)OPT'$ .

A feasible packing of boxes (and hence of the associated rectangles) of height  $(1 + \max\{\alpha, 2(1 - 2\alpha)\} + O(\varepsilon))OPT$  is then obtained as follows. We first pack  $B_{OPT'}$  at the base of the strip, and then on top of it we pack  $B_{M,hor}$ , two additional boxes  $B_{H,round}$  and  $B_{H,cut}$  (which will be used to repack the horizontal items), and a box  $B_S$  (which will be used to pack

<sup>2</sup> For technical reasons, slices have width 1/2 in [31]. For our algorithm, slices of width 1 suffice.

some of the small items). The latter 4 boxes all have width  $W$  and height  $O(\varepsilon OPT')$ . On the top right of this packing we place  $B_{disc,top}$  and  $B_{disc,bot}$ , one on top of the other. Finally, we pack  $B_{M,ver}$ ,  $B_{V,cut}$  and  $B_{V,round}$  on the top left, one next to the other. See Figure 1a for an illustration. The height is minimized for  $\alpha = \frac{2}{5}$ , leading to a  $7/5 + O(\varepsilon)$  approximation.

The main technical contribution of this paper is to show how it is possible to repack a subset of  $\bar{V}_{disc}$  into the *free* space inside  $\bar{B}_{cut} := \bar{B} - \bar{T}_{cut}$  not occupied by sub-boxes, so that the residual sliced rectangles can be packed into a single discarded box  $\bar{B}_{disc}$  of size  $(1 - \gamma)\bar{w} \times (1 + \varepsilon - 2\alpha)\bar{h}$  (*repacking lemma*). See Figure 3e. This apparently minor saving is indeed crucial: with the same approach as above all the discarded sub-boxes  $\bar{B}_{disc}$  can be packed into a single *discarded box*  $B_{disc}$  of size  $(1 - \gamma)W \times (1 + \varepsilon - 2\alpha)OPT'$ . Therefore, we can pack all the previous boxes as before, and  $B_{disc}$  on the top right. Indeed, the total width of  $B_{M,ver}$ ,  $B_{V,cut}$  and  $B_{V,round}$  is at most  $\gamma W$  for a proper choice of the parameters. See Figure 1b for an illustration. Altogether the resulting packing has height  $(1 + \max\{\alpha, 1 - 2\alpha\} + O(\varepsilon))OPT$ . This is minimized for  $\alpha = \frac{1}{3}$ , leading to the claimed  $4/3 + O(\varepsilon)$  approximation.

It remains to prove our repacking lemma.

► **Lemma 6 (Repacking Lemma).** *Consider a partition of  $\bar{D}$  into  $\bar{w}$  unit-width vertical stripes. There is a subset of at least  $\gamma\bar{w}$  such stripes so that the corresponding sliced vertical rectangles  $\bar{V}_{repack}$  can be repacked inside  $\bar{B}_{cut} = \bar{B} - \bar{T}_{cut}$  in the space not occupied by sub-boxes.*

**Proof.** Let  $f(i)$  denote the height of the  $i$ -th free rectangle, where for notational convenience we introduce a degenerate free rectangle of height  $f(i) = 0$  whenever the stripe  $[i - 1, i] \times [0, \bar{h}]$  inside  $\bar{B}$  does not contain any free rectangle. This way we have precisely  $\bar{w}$  free rectangles. We remark that free rectangles are defined before the horizontal rearrangement of tall/pseudo rectangles, and the consequent definition of sub-boxes.

Recall that sub-boxes contain tall and pseudo rectangles. Now consider the area in  $\bar{B}_{cut}$  not occupied by sub-boxes. Note that this area is contained in the central region of height  $\bar{h}(1 - \frac{2\alpha}{1+\varepsilon})$ . Partition this area into maximal-height unit-width vertical stripes as before (*newly free rectangles*). Let  $g(i)$  be the height of the  $i$ -th newly free rectangle, where again we let  $g(i) = 0$  if the stripe  $[i - 1, i] \times [0, \bar{h}]$  does not contain any (positive area) free region. Note that, since tall and pseudo rectangles are only shifted horizontally in the rearrangement, it must be the case that:

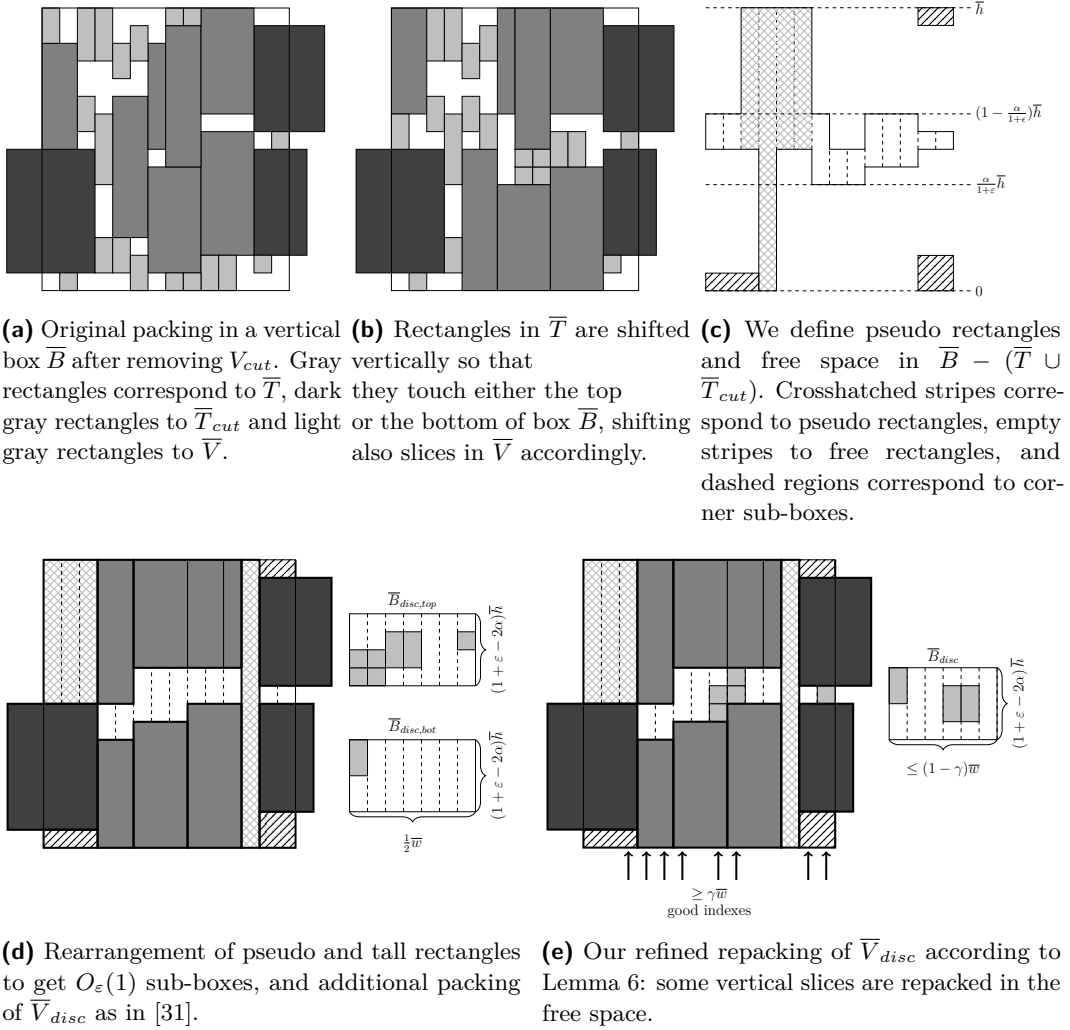
$$\sum_{i=1}^{\bar{w}} f(i) = \sum_{i=1}^{\bar{w}} g(i).$$

Let  $G$  be the (*good*) indexes where  $g(i) \geq f(i)$ , and  $\bar{G} = \{1, \dots, \bar{w}\} - G$  be the (*bad*) indexes with  $g(i) < f(i)$ . Observe that for each  $i \in G$ , it is possible to pack the  $i$ -th free rectangle inside the  $i$ -th newly free rectangle, therefore freeing a unit-width vertical strip inside  $\bar{D}$ . Thus it is sufficient to show that  $|G| \geq \gamma\bar{w}$ .

Observe that, for  $i \in \bar{G}$ ,  $f(i) - g(i) \geq \gamma OPT \geq \gamma \frac{\bar{h}}{1+\varepsilon}$ : indeed, both  $f(i)$  and  $g(i)$  must be multiples of  $\gamma OPT$  since they correspond to the height of  $\bar{B}$  minus the height of one or two tall/pseudo rectangles. On the other hand, for any index  $i$ ,  $g(i) - f(i) \leq g(i) \leq (1 - \frac{2\alpha}{1+\varepsilon})\bar{h}$ , by the definition of  $g$ . Altogether

$$(1 - \frac{2\alpha}{1+\varepsilon})\bar{h} \cdot |G| \geq \sum_{i \in G} (g(i) - f(i)) = \sum_{i \in \bar{G}} (f(i) - g(i)) \geq \frac{\gamma\bar{h}}{1+\varepsilon} \cdot |\bar{G}| = \frac{\gamma\bar{h}}{1+\varepsilon} \cdot (\bar{w} - |G|).$$

We conclude that  $|G| \geq \frac{\gamma}{1+\varepsilon-2\alpha+\gamma}\bar{w}$ . The claim follows since by assumption  $\alpha > \varepsilon \geq \gamma$ . ◀



■ **Figure 3** Creation of pseudo rectangles, how to get constant number of sub-boxes and repacking of vertical slices in a vertical box  $\bar{B}$ .

The original algorithm in [31] use standard LP-based techniques, as in [28], to pack the horizontal rectangles. We can avoid that via a refined structural lemma: here boxes and sub-boxes are further partitioned into vertical (resp., horizontal) containers. Rectangles are then packed into such containers as mentioned earlier: one next to the other from left to right (resp., bottom to top). Containers define a multiple knapsack instance, that can be solved optimally in PPT via dynamic programming. This approach has two main advantages:

- It leads to a simpler algorithm.
- It can be easily adapted to the case with rotations, as discussed in Section 5.

We omit the proof of the following Lemma.

► **Lemma 7.** *By choosing  $\alpha = 1/3$ , there is an integer  $K_F \leq \left(\frac{1}{\epsilon \delta_w}\right)^{O(1/(\delta_w \epsilon))}$  such that, assuming  $\mu_h \leq \frac{\epsilon}{K_F}$  and  $\mu_w \leq \frac{\gamma}{3K_F}$ , there is a packing of  $\mathcal{R} \setminus S$  in the region  $[0, W] \times [0, (4/3 + O(\epsilon))OPT']$  with the following properties:*

- All the rectangles in  $\mathcal{R} \setminus S$  are contained in  $K_{TOTAL} = O_\varepsilon(1)$  horizontal or vertical containers, such that each of these containers is either contained in or disjoint from  $\mathcal{B}_{OPT'}$ ;
- At most  $K_F$  containers are contained in  $\mathcal{B}_{OPT'}$ , and their total area is at most  $a(\mathcal{R} \setminus S)$ .

#### 4 A refined algorithm

First of all, we find  $\mu_h, \delta_h, \mu_w, \delta_w$  as required by Lemma 1; this way, we can find the set  $S$  of small rectangles. Consider the packing of Lemma 7: all the non-small rectangles are packed into  $K_{TOTAL} = O_\varepsilon(1)$  containers, and only  $K_F$  of them are contained in  $\mathcal{B}_{OPT'}$ . Since their position  $(x, y)$  and their size  $(w, h)$  are w.l.o.g. contained in  $\{0, \dots, W\} \times \{0, \dots, nh_{max}\}$ , we can enumerate in PPT over all the possible feasible such packings of  $k \leq K_{TOTAL}$  containers, and one of those will coincide with the packing defined by Lemma 7.

Containers naturally induce a multiple knapsack problem: for each horizontal container  $C_j$  of size  $w_{C_j} \times h_{C_j}$ , we create a (one-dimensional) knapsack  $j$  of size  $h_{C_j}$ . Furthermore, we define the size  $b(i, j)$  of rectangle  $R_i$  w.r.t. knapsack  $j$  as  $h_i$  if  $h_i \leq h_{C_j}$  and  $w_i \leq w_{C_j}$ . Otherwise  $b(i, j) = +\infty$  (meaning that  $R_i$  does not fit in  $C_j$ ). The construction for vertical containers is symmetric. This multiple knapsack problem can be easily solved optimally (hence packing all the rectangles) in PPT via dynamic programming.

Note that unlike [31], we do not use linear programming to pack horizontal rectangles, which will be crucial when we extend our approach to the case with rotations.

##### 4.1 Packing the small rectangles

It remains to pack the small rectangles  $S$ . We will pack them in the free space left by containers inside  $[0, W] \times [0, OPT']$  plus an additional box  $B_S$  of small height as the following lemma states. By placing box  $B_S$  on top of the remaining packed rectangles, the final height of the solution increases only by  $\varepsilon \cdot OPT'$ .

► **Lemma 8.** *Assuming  $\mu_h \leq \frac{1}{31K_F^2}$ , it is possible to pack in polynomial time all the rectangles in  $S$  into the area  $[0, W] \times [0, OPT']$  not occupied by containers plus an additional box  $B_S$  of size  $W \times \varepsilon OPT'$ .*

**Proof.** We first extend the sides of the containers inside  $[0, W] \times [0, OPT']$  in order to define a grid. This procedure partitions the free space in  $[0, W] \times [0, OPT']$  into a constant number of rectangular regions (at most  $(2K_F + 1)^2 \leq 5K_F^2$  many) whose total area is at least  $a(S)$  thanks to Lemma 7. Let  $\mathcal{B}_{small}$  be the set of such rectangular regions with width at least  $\mu_w W$  and height at least  $\mu_h OPT'$  (notice that the total area of rectangular regions not in  $\mathcal{B}_{small}$  is at most  $5K_F^2 \mu_w \mu_h \cdot W \cdot OPT'$ ). We now use NFDH to pack a subset of  $S$  into the regions in  $\mathcal{B}_{small}$ . By standard properties of NFDH, since each region in  $\mathcal{B}_{small}$  has size at most  $W \times OPT'$  and each item in  $S$  has width at most  $\mu_w W$  and height at most  $\mu_h OPT'$ , the total area of the unpacked rectangles from  $S$  can be bounded above by  $5K_F^2 \cdot (\mu_w \mu_h W OPT' + \mu_h OPT' \cdot W + \mu_w W \cdot OPT') \leq 15K_F^2 \mu_h \cdot OPT' \cdot W$ . Therefore we can pack the latter small rectangles with NFDH in an additional box  $B_S$  of width  $W$  and height  $\mu_h OPT' + 30K_F^2 \mu_h OPT' \leq \varepsilon \cdot OPT'$  provided that  $\mu_h \leq \frac{1}{31K_F^2}$ . ◀

The (rather technical) details on how to choose  $f$  and  $k$  (and consequently the actual values of  $\mu_h, \delta_w$ , and  $\mu_w$ ) will be discussed in the full version of this paper. We next summarize the constraints that arise from the analysis:

- $\mu_w = \frac{\varepsilon\mu_h}{12}$  and  $\delta_w = \frac{\varepsilon\delta_h}{12}$  (Lemma 1),
- $\gamma = \frac{\varepsilon\delta_h}{2}$  (Lemma 2),
- $6\varepsilon^k \leq \frac{\gamma}{6}$  (Lemma 3)
- $\mu_h \leq \frac{\varepsilon\delta_w}{K_B}$  (Lemma 4),
- $\mu_w \leq \gamma \frac{\delta_h}{6K_B(1+\varepsilon)}$  (Lemma 5),
- $\mu_w \leq \frac{\gamma}{3K_F}$  (Lemma 7),
- $\mu_h \leq \frac{\varepsilon}{K_F}$  (Lemma 7),
- $\mu_h \leq \frac{1}{31K_F^2}$  (Lemma 8)

It is not difficult to see that all the constraints are satisfied by choosing  $f(x) = (\varepsilon x)^{C/(\varepsilon x)}$  for a large enough constant  $C$  and  $k = \lceil \log_\varepsilon \left( \frac{\gamma}{36} \right) \rceil$ . Finally we achieve the claimed result.

► **Theorem 9.** *There is a PPT  $(\frac{4}{3} + \varepsilon)$ -approximation algorithm for strip packing.*

## 5 Extension to the case with rotations

In this section, we briefly explain the changes needed in the above algorithm for the case with rotations.

We first observe that, by considering the rotation of rectangles as in the optimum solution, Lemma 7 still applies (for a proper choice of the parameters, that can be guessed). Therefore we can define a multiple knapsack instance, where knapsack sizes are defined as before. Some extra care is needed to define the size  $b(i, j)$  of rectangle  $R_i$  into a container  $C_j$  of size  $w_{C_j} \times h_{C_j}$ . Assume  $C_j$  is horizontal, the other case being symmetric. If rectangle  $R_i$  fits in  $C_j$  both rotated and non-rotated, then we set  $b(i, j) = \min\{w_i, h_i\}$  (this dominates the size occupied in the knapsack by the optimal rotation of  $R_i$ ). If  $R_i$  fits in  $C_j$  only non-rotated (resp., rotated), we set  $b(i, j) = h_i$  (resp.,  $b(i, j) = w_i$ ). Otherwise we set  $b(i, j) = +\infty$ .

There is a final difficulty that we need to address: we can not say a priori whether a rectangle is small (and therefore should be packed in the final stage). To circumvent this difficulty, we define one extra knapsack  $k'$  whose size is the total area in  $\mathcal{B}_{OPT'}$  not occupied by the containers. The size  $b(i, k')$  of  $R_i$  in this knapsack is the area  $a(R_i) = w_i \cdot h_i$  of  $R_i$  provided that  $R_i$  or its rotation by  $90^\circ$  is small w.r.t. the current choice of the parameters  $(\delta_h, \mu_h, \delta_w, \mu_w)$ . Otherwise  $b(i, k') = +\infty$ .

By construction, the above multiple knapsack instance admits a feasible solution that packs all the rectangles. This immediately implies a packing of all the rectangles, excluding the (small) ones in the extra knapsack. Those rectangles can be packed using NFDH as in the proof of Lemma 8 (here however we must choose a rotation such that the considered rectangle is small). Altogether we achieve:

► **Theorem 10.** *There is a PPT  $(\frac{4}{3} + \varepsilon)$ -approximation algorithm for strip packing with rotations.*

## 6 Conclusions

In this paper we obtained a PPT  $4/3 + \varepsilon$  approximation for strip packing (with and without rotations). Our approach refines and, in some sense, pushes to its limit the basic approach in previous work by Nadiradze and Wiese [31]. Indeed, the rearrangement of rectangles inside a box crucially exploits the fact that there are at most 2 tall rectangles packed on top of each other in the optimal packing, hence requiring  $\alpha \geq 1/3$ . We believe that any further improvement requires substantially new algorithmic ideas.

A PPT approximation scheme for strip packing is not excluded by the current inapproximability results (essentially, only strong NP-hardness). Note that, like bin packing, strip packing admits an asymptotic polynomial-time approximation scheme (APTAS), and bin packing admits a PPT approximation scheme [22, 15]. It is an interesting open problem to



find a PPT approximation scheme for this problem, or to prove some stronger hardness of approximation result in PPT.

---

## References

- 1 Anna Adamaszek and Andreas Wiese. Approximation schemes for maximum weight independent set of rectangles. In *FOCS*, pages 400–409, 2013.
- 2 Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the two-dimensional geometric knapsack problem. In *SODA*, pages 1491–1505, 2015.
- 3 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. Constant integrality gap LP formulations of unsplittable flow on a path. In *IPCO*, pages 25–36, 2013.
- 4 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A mazing  $2+\varepsilon$  approximation for unsplittable flow on a path. In *SODA*, pages 26–41, 2014.
- 5 Brenda S. Baker, Donna J. Brown, and Howard P. Katseff. A  $5/4$  algorithm for two-dimensional packing. *Journal of algorithms*, 2(4):348–368, 1981.
- 6 Brenda S. Baker, Edward G. Coffman Jr., and Ronald L. Rivest. Orthogonal packings in two dimensions. *SIAM J. Comput.*, 9(4):846–855, 1980.
- 7 Nikhil Bansal, Amit Chakrabarti, Amir Epstein, and Baruch Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *STOC*, pages 721–729, 2006.
- 8 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *SODA*, pages 13–25, 2014.
- 9 Paul S. Bonsma, Jens Schulz, and Andreas Wiese. A constant-factor approximation algorithm for unsplittable flow on paths. *SIAM J. Comput.*, 43(2):767–799, 2014.
- 10 Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In *SODA*, pages 892–901, 2009.
- 11 Timothy M. Chan and Sarel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.
- 12 Edward G. Coffman Jr. and John L. Bruno. *Computer and job-shop scheduling theory*. John Wiley & Sons, 1976.
- 13 Edward G. Coffman Jr., 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, 2013.
- 14 Edward G. Coffman Jr., Michael R. Garey, David S. Johnson, and Robert E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.
- 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.
- 16 Leah Epstein and Rob van Stee. This side up! *ACM Transactions on Algorithms (TALG)*, 2(2):228–243, 2006.
- 17 Michael R. Garey and David S. Johnson. “Strong” NP-completeness results: Motivation, examples, and implications. *JACM*, 25(3):499–508, 1978.
- 18 Igal Golan. Performance bounds for orthogonal oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 10(3):571–582, 1981.
- 19 Fabrizio Grandoni, Salvatore Ingala, and Sumedha Uniyal. Improved approximation algorithms for unsplittable flow on a path with time windows. In *WAOA*, pages 13–24, 2015.
- 20 Rolf Harren, Klaus Jansen, Lars Prädell, and Rob van Stee. A  $(5/3+\varepsilon)$ -approximation for strip packing. *Computational Geometry*, 47(2):248–267, 2014.
- 21 Rolf Harren and Rob van Stee. Improved absolute approximation ratios for two-dimensional packing problems. In *APPROX-RANDOM*, pages 177–189. Springer, 2009.
- 22 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013.

- 23 Klaus Jansen and Lars Prädél. A new asymptotic approximation algorithm for 3-dimensional strip packing. In *SOFSEM*, pages 327–338, 2014.
- 24 Klaus Jansen and Roberto Solis-Oba. Rectangle packing with one-dimensional resource augmentation. *Discrete Optimization*, 6(3):310–323, 2009.
- 25 Klaus Jansen and Rob van Stee. On strip packing with rotations. In *STOC*, pages 755–761, 2005.
- 26 Klaus Jansen and Guochuan Zhang. Maximizing the total profit of rectangles packed into a rectangle. *Algorithmica*, 47(3):323–342, 2007.
- 27 Mohammad M. Karbasioun, Gennady Shaikhov, Evangelos Kranakis, and Ioannis Lambadaris. Power strip packing of malleable demands in smart grid. In *IEEE International Conference on Communications*, pages 4261–4265, 2013.
- 28 Claire Kenyon and Eric Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Math. Oper. Res.*, 25(4):645–656, 2000.
- 29 Arindam Khan. *Approximation Algorithms for Multidimensional Bin Packing*. PhD thesis, Georgia Institute of Technology, USA, 2015.
- 30 Flavio Keidi Miyazawa and Yoshiko Wakabayashi. Packing problems with orthogonal rotations. In *LATIN*, pages 359–368. Springer, 2004.
- 31 Giorgi Nadiradze and Andreas Wiese. On approximating strip packing better than  $3/2$ . In *SODA*, pages 1491–1510, 2016.
- 32 Anshu Ranjan, Pramod Khargonekar, and Sartaj Sahni. Offline first fit scheduling in smart grids. In *IEEE SCC*, pages 758–763, 2015.
- 33 Ingo Schiermeyer. Reverse-fit: A 2-optimal algorithm for packing rectangles. In *Algorithms—ESA ’94*, pages 290–299. Springer, 1994.
- 34 Daniel Sleator. A 2.5 times optimal algorithm for packing in two dimensions. *Information Processing Letters*, 10(1):37–40, 1980.
- 35 A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.
- 36 Shaojie Tang, Qiuyuan Huang, Xiang-Yang Li, and Dapeng Wu. Smoothing the energy consumption: Peak demand reduction in smart grid. In *INFOCOM*, pages 1133–1141. IEEE, 2013.



# Embedding Approximately Low-Dimensional $\ell_2^2$ Metrics into $\ell_1$

Amit Deshpande<sup>1</sup>, Prahladh Harsha<sup>2</sup>, and Rakesh Venkat<sup>3</sup>

1 Microsoft Research, Bangalore, India  
amitdesh@microsoft.com

2 Tata Institute of Fundamental Research, Mumbai, India  
prahladh@tifr.res.in

3 Tata Institute of Fundamental Research, Mumbai, India  
rakesh09@gmail.com

---

## Abstract

Goemans showed that any  $n$  points  $x_1, \dots, x_n$  in  $d$ -dimensions satisfying  $\ell_2^2$  triangle inequalities can be embedded into  $\ell_1$ , with worst-case distortion at most  $\sqrt{d}$ . We consider an extension of this theorem to the case when the points are *approximately* low-dimensional as opposed to exactly low-dimensional, and prove the following analogous theorem, albeit with *average distortion* guarantees: There exists an  $\ell_2^2$ -to- $\ell_1$  embedding with average distortion at most the *stable rank*,  $\text{sr}(M)$ , of the matrix  $M$  consisting of columns  $\{x_i - x_j\}_{i < j}$ . Average distortion embedding suffices for applications such as the SPARSEST CUT problem. Our embedding gives an approximation algorithm for the SPARSEST CUT problem on low threshold-rank graphs, where earlier work was inspired by Lasserre SDP hierarchy, and improves on a previous result of the first and third author [Deshpande and Venkat, In *Proc. 17th APPROX*, 2014]. Our ideas give a new perspective on  $\ell_2^2$  metric, an alternate proof of Goemans' theorem, and a simpler proof for *average distortion*  $\sqrt{d}$ .

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Metric Embeddings, Sparsest Cut, Negative type metrics, Approximation Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.10

## 1 Introduction

A finite metric space consists of a pair  $(\mathcal{X}, d)$ , where  $\mathcal{X}$  is a finite set of points, and  $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$  is a distance function on pairs of points in  $\mathcal{X}$ . Finite metric spaces arise naturally in combinatorial optimization (e.g., the  $\ell_1$  space in cut problems), and in practice (e.g., edit-distance between strings over some alphabet  $\Sigma$ ). Since the input space may not be amenable to efficient optimization, or may not admit efficient algorithms, one looks for *embeddings* from these input spaces to easier spaces, while minimizing the *distortion* incurred. Given its importance, various aspects of such embeddings have been investigated such as dimension, distortion, efficient algorithms, and hardness results (refer to surveys [10, 16, 14] and references therein). In this paper, we provide better distortion guarantees for embedding *approximately* low-dimensional points in the  $\ell_2^2$ -metric into  $\ell_1$ , and give applications to the SPARSEST CUT problem.

In the SPARSEST CUT problem, we are given graphs  $C, D$  on the same vertex set  $V$ , with  $|V| = n$ , called the *cost* and *demand* graphs, respectively. They are specified by non-negative



© Amit Deshpande, Prahladh Harsha, and Rakesh Venkat;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 10; pp. 10:1–10:13



Leibniz International Proceedings in Informatics

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

## 10:2 Embedding Approximately Low-Dimensional $\ell_2^2$ Metrics Into $\ell_1$

edge weights  $c_{ij}, d_{ij} \geq 0$ , for  $i < j \in [n]$  and the (*non-uniform*) *sparsest cut* problem, henceforth referred to as SPARSEST CUT, asks for a subset  $S \subseteq V$  that minimizes

$$\Phi(S) := \frac{\sum_{i < j} c_{ij} |\mathbb{I}_S(i) - \mathbb{I}_S(j)|}{\sum_{i < j} d_{ij} |\mathbb{I}_S(i) - \mathbb{I}_S(j)|},$$

where  $\mathbb{I}_S(i)$  is the indicator function giving 1, if  $i \in S$ , and 0, otherwise. We denote the optimum by  $\Phi^* := \min_{S \subseteq V} \Phi(S)$ . When the demand graph is a complete graph on  $n$  vertices with uniform edge weights, the problem is then commonly referred to as the UNIFORM SPARSEST CUT problem.

The best known (unconditional) approximation guarantee for the UNIFORM SPARSEST CUT problem is  $O(\sqrt{\log n})$ , due to Arora, Rao and Vazirani [3] (henceforth referred to as the ARV algorithm). Building on techniques in this work, Arora, Lee and Naor [2] give a  $O(\sqrt{\log n} \log \log n)$  algorithm for non-uniform SPARSEST CUT. These results come from a semi-definite programming (SDP) relaxation to produce solutions in the  $\ell_2$ -squared metric space, i.e., a set of vectors  $\{x_i\}_{i \in V}$  in some high dimensional space that satisfy triangle inequality constraints on the squared distances in the following sense.

$$\|x_i - x_j\|_2^2 + \|x_j - x_k\|_2^2 \geq \|x_i - x_k\|_2^2 \quad \forall i, j, k \in [n].$$

Since the  $\ell_1$  metric lies in the non-negative cone of cut (semi-)metrics, ARV [3] and Arora-Lee-Naor[2] round their solutions via low-distortion embeddings of the above  $\ell_2^2$  solution into  $\ell_1$  metric. Embeddings with low *average-distortion* suffice for applications to the SPARSEST CUT problem.

Any  $n$  points satisfying  $\ell_2^2$  triangle inequalities make only acute angles among themselves, and therefore must lie in  $\Omega(\log n)$  dimensions (Chapter 15, [1]). However, for low *threshold-rank* graphs, or more generally, when the  $r$ -th smallest generalized eigenvalue of the cost and demand graphs satisfies  $\lambda_r(C, D) \gg \Phi_{SDP}$ , the above SDP solution is known to be *approximately* low-dimensional, that is, the span of its top  $r$  eigenvectors contains nearly all of its total eigenmass (implicit in [9]). Moreover, it can be embedded into  $\ell_1$  using solutions of higher-levels of the Lasserre SDP hierarchy to obtain a PTAS-like approximation guarantee [9]. This motivates the quest for finding more efficient embeddings of low-dimensional or *approximately* low-dimensional  $\ell_2^2$  metrics into  $\ell_1$ .

Goemans (unpublished, appears in [15]) showed that if the points satisfying  $\ell_2^2$  triangle inequalities lie in  $d$  dimensions, then they can be embedded into  $\ell_2$  (and hence into  $\ell_1$ , since there is an isometry from  $\ell_2 \hookrightarrow \ell_1$  [16]) with  $\sqrt{d}$  distortion.

► **Theorem 1.1** (Goemans [15, Appendix B]). *Let  $x_1, x_2, \dots, x_n \in \mathbb{R}^d$  be  $n$  points satisfying  $\ell_2^2$  triangle inequalities. Then there exists an  $\ell_2^2 \hookrightarrow \ell_2$  embedding  $x_i \mapsto f(x_i)$  with distortion  $\sqrt{d}$ , that is,*

$$\frac{1}{\sqrt{d}} \|x_i - x_j\|_2^2 \leq \|f(x_i) - f(x_j)\|_2^2 \leq \|x_i - x_j\|_2^2, \quad \forall i, j \in V.$$

### Comparison of Goemans and ARV

Since  $n$  points satisfying  $\ell_2^2$  triangle inequalities must lie in  $d = \Omega(\log n)$  dimensions (Chapter 15, [1]), the ARV algorithm [3] implies an  $\ell_2^2 \hookrightarrow \ell_1$  embedding with *average* distortion  $O(\sqrt{d})$ , and Arora-Lee-Naor [2] improve it to  $\tilde{O}(\sqrt{d})$  worst-case distortion. In the other direction, is it possible to extend Theorem 1.1 to give ARV-like guarantees? Here are two immediate ideas that come to mind.

- Combine Theorem 1.1 with a dimension reduction to  $O(\log n)$  dimensions for  $\ell_2^2$  metrics, similar to the Johnson-Lindenstrauss lemma for  $\ell_2$ . Such a dimension reduction for  $\ell_2^2$  that approximately preserves all pairwise  $\ell_2^2$  distances is ruled out by Magen and Moharrami [15], although their proof does not rule out dimension reduction for *average* distortion.
- Extend Theorem 1.1 to work with approximate  $\ell_2^2$  triangle inequalities, and then combine it with the Johnson-Lindenstrauss lemma. The Johnson-Lindenstrauss lemma, when applied to points satisfying  $\ell_2^2$  triangle inequalities, preserves their  $\ell_2^2$  triangle inequalities only approximately. That is, the points after the Johnson-Lindenstrauss random projection satisfy

$$\|x_i - x_j\|_2^2 + \|x_j - x_k\|_2^2 \geq (1 - O(\epsilon)) \|x_i - x_k\|_2^2 \quad \forall i, j, k \in [n].$$

We note that a generalization of Theorem 1.1 that accommodates approximate  $\ell_2^2$  triangle inequalities (in the additive sense not multiplicative as above) does hold, but its only proof (due to Trevisan [personal communication]) that we are aware of uses the technical core of the analysis of the ARV algorithm.

Here we seek a robust generalization of Goemans' theorem that avoids the above caveats. Our version of Goemans' theorem uses *average* distortion instead of worst-case. It is robust in the sense that it works with *approximate* dimension instead of the actual dimension. Such a robust version opens up another possible approach to the general SPARSEST CUT problem: reduce the *approximate* dimension while preserving the pairwise distances on *average*, and then apply the robust version of Goemans' theorem. Moreover, our definition of the approximate dimension is spectral, and our results can be easily compared to those of Guruswami-Sinop [9] on Lasserre SDP hierarchies and Kwok et al. [13] on higher order Cheeger inequalities (see Sections 1.1 and 1.2 for comparisons).

## 1.1 Our Results

We consider a robust version of Goemans' theorem, when the points  $x_1, x_2, \dots, x_n$  are only *approximately* low-dimensional. We quantify this *approximate* dimension by the *stable rank* of the difference matrix  $M \in \mathbb{R}^{d \times \binom{n}{2}}$  having columns  $\{x_i - x_j\}_{i < j}$ . Stable rank of the difference matrix is a natural choice because (a) stable rank is a continuous proxy for rank or dimension arising naturally in many applications [5, 17], (b) the difference matrix  $M$  is invariant under any shift of origin, and (c) the difference matrix of the SDP solution for the SPARSEST CUT problem on *low threshold-rank* graphs indeed has low stable rank (implicit in [9]).

► **Definition 1.2** (Stable Rank). Given  $x_1, \dots, x_n \in \mathbb{R}^d$ , let  $M \in \mathbb{R}^{d \times \binom{n}{2}}$  be the matrix with columns  $\{x_i - x_j\}_{i < j}$ . The stable rank of the points is defined as the stable rank of  $M$ , given by  $\text{sr}(M) := \|M\|_F^2 / \|M\|_2^2$ , where  $\|M\|_F$  and  $\|M\|_2$  are the Frobenius and spectral norm of  $M$  respectively.

Note that  $\text{sr}(M) \leq \text{rank}(M) \leq d$ , when the points  $x_1, x_2, \dots, x_n \in \mathbb{R}^d$ . Our robust version of Goemans' theorem is as follows.

► **Theorem 1.3** (Embedding almost low-dimensional vectors). *Let  $x_1, x_2, \dots, x_n \in \mathbb{R}^d$  be  $n$  points satisfying  $\ell_2^2$  triangle inequalities. Then there exists an  $\ell_2^2 \hookrightarrow \ell_2$  embedding  $x_i \mapsto h(x_i)$  with average distortion bounded by the stable rank of  $M$ , that is,*

$$\|h(x_i) - h(x_j)\|_2 \leq \|x_i - x_j\|_2, \quad \forall i, j \in V,$$

and

$$\frac{1}{\text{sr}(M)} \sum_{i < j} \|x_i - x_j\|_2^2 \leq \sum_{i < j} \|h(x_i) - h(x_j)\|_2.$$

We note that the above theorem is not a strict generalization of Goemans' theorem to the approximate dimension case. To obtain a truly robust version of Goemans' theorem quantitatively, one might ask if the dependence on  $\text{sr}(M)$  in the above theorem can be improved from  $\text{sr}(M)$  to  $\sqrt{\text{sr}(M)}$ .

Our proof technique gives a new perspective on  $\ell_2^2$  metric, an alternate proof of Goemans' theorem, and a simpler *algorithmic* proof for average distortion  $\sqrt{d}$  based on a squared-length distribution (see Section 4, and the remark following the proof of Theorem 4.1). Also, the result can be quantitatively compared to guarantees given by higher-order Cheeger inequalities [13]; we discuss this in more detail at the end of this section. While most known embeddings from  $\ell_2^2$  to  $\ell_1$  are Frechet embeddings, our embedding is projective (similar in spirit to [9, 7]).

Theorem 1.3 immediately implies an  $\text{sr}(M)$ -approximation to the UNIFORM SPARSEST CUT problem. In fact, with a slight modification, we obtain a similar result for the general SPARSEST CUT problem (see theorem below).

► **Theorem 1.4.** *There is an  $r/\delta$ -approximation algorithm for SPARSEST CUT instances  $C, D$  satisfying  $\lambda_r(C, D) \geq \Phi_{SDP}/(1 - \delta)$ , where  $\lambda_r(C, D)$  is the  $r$ -th smallest generalized eigenvalue (see Section 2) of the Laplacians of the cost and demand graphs.*

The precondition on  $\lambda_r(C, D)$  is the same as in previous works [9, 7], and we improve the  $O(r/\delta^2)$ -approximation of [7] by a factor of  $1/\delta$ . Our proof follows from the robust version of Goemans' embedding into  $\ell_2$  whereas these previous works gave embeddings directly into  $\ell_1$  by either using higher levels of Lasserre explicitly [9] or using only the basic SDP solution but inspired by the properties of Lasserre vectors [7]. We can infer the following corollary almost immediately:

► **Corollary 1.5.** *For any  $\epsilon > 0$  and a  $d$ -regular cost graph  $C$  satisfying  $\lambda_r(C) \geq \epsilon d$ , there is a  $\max\{O(r), \frac{1}{\sqrt{\epsilon}}\}$  approximation to UNIFORM SPARSEST CUT.*

**Proof.** The implicit demand graph here is  $K_n$ , the complete graph on  $n$  vertices, and thus the generalized eigenvalues are  $\lambda_r(C, K_n) = \lambda_r/n$ . Consider two cases: If  $\Phi_{SDP} \leq \epsilon d/100n$  then  $\lambda_r/n \geq 100\Phi_{SDP}$  yielding an  $O(r)$  approximation by Theorem 1.4. Otherwise, if  $\Phi_{SDP} \geq \epsilon d/100n$ , then running a basic Cheeger rounding and analysis on (one co-ordinate of) the SDP solution would itself give a cut of sparsity  $O(d\sqrt{\epsilon}/n) \leq \Phi_{SDP}/\sqrt{\epsilon}$ . Thus, using the minimum of these gives a cut within a factor  $\max\{O(r), 1/\sqrt{\epsilon}\}$  of the optimum. ◀

## 1.2 Related work

We recall that the best known upper bound for the worst-case distortion of embedding  $\ell_2^2 \hookrightarrow \ell_1$  is  $O(\sqrt{\log n} \cdot \log \log n)$  [3, 2], while the best known lower bound is  $(\log n)^{\Omega(1)}$  for worst-case distortion [6], and  $\exp(\Omega(\sqrt{\log \log n}))$  for average distortion [11]. Guarantees to SPARSEST CUT on *low threshold-rank* graphs were obtained using higher levels of the Lasserre hierarchy for SDPs [4, 9]. In contrast, a previous work of the first and third author [7] showed weaker guarantees, but using just the basic SDP relaxation. Oveis Gharan and Trevisan [8] also give a rounding algorithm for the basic SDP relaxation on low-threshold rank graphs, but require a stricter pre-condition on the eigenvalues ( $\lambda_r \gg \log^{2.5} r \cdot \Phi(G)$ ), and leverage it

to give a stronger  $O(\sqrt{\log r})$ -approximation guarantee. Their improvement comes from a new structure theorem on the SDP solutions of low threshold-rank graphs being clustered, and using the techniques in ARV for analysis.

Kwok et al. [13] showed that a better analysis of Cheeger's inequality gives a  $O(r \cdot \sqrt{d/\lambda_r})$  approximation to UNIFORM SPARSEST CUT on  $d$ -regular graphs. In particular, when  $\lambda_r(G) \geq \epsilon d$ , this gives a  $O(r/\sqrt{\epsilon})$  approximation for the UNIFORM SPARSEST CUT problem. Note that Corollary 1.5 gives a slightly better approximation in this setting.

Further, while the Kwok et al. result is tight with respect to the spectral solution, our approach allows for an improvement in terms of the dependence on  $r$  to  $\sqrt{r}$ , since it uses the SDP relaxation rather than a spectral solution.

## 2 Preliminaries and Notation

### Sets, Matrices, Vectors

We use  $[n] = \{1, \dots, n\}$ . For a matrix  $X \in \mathbb{R}^{d \times d}$ , we say  $X \succeq 0$  or  $X$  is positive-semidefinite (psd) if  $y^T X y \geq 0$  for all  $y \in \mathbb{R}^d$ . The Gram-matrix of a matrix  $M \in \mathbb{R}^{d_1 \times d_2}$  is the matrix  $M^T M$ , which is psd.

Every matrix  $M$  has a singular value decomposition  $M = \sum_i \sigma_i u_i v_i^T = U D V^T$ . Here, the matrices  $U, V$  are Unitary, and  $D$  is the diagonal matrix of the singular values  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n$ , in non-increasing order. When not clear from context, we denote the singular values of  $M$  by  $\sigma_i(M)$ .

The *Frobenius* norm of  $M$  is given by  $\|M\|_F := \sqrt{\sum_i \sigma_i^2(M)} = \sqrt{\sum_{i \in [d_1], j \in [d_2]} M(i, j)^2}$ . In our analysis, we will sometimes view a matrix  $M$  as a collection of its columns viewed as vectors;  $M = (m_j)_{j \in [d_2]}$ . In this case,  $\|M\|_F^2 = \sum_j \|m_j\|_2^2$ . The spectral norm of  $M$  is  $\|M\|_2 := \sigma_1$ .

### Generalized Eigenvalues

Given two symmetric matrices  $X, Y \in \mathbb{R}^d \times d$  with  $Y \succeq 0$ , and for  $i \leq \text{rank}(Y)$ , we define their  $i$ -th smallest generalized eigenvalue as the following:

$$\lambda_i = \max_{\text{rank}(Z) \leq i-1} \min_{w \perp Z; w \neq 0} \frac{w^T X w}{w^T Y w}$$

### Rank and Stable Rank

The rank of the matrix  $M$  (denoted by  $\text{rank}(M)$ ) is the number of non-zero singular values. Recall that the *stable rank* of the matrix  $M$ ,  $\text{sr}(M) = \frac{\|M\|_F^2}{\sigma_1(M)^2}$ . Note that  $\text{sr}(M) = \sum_{i=1}^{\text{rank}(M)} \sigma_i^2(M) / \sigma_1^2(M) \leq \text{rank}(M)$ .

### Metric spaces and embeddings

For our purposes, a (semi-)metric space  $(\mathcal{X}, d)$  consists of a finite set of points  $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$  and a distance function  $d : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_{\geq 0}$  satisfying the following three conditions:

1.  $d(x, x) = 0, \forall x \in \mathcal{X}$ .
2.  $d(x, y) = d(y, x)$ .
3. (Triangle inequality)  $d(x, y) + d(y, z) \geq d(x, z)$ .

## 10:6 Embedding Approximately Low-Dimensional $\ell_2^2$ Metrics Into $\ell_1$

An *embedding* from a metric space  $(\mathcal{X}, d)$  to a metric space  $(\mathcal{Y}, d')$  is a mapping  $f : \mathcal{X} \rightarrow \mathcal{Y}$ . The embedding is called a *contraction*, if

$$d'(f(x_i), f(x_j)) \leq d(x_i, x_j), \quad \forall x_i, x_j \in \mathcal{X}.$$

For convenience, we will only deal with contractive mappings in this paper. A contractive mapping is said to have (worst-case) distortion  $\Delta$ , if

$$\sup_{i,j} \frac{d(x_i, x_j)}{d'(f(x_i), f(x_j))} \leq \Delta.$$

It is said to have *average* distortion  $\beta$ , if

$$\frac{\sum_{i<j} d(x_i, x_j)}{\sum_{i<j} d(f(x_i), f(x_j))} \leq \beta.$$

Note that a mapping with worst-case distortion  $\Delta$  also has average distortion  $\Delta$ , but not necessarily vice-versa.

### The $\ell_2^2$ space

A set of points  $\{x_1, x_2, \dots, x_n\} \in \mathbb{R}^d$  are said to satisfy  $\ell_2^2$  triangle inequality constraints, or said to be in  $\ell_2^2$  space, if it holds that

$$\|x_i - x_j\|_2^2 + \|x_j - x_k\|_2^2 \geq \|x_i - x_k\|_2^2 \quad \forall i, j, k \in [n].$$

These satisfy the triangle inequalities on the *squares* of their  $\ell_2$  distances. The corresponding metric space is  $(\mathcal{X}, d)$ , where  $d(i, j) := \|x_i - x_j\|_2^2$ .

### Graphs and Laplacians

All graphs will be defined on a vertex set  $V$  of size  $n$ . The vertices will usually be referred to by indices  $i, j, k, l \in [n]$ . Given a graph with weights on pairs  $W : \binom{V}{2} \mapsto \mathbb{R}^+$ , the graph Laplacian matrix is defined as:

$$L_W(i, j) := \begin{cases} -W(i, j) & \text{if } i \neq j \\ \sum_k W(i, k) & \text{if } i = j. \end{cases}$$

Note that  $L_W \succeq 0$ . We will denote the eigenvalues of (the Laplacian of)  $G$  by  $0 = \lambda_1 \leq \lambda_2 \dots \leq \lambda_n$ , in *increasing* order.

### Sparsest Cut SDP

The SDP we use for SPARSEST CUT on the vertex set  $V$  with costs and demands  $c_{ij}, d_{kl} \geq 0$  and corresponding cost and demand graphs  $C : \binom{V}{2} \mapsto \mathbb{R}^+$  and  $D : \binom{V}{2} \mapsto \mathbb{R}^+$ , is effectively the following:

$$\begin{aligned} \text{SDP: } \Phi_{SDP} &:= \min \sum_{i<j} c_{ij} \|x_i - x_j\|_2^2 \\ &\text{subject to } \begin{cases} \|x_i - x_j\|_2^2 + \|x_j - x_k\|_2^2 \geq \|x_i - x_k\|_2^2 & \forall i, j, k \in [n]. \\ \sum_{k<l} d_{kl} \|x_k - x_l\|_2^2 = 1. \end{cases} \end{aligned}$$

Note that the solution to the above SDP is in  $\ell_2^2$  space.

### $\ell_1$ embeddings and cuts

Since  $\ell_1$  metrics are exactly the cone of cut-metrics, it follows from the previous discussion on embeddings, that producing an embedding of the SDP solutions  $\mathcal{X} = \{x_1, \dots, x_n\}$  in  $\ell_2^2$  space to  $\ell_1$  space with distortion  $\alpha$  would give an  $\alpha$ -approximation to SPARSEST CUT. Producing one with *average* distortion  $\alpha$  would give an  $\alpha$ -approximation to UNIFORM SPARSEST CUT. Furthermore, since  $\ell_2$  embeds isometrically (distortion 1) into  $\ell_1$ , it suffices to show embeddings into  $\ell_2$  for the above purposes.

### Key Lemma

The following lemma about  $\ell_2^2$  spaces was observed by Deshpande and Venkat [7]. We will reuse this in the rest of the paper.

► **Lemma 2.1** ([7, Proposition 1.3]). *Let  $x_1, x_2, \dots, x_n$  be  $n$  points satisfying  $\ell_2^2$  triangle inequalities. Then*

$$\left\langle x_i - x_j, \frac{x_k - x_l}{\|x_k - x_l\|_2} \right\rangle^2 \leq |\langle x_i - x_j, x_k - x_l \rangle| \leq \|x_i - x_j\|_2^2, \quad \forall i, j, k, l \in V.$$

An immediate consequence of this lemma is that we can show that a large class of naturally defined  $\ell_2^2 \hookrightarrow \ell_2$  embeddings are contractions.

► **Lemma 2.2** (Contraction). *Let  $x_1, x_2, \dots, x_n$  be  $n$  points satisfying  $\ell_2^2$  triangle inequalities. For any probability distribution  $\{p_{kl}\}_{k<l}$ , let  $P$  be the symmetric psd matrix defined as  $P := \sum_{k<l} p_{kl} (x_k - x_l)(x_k - x_l)^T$ . Then the  $\ell_2^2 \hookrightarrow \ell_2$  embedding given by  $x_i \mapsto P^{1/2}x_i$  is a contraction, that is,*

$$\left\| P^{1/2}(x_i - x_j) \right\|_2 \leq \|x_i - x_j\|_2, \quad \forall i, j \in V.$$

**Proof.** The following holds for all  $i, j$ :

$$\begin{aligned} \left\| P^{1/2}(x_i - x_j) \right\|_2 &= \left( (x_i - x_j)^T P (x_i - x_j) \right)^{1/2} \\ &= \left( \sum_{k<l} p_{kl} \langle x_i - x_j, x_k - x_l \rangle^2 \right)^{1/2} \\ &\leq \left( \sum_{k<l} p_{kl} \|x_i - x_j\|_2^4 \right)^{1/2} && \text{[By Lemma 2.1]} \\ &= \|x_i - x_j\|_2^2. && \text{[Since } \sum_{k<l} p_{kl} = 1] \end{aligned}$$

◀

## 3 Embedding almost low-dimensional vectors

We now prove the robust version of Goemans' theorem in terms of stable rank. We give two proofs, and show an application to round solutions to SPARSEST CUT on low-threshold-rank graphs. As before, given a set of points  $x_1, \dots, x_n$  in  $\mathbb{R}^d$ , define their difference matrix  $M \in \mathbb{R}^{d \times \binom{n}{2}}$  as the matrix with columns as  $\{x_i - x_j\}_{i<j}$ .

## 10:8 Embedding Approximately Low-Dimensional $\ell_2^2$ Metrics Into $\ell_1$

**Proof of Theorem 1.3.** Let  $u$  and  $v$  be the top left and right singular vector of  $M$ , respectively, and  $\sigma_1 \leq \sigma_2 \leq \dots \leq \sigma_d$  be the singular values of  $M$ . Then  $Mv = \sigma_1 u$ , or in other words,  $\sigma_1 u = \sum_{k < l} v_{kl}(x_k - x_l)$ . Now consider the embedding  $x_i \mapsto h(x_i) = P^{1/2}x_i$ , where the probability distribution  $p_{kl} \propto |v_{kl}|$ , that is

$$P = \sum_{k < l} \frac{|v_{kl}|}{\|v\|_1} (x_k - x_l)(x_k - x_l)^T.$$

This embedding is a contraction by Lemma 2.2. Now let's bound its average distortion.

$$\begin{aligned} \sum_{i < j} \|h(x_i) - h(x_j)\|_2 &= \sum_{i < j} \left\| P^{1/2}(x_i - x_j) \right\|_2 \\ &= \sum_{i < j} \left( (x_i - x_j)^T P (x_i - x_j) \right)^{1/2} \\ &= \sum_{i < j} \left( \sum_{k < l} \frac{|v_{kl}|}{\|v\|_1} \langle x_i - x_j, x_k - x_l \rangle^2 \right)^{1/2} \\ &\geq \sum_{i < j} \sum_{k < l} \frac{|v_{kl}|}{\|v\|_1} |\langle x_i - x_j, x_k - x_l \rangle| \quad [\text{By Jensen's inequality}] \\ &\geq \sum_{i < j} \frac{1}{\|v\|_1} \left| \left\langle x_i - x_j, \sum_{k < l} v_{kl}(x_k - x_l) \right\rangle \right| \quad [\text{By triangle inequality}] \\ &= \frac{1}{\|v\|_1} \sum_{i < j} |\langle x_i - x_j, \sigma_1 u \rangle| \\ &= \frac{1}{\|v\|_1} \sum_{i < j} \sigma_1^2 |v_{ij}| \\ &= \sigma_1^2 = \frac{\|M\|_F^2}{\text{sr}(M)} \\ &= \frac{1}{\text{sr}(M)} \sum_{i < j} \|x_i - x_j\|_2^2. \end{aligned}$$

◀

### 3.1 An alternative proof

We can alternatively get the same guarantee as in Theorem thm:stable-rank, by giving a one-dimensional  $\ell_2$  embedding (and hence also  $\ell_1$  embedding without any extra effort) along the top singular vector of the difference matrix  $M$ . This gives an interesting “spectral” algorithm that uses spectral information about the point set, akin to spectral algorithms in graphs that use the spectrum of the graph Laplacian.

► **Theorem 3.1.** *Let  $x_1, x_2, \dots, x_n \in \mathbb{R}^d$  be  $n$  points satisfying  $\ell_2^2$  triangle inequalities with  $M$  as their difference matrix. Let  $u \in \mathbb{R}^d$  and  $v \in \mathbb{R}^{\binom{n}{2}}$  be its top left and right singular vectors, respectively. Then  $x_i \mapsto \frac{\sigma_1}{\|v\|_1} \langle x_i, u \rangle$  is an  $\ell_2^2 \hookrightarrow \ell_2$  embedding with average distortion bounded by the stable rank of  $M$ .*

**Proof.** We have  $Mv = \sigma_1 u$ , or equivalently,  $\sigma_1 u = \sum_{k < l} v_{kl}(x_k - x_l)$ . Our embedding is a contraction since



$$\begin{aligned}
\frac{\sigma_1}{\|v\|_1} |\langle x_i - x_j, u \rangle| &= \frac{1}{\|v\|_1} \left| \left\langle x_i - x_j, \sum_{k < l} v_{kl} (x_k - x_l) \right\rangle \right| \\
&\leq \frac{1}{\|v\|_1} \sum_{k < l} |v_{kl}| |\langle x_i - x_j, x_k - x_l \rangle| \\
&\leq \frac{1}{\|v\|_1} \sum_{k < l} |v_{kl}| \|x_i - x_j\|_2^2 && \text{[By Lemma 2.1]} \\
&= \|x_i - x_j\|_2^2.
\end{aligned}$$

Now let's bound the average distortion.

$$\begin{aligned}
\sum_{i < j} \frac{\sigma_1}{\|v\|_1} |\langle x_i - x_j, u \rangle| &= \sum_{i < j} \frac{\sigma_1}{\|v\|_1} |\sigma_1 v_{ij}| && \text{[Since } u^T M = \sigma_1 v^T\text{]} \\
&= \sigma_1^2 = \frac{\|M\|_F^2}{\text{sr}(M)} \\
&= \frac{1}{\text{sr}(M)} \sum_{i < j} \|x_i - x_j\|_2^2.
\end{aligned}$$

◀

### 3.2 Application to Sparsest Cut on low-threshold rank graphs

We first state a property of SDP solutions on low threshold-rank graphs, proved by Guruswami and Sinop [9] using the Von-Neumann inequality.

► **Proposition 3.2** (Von-Neumann inequality [9, Theorem 3.3]). *Let  $0 \leq \lambda_1 \leq \dots \leq \lambda_m$  be the generalized eigenvalues of the Laplacian matrices of the cost and demand graphs. Let  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0$  be the singular vectors of the matrix  $M$  with columns  $\{\sqrt{d_{ij}}(x_i - x_j)\}_{i < j}$ . Then*

$$\frac{\sum_{t \geq r+1} \sigma_j^2}{\sum_{t=1}^n \sigma_j^2} \leq \frac{\Phi_{SDP}}{\lambda_{r+1}}.$$

In particular, note that on graphs where  $\lambda_r \geq \Phi_{SDP}/(1 - \delta)$ ,  $\sum_{i \leq r} \sigma_i^2 \geq \delta \sum_i \sigma_i^2$ . This implies that  $\text{sr}(M) = \sum_i \sigma_i^2 / \sigma_1^2 \leq r \cdot \sum_i \sigma_i^2 / \sum_{i \leq r} \sigma_i^2 \leq r/\delta$ .

We can now modify the proof of Theorem 3.1 to prove Theorem 1.4.

**Proof of Theorem 1.4.** Let  $x_1, \dots, x_n$  be the SDP solution on given instance  $C, D$ . We now let  $M$  be the matrix with columns  $\{\sqrt{d_{kl}}(x_k - x_l)\}_{k < l}$ , and  $u, v, \sigma_1$  to be the top left singular vector, top right singular vector, and the maximum singular value respectively of  $M$ . By the preceding remark,  $\text{sr}(M) \leq r/\delta$ . The mapping we use is as follows

$$x_i \mapsto \frac{1}{\sum_{kl} \sqrt{d_{kl}} v_{kl}} \langle x_i, u \rangle.$$

The proofs to show contraction and bound the distortion follow exactly as in the proof of Theorem 3.1. Note that while looking at the distortion, we need to lower bound the quantity  $\sum_{ij} d_{ij} \|g(x_i) - g(x_j)\|_2$ . ▶

As in Deshpande and Venkat [7], the above algorithm is a fixed polynomial time algorithm and does not grow with the threshold rank unlike the algorithm of Guruswami and Sinop [9] where they use  $r$ -levels of the Lasserre SDP hierarchy to secure the guarantee. Furthermore, the above analysis improves the guarantee of Deshpande and Venkat [7] by a factor of  $O(1/\delta)$ .

## 4 Embedding low-dimensional vectors à la Goemans

In this section, we first view the proof of Goemans' theorem in the framework of Lemma 2.2 by giving a probability distribution using the minimum volume enclosing ellipsoid of the difference vectors  $(x_i - x_j)$ 's. We then give a simpler proof, albeit for the *average* distortion case, based on a probability distribution arising from a squared-length distribution. Via a well-known duality statement, this technique recovers Goemans' theorem for *worst-case* distortion for embeddings into  $\ell_1$ , although non-constructively.

### 4.1 An alternate proof of Goemans' theorem

Here is an adaptation of the proof from [15] re-stated in our framework. The following proof is arguably simpler and more straightforward as it works with the difference vectors instead of the original vectors and their negations.

► **Theorem 1.1** (restated – Goemans [15, Appendix B]). Let  $x_1, x_2, \dots, x_n \in \mathbb{R}^d$  be  $n$  points satisfying  $\ell_2^2$  triangle inequalities. Then there exists an  $\ell_2^2 \hookrightarrow \ell_2$  embedding  $x_i \mapsto f(x_i)$  with distortion  $\sqrt{d}$ , that is,

$$\frac{1}{\sqrt{d}} \|x_i - x_j\|_2^2 \leq \|f(x_i) - f(x_j)\|_2 \leq \|x_i - x_j\|_2^2, \quad \forall i, j \in V.$$

**Proof.** Consider all the difference vectors  $(x_i - x_j)$ 's, and let their minimum volume enclosing ellipsoid be given by  $E := \{x : x^T Q x \leq 1\}$ , for some psd matrix  $Q \in \mathbb{R}^{d \times d}$ . By John's theorem (or Lagrangian duality for the corresponding convex program), we have  $Q^{-1} = \sum_{k < l} \alpha_{kl} (x_k - x_l)(x_k - x_l)^T$ , with all  $\alpha_{kl} \geq 0$ . Moreover,  $\alpha_{kl} \neq 0$  iff  $(x_k - x_l)^T Q (x_k - x_l) = 1$ . Notice that  $d = \text{Tr}(\mathbb{I}_d) = \text{Tr}(Q^{1/2} Q^{-1} Q^{1/2}) = \sum_{k < l} \alpha_{kl}$ . We define the embedding as

$$f(x_i) := \frac{1}{\sqrt{d}} Q^{-1/2} x_i.$$

This embedding is a contraction by Lemma 2.2. We now bound the distortion:

$$\begin{aligned} \|f(x_i) - f(x_j)\|_2 &= \frac{1}{\sqrt{d}} \left\| Q^{-1/2} (x_i - x_j) \right\|_2 \\ &\geq \frac{1}{\sqrt{d}} \frac{\|x_i - x_j\|_2^2}{\|Q^{1/2} (x_i - x_j)\|_2} && \text{[By Cauchy-Schwarz inequality]} \\ &\geq \frac{1}{\sqrt{d}} \|x_i - x_j\|_2^2. && \text{[Since } (x_i - x_j)^T Q (x_i - x_j) \leq 1, \text{ for all } i, j] \end{aligned}$$

◀

### 4.2 A simpler proof for average distortion embedding

We now give an average distortion version of Goemans' theorem using a simple squared-length distribution on the difference vectors  $(x_i - x_j)$ 's in the Lemma 2.2. Interestingly, this can be modified to weighted averages and gives yet another proof of Goemans' worst-case distortion result, although non-constructively.

► **Theorem 4.1.** Let  $x_1, x_2, \dots, x_n \in \mathbb{R}^d$  be points satisfying  $\ell_2^2$  triangle inequalities. Then there exists an  $\ell_2^2$ -to- $\ell_2$  embedding  $x_i \mapsto g(x_i)$  with average distortion  $\sqrt{d}$ , that is,

$$\|g(x_i) - g(x_j)\|_2 \leq \|x_i - x_j\|_2^2, \quad \text{for all } i, j,$$

and

$$\frac{1}{\sqrt{d}} \sum_{i < j} \|x_i - x_j\|_2^2 \leq \sum_{i < j} \|g(x_i) - g(x_j)\|_2$$

**Proof.** Let  $\{p_{kl}\}_{k<l}$  define a probability distribution with  $p_{kl} \propto \|x_k - x_l\|_2^2$ . Given this distribution, let  $P$  be the symmetric psd matrix defined as  $P := \sum_{k<l} p_{kl} (x_k - x_l)(x_k - x_l)^T \in \mathbb{R}^{d \times d}$ . Consider the embedding that maps  $x_i$  to  $g(x_i) := P^{1/2}x_i$ . The embedding is a contraction by Lemma 2.2.

Now let's bound the average distortion. First, note that:

$$\|g(x_i) - g(x_j)\|_2 = \left\| P^{1/2}(x_i - x_j) \right\|_2 \geq \frac{\|x_i - x_j\|_2^2}{\|P^{-1/2}(x_i - x_j)\|_2},$$

where the inequality follows from the Cauchy-Schwarz inequality.

Summing over all pairs  $i, j$  and using the definition of  $p_{ij}$  we have

$$\begin{aligned} \sum_{i<j} \|g(x_i) - g(x_j)\|_2 &\geq \left( \sum_{k<l} \|x_k - x_l\|_2^2 \right) \sum_{i<j} \frac{p_{ij}}{\sqrt{(x_i - x_j)^T P^{-1} (x_i - x_j)}} \\ &\geq \left( \sum_{k<l} \|x_k - x_l\|_2^2 \right) \left( \sum_{i<j} p_{ij} (x_i - x_j)^T P^{-1} (x_i - x_j) \right)^{-1/2} \\ &\hspace{15em} \text{[by Jensen's inequality]} \\ &= \left( \sum_{k<l} \|x_k - x_l\|_2^2 \right) \left( \text{Tr} \left( P^{-1/2} P P^{-1/2} \right)^{-1/2} \right) \\ &= \left( \sum_{k<l} \|x_k - x_l\|_2^2 \right) \text{Tr} (\mathbb{I}_d)^{-1/2} \\ &= \frac{1}{\sqrt{d}} \sum_{i<j} \|x_i - x_j\|_2^2. \end{aligned}$$

We note that if  $P$  is not invertible then the same proof can be carried out using pseudo-inverse of  $P$  instead.  $\blacktriangleleft$

► **Remark.** Although an enclosing ellipsoid of approximately optimal volume can be computed by a convex program [12], the proof of Theorem 1.1 requires a stronger, spectral approximation to the quadratic form of the minimum enclosing ellipsoid. We are not aware of any efficient algorithms for this. On the other hand, sampling  $(i, j)$  with probability  $\propto \|x_i - x_j\|_2^2$  can be done in  $O(nd)$  time as follows. First we compute the mean  $\mu = \sum_{i=1}^n x_i/n$ , and all the marginals for  $(i, \cdot)$  using

$$\sum_{j=1}^n \|x_j - x_i\|_2^2 = \sum_{j=1}^n \|x_j - \mu\|_2^2 + n \| \mu - x_i \|_2^2.$$

Now we can first sample  $i$  from the marginals, and then sample  $j$  with probability  $\propto \|x_i - x_j\|_2^2$ . This takes  $O(nd)$  time in total.

Theorem 4.1 immediately gives an efficient  $\sqrt{d}$  approximation algorithm for UNIFORM SPARSEST CUT when the SDP optimum solution resides in  $\mathbb{R}^d$ . Furthermore, as we point out next, the same proof can be tweaked to yield a similar result for the general SPARSEST CUT problem.

► **Theorem 4.2** (SPARSEST CUT SDP rounding in dimension  $d$ ). *A SPARSEST CUT instance  $C, D$  with SDP optimum solution in  $\mathbb{R}^d$  has an integrality gap of at most  $\sqrt{d}$ .*

## 10:12 Embedding Approximately Low-Dimensional $\ell_2^2$ Metrics Into $\ell_1$

**Proof.** Let  $x_1, \dots, x_n$  be the optimum solution in  $\mathbb{R}^d$  to the SPARSEST CUT SDP. We slightly modify the embedding given in the proof of Theorem 4.1, by choosing the  $p_{ij}$ 's based on the demand graph  $D$ . Let  $P = \sum_{k < l} p_{kl} (x_k - x_l)(x_k - x_l)^T \in \mathbb{R}^{d \times d}$ , where  $p_{kl}$ 's define a probability distribution with  $p_{kl} \propto d_{kl} \|x_k - x_l\|_2^2$ . We define the embedding as  $x_i \mapsto g(x_i) = P^{1/2} x_i$ . Lemma 2.2 shows that it is a contraction. We now need to show  $\sum_{i < j} d_{ij} \|g(x_i) - g(x_j)\|_2 \geq \frac{1}{\sqrt{d}} \sum_{i < j} d_{ij} \|x_i - x_j\|_2^2$ . It is easy to check that the same proof goes through without any major changes. ◀

By a well-known duality (cf. [16, Proposition 15.5.2 and Exercise 4]), Theorem 4.2 also implies Goemans' worst-case distortion result (Theorem 1.1), although non-constructively.

**Acknowledgements.** We thank Luca Trevisan for helpful discussions and suggestions, in particular, for bringing to our attention that Goemans' Theorem was true even with approximate triangle inequalities.

---

### References

- 1 Martin Aigner and Günter M. Ziegler. *Proofs from THE BOOK*. Springer, 4th edition, 2009. doi:10.1007/978-3-662-44205-0.
- 2 Sanjeev Arora, James R. Lee, and Assaf Naor. Euclidean distortion and the sparsest cut. *J. Amer. Math. Soc.*, 21:1–21, 2008. (Preliminary version in *37th STOC*, 2008). doi:10.1090/S0894-0347-07-00573-5.
- 3 Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2), 2009. (Preliminary version in *36th STOC*, 2004). doi:10.1145/1502793.1502794.
- 4 Boaz Barak, Prasad Raghavendra, and David Steurer. Rounding semidefinite programming hierarchies via global correlation. In *Proc. 51st IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 472–481, 2011. arXiv:1104.4680, doi:10.1109/FOCS.2011.95.
- 5 Jean Bourgain and Lior Tzafriri. Invertibility of large submatrices with applications to the geometry of Banach spaces and harmonic analysis. *Israel Journal of Mathematics*, 57(2):137–224, 1987. doi:0.1007/BF02772174.
- 6 Jeff Cheeger, Bruce Kleiner, and Assaf Naor. A  $(\log n)^{\Omega(1)}$  integrality gap for the sparsest cut SDP. In *Proc. 50th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 555–564, 2009. arXiv:0910.2024, doi:10.1109/FOCS.2009.47.
- 7 Amit Deshpande and Rakesh Venkat. Guruswami-Sinop rounding without higher level Lasserre. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore, editors, *Proc. 17th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 28 of *LIPICs*, pages 105–114. Schloss Dagstuhl, 2014. arXiv:1406.7279, doi:10.4230/LIPICs.APPROX-RANDOM.2014.105.
- 8 Shayan Oveis Gharan and Luca Trevisan. Improved ARV rounding in small-set expanders and graphs of bounded threshold rank. 2013. arXiv:1304.2060.
- 9 Venkatesan Guruswami and Ali Kemal Sinop. Approximating non-uniform sparsest cut via generalized spectra. In *Proc. 24th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 295–305, 2013. arXiv:1112.4109, doi:10.1137/1.9781611973105.22.
- 10 Piotr Indyk and Jirí Matoušek. Low-distortion embeddings of finite metric spaces. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 177–196. Chapman and Hall/CRC, 2nd edition, 2004. doi:10.1201/9781420035315.ch8.

- 11 Daniel M. Kane and Raghu Meka. A PRG for Lipschitz functions of polynomials with applications to sparsest cut. In *Proc. 45th ACM Symp. on Theory of Computing (STOC)*, pages 1–10, 2013. [arXiv:1211/1109](#), [doi:10.1145/2488608.2488610](#).
- 12 Leonid G. Khachiyan. Rounding of polytopes in the real number model of computation. *Math. Oper. Res.*, 21(2):307–320, 1996. [doi:10.1287/moor.21.2.307](#).
- 13 Tsz Chiu Kwok, Lap Chi Lau, Yin Tat Lee, Shayan Oveis Gharan, and Luca Trevisan. Improved Cheeger’s inequality: analysis of spectral partitioning algorithms through higher order spectral gap. In *Proc. 45th ACM Symp. on Theory of Computing (STOC)*, pages 11–20, 2013. [arXiv:1301.5584](#), [doi:10.1145/2488608.2488611](#).
- 14 Nathan Linial. Finite metric spaces: combinatorics, geometry and algorithms. In *Proc. of the ICM, Beijing*, volume 3, pages 573–586, 2002. [arXiv:math/0304466](#).
- 15 Avner Magen and Mohammad Moharrami. On the nonexistence of dimension reduction for  $\ell_2^2$  metrics. In *Proc. 20th Annual Canadian Conf. on Comp. Geom.*, 2008. URL: <http://cccg.ca/proceedings/2008/paper37full.pdf>.
- 16 Jirí Matoušek. Embedding finite metric spaces into normed spaces. In *Lectures on Discrete Geometry*, Graduate Texts in Mathematics, chapter 5, pages 355–400. Springer, 2002. [doi:10.1007/978-1-4613-0039-7\\_15](#).
- 17 Joel A. Tropp. Column subset selection, matrix factorization, and eigenvalue optimization. In *Proc. 20th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 978–986, 2009. URL: <http://dl.acm.org/citation.cfm?id=1496770.1496876>, [arXiv:0806.4404](#).



# Relational Logic with Framing and Hypotheses\*

Anindya Banerjee<sup>1</sup>, David A. Naumann<sup>2</sup>, and Mohammad Nikouei<sup>3</sup>

1 IMDEA Software Institute, Madrid, Spain

2 Stevens Institute of Technology, Hoboken, USA

3 Stevens Institute of Technology, Hoboken, USA

---

## Abstract

Relational properties arise in many settings: relating two versions of a program that use different data representations, noninterference properties for security, etc. The main ingredient of relational verification, relating aligned pairs of intermediate steps, has been used in numerous guises, but existing relational program logics are narrow in scope. This paper introduces a logic based on novel syntax that weaves together product programs to express alignment of control flow points at which relational formulas are asserted. Correctness judgments feature hypotheses with relational specifications, discharged by a rule for the linking of procedure implementations. The logic supports reasoning about program-pairs containing both similar and dissimilar control and data structures. Reasoning about dynamically allocated objects is supported by a frame rule based on frame conditions amenable to SMT provers. We prove soundness and sketch how the logic can be used for data abstraction, loop optimizations, and secure information flow.

**1998 ACM Subject Classification** F.3.1 Specifying and Verifying and Reasoning about Programs

**Keywords and phrases** Relational Hoare logic, program equivalence, product programs, frame conditions, region logic

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.11

## 1 Introduction

Relational properties are ubiquitous. Compiler optimizations, changes of data representation, and refactoring involve two different programs. Non-interference (secure information flow) is a non-functional property of a single program; it says the program preserves a “low indistinguishability” relation [44]. Many recent works deal with one or more of these applications, using relational logic and/or some form of product construction that reduces the problem to partial correctness, though mostly for simple imperative programs. This paper advances extant work by providing a relational logic for local reasoning about heap data structures and programs with procedures.

To set the stage, first consider the two simple imperative programs:

$$C \hat{=} x := 1; \text{ while } y > 0 \text{ do } x := x * y; y := y - 1 \text{ od}$$
$$C' \hat{=} x := 1; y := y - 1; \text{ while } y \geq 0 \text{ do } x := x * y + x; y := y - 1 \text{ od}$$

---

\* A. Banerjee was partially supported by the U.S. National Science Foundation (NSF). D. A. Naumann and M. Nikouei were partially supported by NSF award 1228930. Any opinion, findings, and conclusions or recommendations expressed in the material are those of the authors and do not necessarily reflect the views of NSF.



## 11:2 Relational Logic with Framing and Hypotheses

Both  $C$  and  $C'$  change  $x$  to be the factorial of the initial value of  $y$ , or to 1 if  $y$  is initially negative. For a context where  $y$  is known to be positive and its final value is not used, we could reason that they are interchangeable by showing both

$$C : y = z \wedge y \geq 0 \rightsquigarrow x = z! \quad \text{and} \quad C' : y = z \wedge y \geq 0 \rightsquigarrow x = z! \quad (1)$$

This is our notation for partial correctness judgments, with evident pre- and postconditions, for  $C$  and  $C'$ . It is not always easy to express and prove functional correctness, which motivates a less well developed approach to showing interchangeability of the examples. The two programs have a relational property which we write as

$$(C|C') : \mathbb{B}(y \geq 0) \wedge y \doteq y \approx x \doteq x \quad (2)$$

This relational correctness judgment says that a pair of terminating executions of  $C$  and  $C'$ , from a pair of states which both satisfy  $y \geq 0$  and which agree on the value of  $y$ , yields a pair of final states that agree on the value of  $x$ . The relational formula  $x \doteq x$  says that the value of  $x$  in the left state is the same as its value in the right state.

Property (2) is a consequence of functional correctness (1), but there is a direct way to prove it. Any pair of runs, from states that agree on  $y$ , can be aligned in such a way that both  $x \doteq x$  and  $y \doteq y + 1$  hold at the aligned pairs of intermediate states. The alignment is almost but not quite step by step, owing to the additional assignment in  $C'$ . The relational property is more complicated than partial correctness, in that it involves pairs of runs. On the other hand the requisite intermediate assertions are much simpler; they do not involve  $!$  which is recursively defined. Prior work showed such assertions are amenable to automated inference (see Section 7).

Despite the ubiquity of relational properties and recent logic-based or product-based approaches to reasoning with them (see Section 7), simple heap-manipulating examples like the following remain out of reach:

$$C'' \doteq xp := \text{new Int}(1); \text{ while } y > 0 \text{ do } xp.\text{set}(xp.\text{get()} * y); y := y - 1 \text{ od}; x := xp.\text{get}()$$

This Java-like program uses get/set procedures acting on an object that stores an integer value, and  $(C|C'')$  satisfies the same relational specification as (2). This code poses significant new challenges. It is not amenable to product reductions that rely on renaming of identifiers to encode two states as a single state: encoding of two heaps in one can be done, but at the cost of significant complexity [35] or exposing an underlying heap model below the level of abstraction of the programming language. Code like  $C''$  also needs to be linked with implementations of the procedures it calls. For reasoning about two versions of a module or library, relational hypotheses are needed, and calls need to be aligned to enable use of such hypotheses.

Floyd [22] articulates the fundamental method of inductive assertions for partial correctness: establish that certain conditions hold at certain intermediate steps of computation, designating those conditions/steps by associating formulas with control flow points. For relational reasoning, pairs of steps need to be aligned and it is again natural to designate those in terms of points in control flow. Alignment of steps has appeared in many guises in prior work, often implicit in simulation proofs but explicit in a few works [47, 8, 28].

**First contribution:** In this paper we embody the alignment principle in a formal system at the level of abstraction of the programming language – as Hoare logic does for the inductive assertion method – with sufficient generality to encompass many uses of relational



properties for programs including procedures and dynamically allocated mutable objects. Our logic (Section 6) manifests the reasoning principle directly, in structured syntax. It also embodies other reasoning principles, such as frame rules, case analysis, and hypothetical specifications for procedures. The rules encompass relations between both similarly- and differently-structured programs, and handle partially and fully aligned iterations. This achievement brings together ideas from many recent works (Section 7), together with two ingredients we highlight as contributions in their own right.

**Second contribution:** Our relational assertion language (Section 4) can describe agreement between unbounded pointer structures, allowing for differences in object allocation, as is needed to specify noninterference [4] and for simulation relations [3] in languages like Java and ML where references are abstract. Such agreements are expressed without the need for recursively defined predicates, and the assertion language has a direct translation to SMT-friendly encodings of the heap. (For lack of space we do not dwell on such encodings in this paper, which has a foundational focus, but see [40, 7].)

**Third contribution:** We introduce a novel form of “biprogram” (Section 5) that makes explicit the reasoner’s choice of alignments. A biprogram run models an aligned pair of executions of the underlying programs. The semantics of biprograms involves a number of subtleties: To provide a foundation for extending the logic with encapsulation (based on [5]), we need to use small-step semantics – which makes it difficult to prove soundness of linking, even in the unary case [5]. For this to work we need to keep the semantics deterministic and to deal with semantics of hypotheses in judgments.

Section 2 provides background and Section 3 is an overview of the logic using examples. We have chosen to use the available space to explain fundamental intuitions. An accompanying technical report includes worked proofs of the examples, additional examples like a loop tiling transformation, details of semantics, and the soundness theorem.

## 2 Background: synopsis of region logic

For reasoning about the heap, separation logic is very effective, with modal operators that implicitly describe heap regions. But for relations on unbounded heap structures at the Java/ML level of abstraction we need explicit means to refer to heap regions, as in the dependency logic of Amtoft et al. [2]. Our relational logic is based on an underlying unary logic dubbed “region logic” (*RL*), developed in a series of papers [10, 5, 7] to which we refer for rationale and omitted details. RL is a Hoare logic augmented with some side conditions (first order verification conditions) which facilitate local reasoning about frame conditions [10] in the manner of dynamic frames [27, 31]. In the logic such reasoning hinges on a frame rule. In a verifier, framing can be done by the VC-generator, optionally guided by annotation [40]. Stateful frame conditions also support an approach to encapsulation that validates a second order frame rule (at the cost of needing to use small-step semantics) [5]. Read effects enable the use of pure method calls in assertions and in frame conditions [7] and are useful for proving some equivalences, like commuting assignments, that hold in virtue of disjointness of effects [15].

The logic is formalized for imperative programs with first order procedures and dynamically allocated mutable objects (records), see Fig. 1. As in Java and ML, references are distinct from integers; they can be tested for equality but there is no pointer arithmetic. Typing

## 11:4 Relational Logic with Framing and Hypotheses

$$\begin{array}{l}
m \in ProcName \quad x, y, r \in VarName \quad f, g \in FieldName \quad K \in DeclaredClassNames \\
\text{(Types)} \quad T ::= \text{int} \mid \text{bool} \mid \text{rgn} \mid K \\
\text{(Program Expr.) } E ::= x \mid c \mid \text{null} \mid E \oplus E \quad \text{where } c \text{ is in } \mathbb{Z} \text{ and } \oplus \text{ in } \{=, +, -, *, \geq, \wedge, \neg, \dots\} \\
\text{(Region Expr.) } G ::= x \mid \emptyset \mid \{E\} \mid G^{\bullet} f \mid G \otimes G \quad \text{where } \otimes \text{ is in } \{\cup, \cap, \setminus\} \\
\text{(Expressions)} \quad F ::= E \mid G \\
\text{(Atomic comm.) } A ::= \text{skip} \mid m() \mid x := F \mid x := \text{new } K \mid x := x.f \mid x.f := x \\
\text{(Commands)} \quad C ::= A \mid \text{let } m = C \text{ in } C \mid \text{if } E \text{ then } C \text{ else } C \mid \text{while } E \text{ do } C \mid C ; C \\
\text{(Biprograms)} \quad CC ::= (C|C) \mid [A] \mid \text{let } m = (C|C) \text{ in } CC \mid CC ; CC \\
\quad \quad \quad \mid \text{if } E|E \text{ then } CC \text{ else } CC \mid \text{while } E|E \bullet \mathcal{P}|\mathcal{P} \text{ do } CC
\end{array}$$

■ **Figure 1** Programs and biprograms. Assume each class type  $K$  has a declared list of fields,  $\bar{f} : \bar{T}$ . Biprograms are explained in Section 3.

of programs is standard. In specifications we use ghost variables and fields of type **rgn**. A **region** is a set of object references, which may include the improper null reference.

A **specification**  $P \rightsquigarrow Q [\varepsilon]$  is comprised of precondition  $P$ , postcondition  $Q$ , and frame condition  $\varepsilon$ . Frame conditions include both read and write effects:

$$\varepsilon ::= \text{rd } x \mid \text{rd } G^{\bullet} f \mid \text{wr } x \mid \text{wr } G^{\bullet} f \mid \varepsilon, \varepsilon \mid (\text{empty})$$

The form  $\text{rd } G^{\bullet} f$  means the program may read locations  $o.f$  where  $o$  is a reference in the region denoted by expression  $G$ . We write  $\text{rw } x$  to abbreviate the composite effect  $\text{rd } x, \text{wr } x$ , and omit repeated tags:  $\text{rd } x, y$  abbreviates  $\text{rd } x, \text{rd } y$ . Predicate formulas  $P$  include standard first order logic with equality, region subset ( $G \subseteq G$ ), and the “points-to” relation  $x.f = E$ , which says  $x$  is non-null and the value of field  $f$  equals  $E$ . A **correctness judgment** has the form  $\Phi \vdash C : P \rightsquigarrow Q [\varepsilon]$  where the **hypothesis context**  $\Phi$  maps procedure names to specifications. In  $C$  there may be **environment calls** to procedures bound by **let** inside  $C$ , and also **context calls** to procedures in  $\Phi$ . The form  $G^{\bullet} f$  is termed an **image expression**. For an example of image expressions, consider this command which sums the elements of a singly-linked null-terminated list, ignoring nodes for which a deletion flag,  $del$ , has been set.

$$C_1 \hat{=} s := 0; \text{while } p \neq \text{null} \text{ do if } \neg p.del \text{ then } s := s + p.val \text{ fi; } p := p.nxt \text{ od}$$

For its specification we use ghost variable  $r : \text{rgn}$  to contain the nodes. Its being closed under  $nxt$  is expressed by  $r^{\bullet}nxt \subseteq r$  in this specification:

$$p \in r \wedge r^{\bullet}nxt \subseteq r \rightsquigarrow s = \text{sum}(\text{listnd}(\text{old}(p))) \text{ [rw } s, p, \text{rd } r, r^{\bullet}val, r^{\bullet}nxt, r^{\bullet}del]$$

The  $r$ -value of the image expression  $r^{\bullet}nxt$  is the set of values of  $nxt$  fields of the objects in  $r$ . In frame conditions, expressions are used for their l-values. In this case, the frame condition uses image expressions to say that for any object  $o$  in  $r$ , locations  $o.val, o.nxt, o.del$  may be read. The frame condition also says that variables  $s$  and  $p$  may be both read and written. Let function  $\text{listnd}$  give the mathematical list of non-deleted values.

Some proof rules in RL have side conditions which are first order formulas on one or two states. One kind of side condition, dubbed the “frames judgment”, delimits the part of state on which a formula depends (its read effect). RL’s use of stateful frame conditions provides for a useful frame rule, and even second order frame rule [37, 5], but there is a price to be paid. Frame conditions involving state dependent region expressions are themselves susceptible to interference by commands. That necessitates side conditions, termed “immunity” and “read-framed”, in the proof rules for sequence and iteration [5, 7]. The frame rule allows to infer from  $\Phi \vdash C : P \rightsquigarrow Q [\varepsilon]$  the conclusion  $\Phi \vdash C : P \wedge R \rightsquigarrow Q \wedge R [\varepsilon]$  provided that

$R$  is framed by read effects  $\eta$  (written  $\eta \text{ frm } R$ ) for locations disjoint from those writable according to  $\varepsilon$  (written  $\eta \cdot / \cdot \varepsilon$ ).

In keeping with our goal to develop a comprehensive deductive system, our unary and relational logics include a rule for discharging hypotheses, expressed in terms of the linking construct. Here is the special case of a single non-recursive procedure.

$$\text{LINK} \frac{m : R \rightsquigarrow S [\eta] \vdash C : P \rightsquigarrow Q [\varepsilon] \vdash B : R \rightsquigarrow S [\eta]}{\vdash \text{let } m = B \text{ in } C : P \rightsquigarrow Q [\varepsilon]}$$

### 3 Overview of the relational logic

This section sketches highlights of relational reasoning about a number of illustrative examples, introducing features of the logic incrementally. Some details are glossed over.

We write  $(C|C') : \mathcal{Q} \approx \mathcal{R}$  to express that a pair of programs  $C, C'$  satisfies the relational contract with precondition  $\mathcal{Q}$  and postcondition  $\mathcal{R}$ , leaving aside frame conditions for now. The judgment constrains executions of  $C$  and  $C'$  from pairs of states related by  $\mathcal{Q}$ . (For the grammar of relational formulas, see (7) in Section 4.) It says neither execution faults (e.g., due to null dereference), and if both terminate then the final states are related by  $\mathcal{R}$ . Moreover no context procedure is called outside its precondition. (We call this property the  $\forall\forall$  form, for contrast with refinement properties of  $\forall\exists$  form.)

Assume  $f, g$  are pure functions. The programs

$$C_0 \hat{=} x := f(z); y := g(z) \quad C'_0 \hat{=} y := g(z); x := f(z)$$

are equivalent. Focusing on relevant variables, the equivalence can be specified as

$$(C_0 | C'_0) : z \dot{=} z \approx x \dot{=} x \wedge y \dot{=} y \quad (3)$$

which can be proved as follows. Both  $C_0$  and  $C'_0$  satisfy  $\text{true} \rightsquigarrow x = f(z) \wedge y = g(z)$ , which directly entails that  $(C_0 | C'_0) : \mathbb{B}\text{true} \approx \mathbb{B}(x = f(z) \wedge y = g(z))$  by an embedding rule. The general form of embedding combines two different unary judgments, with different specifications, using relational formulas that assert a predicate on just the left ( $\triangleleft$ ) or right ( $\triangleright$ ) state. So  $\mathbb{B}P$  is short for  $\triangleleft P \wedge \triangleright P$ . Since  $z$  is not written by  $C_0$  or  $C'_0$ , we can introduce  $z \dot{=} z$  using the relational frame rule, to obtain  $(C_0 | C'_0) : z \dot{=} z \approx \mathbb{B}(x = f(z) \wedge y = g(z)) \wedge z \dot{=} z$ . This yields (3) using the relational rule of consequence with the two valid relational assertion schemas  $u \dot{=} u' \wedge \triangleleft(u = v) \wedge \triangleright(u' = v') \Rightarrow v \dot{=} v'$  and  $z \dot{=} z \Rightarrow f(z) \dot{=} f(z)$ .

For the factorial example  $(C|C')$  in Section 1, we would like to align the loops and use the simple relational invariant  $x \dot{=} x \wedge y \dot{=} y + 1$ . We consider the form  $(C|C')$  as a biprogram which can be rewritten to equivalent forms using the *weaving* relation which preserves the underlying programs but aligns control points together so that relational assertions can be used. (A minor difference from most other forms of product program is that we do not need to rename apart the variables on the left and right.) The weaving relation is given in Section 5. In this case we weave to the form

$$(x := 1 | x := 1; y := y - 1); \text{while } y > 0 \mid y \geq 0 \text{ do } (x := x * y \mid x := x * y + x); [y := y - 1]$$

This enables us to assert the relational invariant at the beginning and end of the loop bodies. Indeed, we can also assert it just before the last assignments to  $y$ . The rule for this form of loop requires the invariant to imply equivalence of the two loops' guard conditions, which it does:  $x \dot{=} x \wedge y \dot{=} y + 1 \Rightarrow (y > 0 \dot{=} y \geq 0)$ . For a biprogram of the *split* form  $(C|C')$ ,

## 11:6 Relational Logic with Framing and Hypotheses

the primary reasoning principle is the lifting of unary judgments about  $C$  and  $C'$ . For an atomic command  $A$ , the *sync* notation  $[A]$  is an alternative to  $(A|A)$  that indicates its left and right transition are considered together. This enables the use of relational specifications for procedures, and a relational principle for object allocation. For an ordinary assignment, *sync* merely serves to abbreviate, as in  $[y := y - 1]$  above.

The next example involves the heap and it also involves a loop that is “dissonant” in the sense that we do not want to align all iterations – that is, *alignment is ultimately about traces*, not program texts. Imagine the command  $C_1$  from Section 2 is run on a list from which secret values have been deleted. To specify that no secrets are leaked, we use the relational judgment  $(C_1|C_1) : listnd(p) \doteq listnd(p) \approx s \doteq s$  which says: Starting from any two states containing the same non-deleted values, terminating computations agree on the sums. The judgment can be proved by showing the functional property that  $s$  ends up as  $sum(listnd(old(p)))$ . But we can avoid reasoning about list sums and prove this relational property by aligning some of the loop iterations in such a way that  $listnd(p) \doteq listnd(p) \wedge s \doteq s$  holds at every aligned pair, that is, it is a relational invariant. Not every pair of loop iterations should be aligned: When  $p.del$  holds for the left state but not the right, a left-only iteration maintains the invariant, and *mutatis mutandis* when  $p.del$  holds only on the right. To handle such non-aligned iterations we use a novel syntactic annotation dubbed *alignment guards*. The idea is that the loop conditions are in agreement, and thus the iterations are synchronized, unless one of the alignment guards hold – and then that iteration is unsynchronized but the relational invariant must still be preserved. We weave  $(C_1|C_1)$  to the form

$$[s := 0]; \text{ while } p \neq \text{null} \mid p \neq \text{null} \bullet \triangleleft(p.del) \mid \triangleright(p.del) \quad (4)$$

$$\text{ do if } \neg p.del \mid \neg p.del \text{ then } [s := s + p.val] \text{ fi}; [p := p.next] \text{ od}$$

with alignment guards  $\triangleleft p.del$  and  $\triangleright p.del$ . The rule for the while biprogram has three premises for the loop body: for executions on the left (resp. right) under alignment guard  $\triangleleft p.del$  (resp.  $\triangleright p.del$ ) and for simultaneous executions when neither of the alignment guards hold. Each premise requires the invariant to be preserved.

The final example is a change of data representation. It illustrates dynamic allocation and frame conditions, as well as procedures and linking. A substantive example of this sort would be quite lengthy, so we contrive a toy example to provide hints of the issues that motivate various elements of our formal development. Our goal is to prove a conditional equivalence between these programs, whose components are defined in due course.

$$C_4 \hat{=} \text{let } push(x : \text{int}) = B \text{ in } Cli \quad C'_4 \hat{=} \text{let } push(x : \text{int}) = B' \text{ in } Cli$$

These differ only in the implementations  $B, B'$  of the stack interface (here stripped down to a single procedure), to which the client program  $Cli$  is linked. For modular reasoning, the unary contract for  $push$  should not expose details of the data representation. We also want to avoid reliance on strong functional specifications – the goal is equivalence of the two versions, not functional correctness of the client. The client, however, should respect encapsulation of the stack representation, to which end frame conditions are crucial. A simple pattern is for contracts to expose a ghost variable  $rep$  (of type  $\text{rgn}$ ) for the set of objects considered to be owned by a program module. Here is the specification for  $push$ , with parts named for later reference. Let  $size$  and  $rep$  be *spec-public*, i.e., they can be used in public contracts but not in client code [30].

$$push(x : \text{int}) : R \rightsquigarrow S[\eta] \text{ where } R \hat{=} size < 100 \quad (5)$$

$$S \hat{=} size = \text{old}(size) + 1$$

$$\eta \hat{=} \text{rw } rep, size, rep^{\text{any}}$$

Variables  $rep$  and  $size$  can be read and written (keyword  $rw$ ) by  $push$ . This needs to be explicit, even though client code cannot access them, because reasoning about client code involves them. The notation  $rep^{\text{any}}$  designates all fields of objects in  $rep$ ; these too may be read and written. The specification makes clear that calls to  $push$  affect the encapsulated state, while not exposing details. Here is one implementation of  $push(x)$ .

$$B \hat{=} top := \text{new Node}(top, x); rep := rep \cup \{top\}; size++$$

Variable  $top$  is considered internal to the stack module, so it need not appear in the frame condition. The alternate implementation of  $push$  replaces  $top$  by module variables  $free : \text{int}; slots : \text{String}[]$ ;

$$B' \hat{=} \text{if } slots = \text{null} \text{ then } slots := \text{new String}[100]; rep := rep \cup \{slots\}; free := 0 \text{ fi}; \\ slots[free++] := x; size++$$

Correctness of the two versions is proved using module invariants

$$I \hat{=} (top = \text{null} \wedge size = 0) \vee (top \in rep \wedge rep^{\text{next}} \subseteq rep \wedge size = \text{length}(\text{list}(top)))$$

$$I' \hat{=} (slots = \text{null} \wedge size = 0) \vee (slots \in rep \wedge size = free)$$

Here  $\text{list}(top)$  is the mathematical list of values reached from  $top$ . Recall that in an assertion the expression  $rep^{\text{next}}$  is the image of set  $rep$  under the  $\text{next}$  field, i.e., the set of values of  $\text{next}$  fields of objects in  $rep$ . The condition  $rep^{\text{next}} \subseteq rep$  says that  $rep$  is closed under  $\text{next}$ . This form is convenient in using ghost code to express shapes of data structures without recourse to reachability or other inductive predicates [10, 40].

As a specific  $Cli$ , we consider one that allocates and updates a node of the same type as used by the list implementation; this gets assigned to a global variable  $p$ .

$$Cli \hat{=} push(1); p := \text{new Node}(\text{null}, 2); p.val := 3; push(4)$$

Having completed the definitions of  $C_4, C'_4$  we can ask: In what sense are  $C_4, C'_4$  equivalent? A possible specification for  $(C_4|C'_4)$  requires agreement on  $size$  and ensures agreement on  $size$  and on  $p$  and  $p.val$ . However, the latter agreements cannot be literal equality: following the call  $push(1)$ , one implementation has allocated a  $Node$  whereas the array implementation has not. Depending on the allocator, different references may be assigned to  $p$  in the two executions. The appropriate relation is “equivalence modulo renaming of references” [2, 3, 4, 16, 17]. For region expression  $G$  and field name  $f$ , we write  $\mathbb{A}G^f$  for the **agreement** relation that says there is a partial bijection on references between the two states, that is total on the region  $G$ , and for which corresponding  $f$ -fields are equal. The notation  $\mathbb{A}G^{\text{any}}$  means agreement on all fields. In the present example, we only need the singleton region  $\{p\}$  containing the reference denoted by  $p$ .

To prove a relational judgment for  $(C_4|C'_4)$  we need suitable relational judgments for  $(B|B')$  for the implementations of  $push$ . It is standard [26] that they should preserve a “coupling relation” that connects the two data representations and also includes the data invariants for each representation. For the example, the connection is that the sequence of elements reached from  $top$ , written  $\text{list}(top)$ , is the same as the reversed sequence of elements in  $slots[0..free - 1]$ . Writing  $rev$  for reversal, we define the coupling and specification

$$\mathcal{L} \hat{=} \triangleleft I \wedge \triangleright I' \wedge LtR \quad LtR \hat{=} \text{list}(top) \doteq rev(\langle \rangle \text{ if } slots = \text{null} \text{ else } slots[0..free - 1]) \\ (C_4|C'_4) : \mathbb{B}(size = 0) \wedge \mathcal{L} \approx \triangleright p \doteq p \wedge size \doteq size \wedge \mathbb{A}\{p\}^{\text{any}} \wedge \mathcal{L} \quad (6)$$

We now proceed to sketch a proof of (6). First, we weave  $(C_4|C'_4)$  to let  $push(x : \text{int}) = (B|B')$  in  $\llbracket Cli \rrbracket$ . Here  $\llbracket Cli \rrbracket$  abbreviates the fully aligned biprogram  $[push(1)]; [p := \text{new Node}(\text{null}, 2)]; [p.val := 3]; [push(4)]$ . This biprogram simultaneously links the procedure bodies on left and right, and aligns the client. Using  $[p := \text{new Node}(\text{null}, 2)]$  enables use of a relational postcondition that says the objects are in agreement. Using  $[push(4)]$  enables use of  $push$ 's relational specification.

Like in unary RL, the proof rule for linking has two premises: one says the bodies  $(B|B')$  satisfy their specification, the other says  $\llbracket Cli \rrbracket$  satisfies the overall specification under the hypothesis that  $push$  satisfies its spec (see RLINK in Fig. 2). This hypothesis context gives  $push$  a relational specification, using  $\mathbb{A}x$  as sugar for  $x \doteq x$ :

$$\Phi \triangleq push(x) : \mathbb{B}R \wedge \mathbb{A}size \wedge \mathbb{A}x \wedge \mathcal{L} \approx \mathbb{B}S \wedge \mathbb{A}size \wedge \mathcal{L} [\eta, \text{rw top} \mid \eta, \text{rw slots}, \text{free}]$$

Here  $\eta$  is the effect  $\text{rw rep}, \text{size}, \text{rep}'\text{any}$  in the original specification (5) of  $push$ .

The specification in  $\Phi$  is not simply a relational lift of  $push$ 's public specification (5). Invariants  $I$  and  $I'$  on internal data structures should not appear in  $push$ 's API: they should be hidden, because the client should not touch the internal state on which they depend. Effects on module variables (like  $top$ ) should also be hidden. This kind of reasoning is the gist of second order framing [37, 5]. The relational counterpart is a relational second order frame rule which says that any client that respects encapsulation will preserve  $\mathcal{L}$ . Hiding is the topic of another paper, for which this one is laying the groundwork (see Section 8).

#### 4 Relational formulas

The relational assertion language is essentially syntax for a first order structure comprised of the variables and heaps of two states, together with a *refperm* connecting the states.

$$\mathcal{P} ::= F \doteq F \mid \mathbb{A}G'f \mid \diamond \mathcal{P} \mid \triangleleft \mathcal{P} \mid \triangleright \mathcal{P} \mid \mathcal{P} \wedge \mathcal{P} \mid \mathcal{P} \Rightarrow \mathcal{P} \mid \forall x|x' : K. \mathcal{P} \quad (7)$$

A *refperm* is a type-respecting partial bijection from references allocated in one state to references allocated in the other state. For use with SMT provers, a *refperm* can be encoded by a pair of maps with universal formulas stating they are inverse [7]. The syntax for relations caters for dynamic allocation by providing primitives such as  $F \doteq F'$  that says the value of  $F$  in the left state equals that of  $F'$  in the right state, modulo the *refperm*. In case of integer expressions, this is ordinary equality. For reference expressions, it means the two values are related by the *refperm*. For region expressions,  $G \doteq G'$  means the *refperm* forms a bijection between the reference set denoted by  $G$  in the left state and  $G'$  in the right state (ignoring *null*). The agreement formula  $\mathbb{A}G'f$  says, of a pair of states, that the *refperm* is total on the set denoted by  $G$  in the left state, and moreover the  $f$ -field of each object in that set has the same value, modulo *refperm*, as the  $f$ -field of its corresponding object in the right state.

For commands that allocate, the postcondition needs to allow the *refperm* to be extended, which is expressed by the modal operator  $\diamond$  (read “later”):  $\diamond \mathcal{P}$  holds if there is an extension of the *refperm* with zero or more pairs of references for which  $\mathcal{P}$  holds. For example, after the assignment to  $p$  in the stack example, the relational rule for allocation yields postcondition  $\diamond(p \doteq p \wedge \mathbb{A}\{p\}'\text{any})$ . Aside from the left and right embeddings of unary predicates ( $\triangleleft \mathcal{P}$  and  $\triangleright \mathcal{P}$ ), the only other constructs are the logical ones (conjunction, implication, quantification over values).

Let  $\square \mathcal{P} \triangleq \neg \diamond \neg \mathcal{P}$ . Validity of  $\mathcal{P} \Rightarrow \square \mathcal{P}$  is equivalent to  $\mathcal{P}$  being *monotonic*, i.e., not falsified by extension of the *refperm*. Here are some valid schemas:  $\mathcal{P} \Rightarrow \diamond \mathcal{P}$ ,  $\diamond \diamond \mathcal{P} \Rightarrow \diamond \mathcal{P}$ , and  $\diamond(\mathcal{P} \wedge \mathcal{Q}) \Rightarrow \diamond \mathcal{P} \wedge \diamond \mathcal{Q}$ . The converse of the latter is not valid. For framing, a key

property is that  $\diamond\mathcal{P} \wedge \mathcal{Q} \Rightarrow \diamond(\mathcal{P} \wedge \mathcal{Q})$  is valid if  $\mathcal{Q}$  is monotonic. In practice,  $\diamond$  is only needed in postconditions, and only at the top level. Owing to  $\diamond\diamond\mathcal{P} \Rightarrow \diamond\mathcal{P}$ , this works fine with sequenced commands. Many useful formulas are monotonic, including  $\text{AG}^c f$  and  $F \doteq F'$ , but not  $\neg(F \doteq F')$ .

## 5 Biprograms

A biprogram  $CC$  (Fig. 1) represents a pair of commands, which are given by syntactic projections defined by clauses including the following:  $\overline{(C|C')} \hat{=} C$ ,  $\overline{(C|C')} \hat{=} C'$ ,  $\overline{[A]} \hat{=} A$ ,  $\overline{\text{if } E|E' \text{ then } BB \text{ else } CC} \hat{=} \text{if } E \text{ then } \overline{BB} \text{ else } \overline{CC}$ , and  $\overline{\text{let } m = (C|C') \text{ in } CC} \hat{=} \text{let } m = C \text{ in } \overline{CC}$ . The weaving relation has clauses including the following.

$$\begin{aligned} (A|A) &\hookrightarrow [A] && \text{(for atomic commands } A) \\ (C;D | C';D') &\hookrightarrow (C|C'); (D|D') \\ (\text{if } E \text{ then } C \text{ else } D | \text{if } E' \text{ then } C' \text{ else } D') &\hookrightarrow \text{if } E|E' \text{ then } (C|C') \text{ else } (D|D') \\ (\text{while } E \text{ do } C | \text{while } E' \text{ do } C') &\hookrightarrow \text{while } E|E' \bullet \mathcal{P}|\mathcal{P}' \text{ do } (C|C') && \text{(for any } \mathcal{P}, \mathcal{P}') \end{aligned}$$

Additional clauses are needed for congruence, e.g.,  $CC \hookrightarrow DD$  implies  $BB;CC \hookrightarrow BB;DD$ . The loop weaving introduces chosen alignment guards. The **full alignment** of a command  $C$  is written  $\llbracket C \rrbracket$  and defined by  $\llbracket A \rrbracket \hat{=} [A]$ ,  $\llbracket C;D \rrbracket \hat{=} \llbracket C \rrbracket; \llbracket D \rrbracket$ ,  $\llbracket \text{if } E \text{ then } C \text{ else } D \rrbracket \hat{=} \text{if } E|E' \text{ then } \llbracket C \rrbracket \text{ else } \llbracket D \rrbracket$ ,  $\llbracket \text{while } E \text{ do } C \rrbracket \hat{=} \text{while } E|E' \bullet \text{false}|\text{false} \text{ do } \llbracket C \rrbracket$ , etc. Note that  $(C|C) \hookrightarrow^* \llbracket C \rrbracket$  for any  $C$ .

Commands are deterministic (modulo allocation), so termination-insensitive noninterference and equivalence properties can be expressed in a simple  $\forall\forall$  form described at the start of Section 3, rather than the  $\forall\exists$  form needed for refinement and for possibilistic noninterference (“for all runs . . . there exists a run . . .”). The transition rules for biprograms must ensure that the behavior is compatible with the underlying unary semantics, while enforcing the intended alignment. That would still allow some degree of nondeterminacy in biprogram transitions. However, we make biprograms deterministic (modulo allocation), because it greatly simplifies the soundness proofs. Rather than determinize by means of a scheduling oracle or other artifacts that would clutter the semantics, we build determinacy into the transition semantics. Whereas the syntax aligns points of interest in control flow, biprogram traces explicitly represent aligned pairs of executions. We make the arbitrary choice of left-then-right semantics for the split form. In a trace of  $(C|C')$ , every step taken by  $C$  is effectively aligned with the initial state for  $C'$ . This is followed by the steps of  $C'$ , each aligned with the final state of  $C$ . To illustrate the idea, here is a sketch of the trace of a split biprogram (center column) and its alignment with left and right unary traces.

$$\begin{array}{ccccc} \langle x:=0; y:=0 \rangle & \text{----} & \langle (x:=0; y:=0 | x:=0; y:=0) \rangle & \text{----} & \langle x:=0; y:=0 \rangle \\ \langle y:=0 \rangle & \text{-----} & \langle (y:=0 | x:=0; y:=0) \rangle & \text{-----} & \langle y:=0 \rangle \\ \langle \text{skip} \rangle & \text{-----} & \langle (\text{skip} | x:=0; y:=0) \rangle & \text{-----} & \langle \text{skip} \rangle \\ & & \langle (\text{skip} | y:=0) \rangle & & \langle y:=0 \rangle \\ & & \langle [\text{skip}] \rangle & & \langle \text{skip} \rangle \end{array}$$

This pattern is also typical for “high conditionals” in noninterference proofs, where different branches may be taken (cf. rule RIF4). Here is the sync’d version in action.

$$\begin{array}{ccccc} \langle x:=0; y:=0 \rangle & \text{----} & \langle [x:=0]; [y:=0] \rangle & \text{----} & \langle x:=0; y:=0 \rangle \\ \langle y:=0 \rangle & \text{-----} & \langle [y:=0] \rangle & \text{-----} & \langle y:=0 \rangle \\ \langle \text{skip} \rangle & \text{-----} & \langle [\text{skip}] \rangle & \text{-----} & \langle \text{skip} \rangle \end{array}$$



$$\begin{array}{c}
 \text{RLINK} \frac{m : \mathcal{R} \approx \mathcal{S} [\eta] \vdash \llbracket C \rrbracket : \mathcal{P} \approx \mathcal{Q} [\varepsilon] \quad \vdash (B|B') : \mathcal{R} \approx \mathcal{S} [\eta]}{\vdash \text{let } m = (B|B') \text{ in } \llbracket C \rrbracket : \mathcal{P} \approx \mathcal{Q} [\varepsilon]} \\
 \\
 \text{RIF4} \frac{\begin{array}{l} \Phi \vdash (C|C') : \mathcal{P} \wedge \triangleleft E \wedge \triangleright E' \approx \mathcal{Q} [\varepsilon|\varepsilon'] \quad \Phi \vdash (C|D') : \mathcal{P} \wedge \triangleleft E \wedge \triangleright \neg E' \approx \mathcal{Q} [\varepsilon|\varepsilon'] \\ \Phi \vdash (D|C') : \mathcal{P} \wedge \triangleleft \neg E \wedge \triangleright E' \approx \mathcal{Q} [\varepsilon|\varepsilon'] \quad \Phi \vdash (D|D') : \mathcal{P} \wedge \triangleleft \neg E \wedge \triangleright \neg E' \approx \mathcal{Q} [\varepsilon|\varepsilon'] \end{array}}{\Phi \vdash (\text{if } E \text{ then } C \text{ else } D | \text{if } E' \text{ then } C' \text{ else } D') : \mathcal{P} \approx \mathcal{Q} [\varepsilon, \text{ftpt}(E)|\varepsilon', \text{ftpt}(E')]} \\
 \\
 \text{RIF} \frac{\begin{array}{l} \mathcal{P} \Rightarrow E \doteq E' \\ \Phi \vdash CC : \mathcal{P} \wedge \triangleleft E \wedge \triangleright E' \approx \mathcal{Q} [\varepsilon|\varepsilon'] \quad \Phi \vdash DD : \mathcal{P} \wedge \triangleleft \neg E \wedge \triangleright \neg E' \approx \mathcal{Q} [\varepsilon|\varepsilon'] \end{array}}{\Phi \vdash \text{if } E|E' \text{ then } CC \text{ else } DD : \mathcal{P} \approx \mathcal{Q} [\varepsilon, \text{ftpt}(E)|\varepsilon', \text{ftpt}(E')]} \\
 \\
 \text{RWEAVE} \frac{\begin{array}{l} \Phi \vdash DD : \mathcal{P} \approx \mathcal{Q} [\varepsilon|\varepsilon'] \\ CC \hookrightarrow DD \quad \text{unaryOnly}(\Phi) \quad \text{terminates}(\overrightarrow{\mathcal{P}}, \overrightarrow{DD}) \quad \text{terminates}(\overrightarrow{\mathcal{P}}, \overrightarrow{DD}) \end{array}}{\Phi \vdash CC : \mathcal{P} \approx \mathcal{Q} [\varepsilon|\varepsilon']}
 \end{array}$$

■ **Figure 2** Selected relational proof rules.

The relational correctness judgment has the form  $\Phi \vdash CC : \mathcal{P} \approx \mathcal{Q} [\varepsilon|\varepsilon']$ . The hypothesis context  $\Phi$  maps some procedure names to their specifications:  $\Phi(m)$  may be a unary specification as before or else a relational one of the form  $\mathcal{R} \approx \mathcal{S} [\varepsilon|\varepsilon']$ . Frame conditions retain their meaning, separately for the left and the right side. In case  $\varepsilon$  is the same as  $\varepsilon'$ , the judgment or specification is abbreviated as  $\mathcal{P} \approx \mathcal{Q} [\varepsilon]$ .

The semantics of biprograms uses small steps, which makes alignments explicit. A configuration is comprised of a biprogram, two states, and two environments for procedures. The transition relation depends on a semantic interpretation for each procedure in the hypothesis context  $\Phi$ . Context calls, i.e., calls to procedures in the context, take a single step in accord with the interpretation. For the sake of determinacy, this is formalized in the semantics of relational correctness by quantifying over deterministic “interpretations” of the specifications (as in [7]), rather than a single nondeterministic transition rule (as in [5, 37]).

Let us sketch the semantic consistency theorem, which confirms that executions of a biprogram from a pair of states correspond to pairs of executions of the underlying commands, so that judgments about biprograms represent relational properties of the underlying commands. Suppose  $\Phi \vdash (C|C') : \mathcal{P} \approx \mathcal{Q} [\varepsilon|\varepsilon']$  is valid and  $\Phi$  has only unary specifications. Consider any states  $\sigma, \sigma'$  that are related by  $\mathcal{P}$  (modulo some reframe). Suppose  $C$  and  $C'$ , when executed from  $\sigma, \sigma'$ , reach final states  $\tau, \tau'$ . (In the formal semantics, transitions are defined in terms of interpretations  $\varphi$  that satisfy the specifications  $\Phi$ , so this is written  $\langle C, \sigma \rangle \xrightarrow{\varphi}^* \langle \text{skip}, \tau \rangle$  and  $\langle C', \sigma' \rangle \xrightarrow{\varphi}^* \langle \text{skip}, \tau' \rangle$ .) Then  $\tau, \tau'$  satisfy  $\mathcal{Q}$ .

## 6 Relational region logic

Selected proof rules appear in Fig. 2.

For linking a procedure with its implementation, rule RLINK caters for a client program  $C$  related to itself, in such a way that its executions can be aligned to use the same pattern of calls. The procedure implementations may differ, as in the stack example, Section 3. The rule shown here is for the special case of a single procedure, and the judgment for  $(B|B')$  has empty hypothesis context, to disallow recursion. We see no difficulty to add mutually recursive procedures, as done for the unary logic in [5], but have not yet included that in a



detailed soundness proof. The soundness proof is basically an induction on steps as in [5] but with the construction of an interpretation as in the proof of the linking rule in [7]. The general rule also provides for un-discharged hypotheses for ambient libraries used in the client and in the procedure implementations [5].

Rule rIF4 is the obvious rule that considers all paths for a conditional not aligned with itself (e.g., for “high branches”), whereas rIF leverages the alignment designated by the biprogram form. The disjunction rule – i.e., from  $\Phi \vdash CC : \mathcal{P}_0 \approx \mathcal{Q} [\varepsilon|\varepsilon']$  and  $\Phi \vdash CC : \mathcal{P}_1 \approx \mathcal{Q} [\varepsilon|\varepsilon']$  infer  $\Phi \vdash CC : \mathcal{P}_0 \vee \mathcal{P}_1 \approx \mathcal{Q} [\varepsilon|\varepsilon']$  – serves to split cases on the initial states, allowing different weavings to be used for different circumstances, which is why there is no notion like alignment guards for the biprogram conditional. The obvious conjunction rule is sound. It is useful for deriving other rules. For example, we have this simple axiom for allocation:  $\vdash [x := \text{new } K] : \text{true} \approx \diamond(x \doteq x) [\text{wr } x, \text{rw } \text{alloc}]$ . Using conjunction, embedding, and framing, one can add postconditions like  $\mathbb{A}\{x\}^f$  and freshness of  $x$ .

A consequence of our design decisions is “one-sided divergence” of biprograms, which comes into play with weaving. For example, assuming *loop* diverges,  $(y := 0; z.f := 0 \mid \text{loop}; x := 0)$  assigns  $z.f$  before diverging. But it weaves to  $(y := 0 \mid \text{loop}); (z.f := 0 \mid x := 0)$  which never assigns  $z.f$ . This biprogram’s executions do not cover all executions of the underlying unary programs. The phenomenon becomes a problem for code that can fault (e.g., if  $z$  is null). Were the correctness judgments to assert termination, this shortcoming would not be an issue, but in this paper we choose the simplicity of partial correctness. Rule rWEAVE needs to be restricted to prevent one-sided divergence of the premise biprogram *DD* from states where *CC* in the conclusion terminates. For simplicity in this paper we assume given a termination check:  $\text{terminates}(P, C)$  means that  $C$  faults or terminates normally, from any initial state satisfying  $P$ . This is about unary programs, so the condition can be discharged by standard means.

The relational frame rule is a straightforward extension of the unary frame rule. From a judgment  $\Phi \vdash CC : \mathcal{P} \approx \mathcal{Q} [\varepsilon|\varepsilon']$  it infers  $\Phi \vdash CC : \mathcal{P} \wedge \mathcal{R} \approx \mathcal{Q} \wedge \mathcal{R} [\varepsilon|\varepsilon']$  provided that  $\mathcal{R}$  is framed by read effects (on the left and right) that are disjoint from the write effects in  $\varepsilon|\varepsilon'$ .

To prove a judgment  $\Phi \vdash \text{while } E|E' \bullet \mathcal{P}|\mathcal{P}' \text{ do } CC : \mathcal{Q} \approx \mathcal{Q} [\varepsilon, \text{ftpt}(E)|\varepsilon', \text{ftpt}(E')]$ , the rule has three main premises:  $\Phi \vdash (\overleftarrow{CC}|\text{skip}) : \mathcal{Q} \wedge \mathcal{P} \wedge \triangleleft E \approx \mathcal{Q} [\varepsilon]$  for left-only execution of the body,  $\Phi \vdash (\text{skip}|\overrightarrow{CC}) : \mathcal{Q} \wedge \mathcal{P}' \wedge \triangleright E' \approx \mathcal{Q} [|\varepsilon']$  for right-only, and  $\Phi \vdash CC : \mathcal{Q} \wedge \neg \mathcal{P} \wedge \neg \mathcal{P}' \wedge \triangleleft E \wedge \triangleright E' \approx \mathcal{Q} [\varepsilon|\varepsilon']$  for aligned execution. A side condition requires that the invariant  $\mathcal{Q}$  implies these cases are exhaustive:  $\mathcal{Q} \Rightarrow E \doteq E' \vee (\mathcal{P} \wedge \triangleleft E) \vee (\mathcal{P}' \wedge \triangleright E')$ . Additional side conditions require the effects to be self-immune, just as in unary RL [10, 7]. Finally, the formulas  $\diamond \mathcal{P} \Rightarrow \mathcal{P}$  and  $\diamond \mathcal{P}' \Rightarrow \mathcal{P}'$  must be valid; this says the alignment guards are reframe-independent, which is needed because reframe are part of the semantics of judgments but are not part of the semantics of biprograms.

The above rule is compatible with weaving a loop body, as in (4). The left and right projections  $\overleftarrow{CC}$  and  $\overrightarrow{CC}$  undo the weaving and take care of unaligned iterations.

There are many other valid and useful rules. Explicit frame conditions are convenient, both in tools and in a logic, in part because they compose in simple ways. This may lose precision, but that can be overcome using postconditions to express, e.g., that  $x := x$  does not observably write  $x$ . This is addressed, in unary RL, by a rule to “mask” write effects [10]. Similarly, the relational logic supports a rule to mask read effects. There is a rule of transitivity along these lines: from  $(B|C) : \mathcal{P} \approx \mathcal{Q}$  and  $(C|D) : \mathcal{R} \approx \mathcal{S}$  infer  $(B|D) : \mathcal{P}; \mathcal{R} \approx \mathcal{Q}; \mathcal{S}$  where  $(;)$  denotes composition of relations. A special case is where the pre-relations (resp. post-relations) are the same, transitive, relation. The rule needs to take care about termination of  $C$ .

## 7 Related work

Benton [15] introduced relational Hoare logic, around the same time that Yang was developing relational separation logic [45]. Benton’s logic does not encompass the heap. Yang’s does; it features separating conjunction and a frame rule. Pointers are treated concretely in [45]; agreement means identical addresses, which suffices for some low level C code. Neither work includes procedures. Beringer [18] reduces relational verification to unary verification via specifications and uses that technique to derive rules of a relational Hoare logic for programs including the heap (but not procedures). Whereas the logics of Benton, Yang, and others provide only rules for synchronized alignment of loops, Beringer derives a rule that allows for unsynchronized (“dissonant”) iterations; our alignment guards are similar to side conditions of that rule. RHTT [34] implements a relational program logic in dependent type theory (Coq). The work focuses on applications to information flow. It handles dynamically allocated mutable state and procedures, and both similar and dissimilar control structures. Like the other relational logics it does not feature frame conditions. RHTT is the only prior relational logic to include both the heap and procedures, and the only one to have a procedure linking rule. It is also the only one to address any form of encapsulation; it does so using abstract predicates, as opposed to hiding [5, 37].

Several works investigate construction of product programs that encode nontrivial choices of alignment [38, 42, 46, 11, 12, 13]. In particular, our weaving relation was inspired by [11, 13] which address programs that differ in structure. In contrast to the 2-safety properties for deterministic programs considered in this paper and most prior work, Barthe et al. [12] handle properties of the form “for all traces . . . there exists a trace . . .” which are harder to work with but which encompass notions of refinement and continuity. Relational specifications of procedures are used in a series of papers by Barthe et al. (e.g., [14]) for computer-aided cryptographic proofs. Sousa and Dillig [41] implement a logic that encompasses  $k$ -ary relations, e.g., the 3-safety property that a binary method is computing a transitive relation; their verification algorithm is based on an implicit product construction. None of these works address the heap or the linking of procedure implementations. Several works show that syntactic heuristics can often find good weavings in the case of similarly-structured programs not involving the heap [28, 32, 41]. Mueller et al. [32] use a form of product program and a relational logic to prove correctness of a static analysis for dependency, including procedures but no heap.

Works on translation validation and conditional equivalence checking use verification conditions (VCs) with implicit or explicit product constructions [46, 47]. Godlin and Strichman formulate and prove soundness of rules for proving equivalence of programs with similar control structure [23]. They use one of the rules to devise an algorithm for VCs using uninterpreted functions to encode equivalence of called procedures, which has been implemented in two prototype tools for equivalence checking [24]. (Pointer structures are limited to trees, i.e., no sharing.) Hawblitzel et al. [25] and Lahiri et al. [29] use relational procedure summaries for intra- and inter-procedural reasoning about program transformations. The heap is modeled by maps. These and related works report good experimental results using SMT or SAT solvers to discharge VCs. Felsing et al. [21] use Horn constraint solving to infer coupling relations and relational procedure summaries, which works well for similarly structured programs; they do not deal with the heap. The purpose of our logic is not to supplant VC-based tools approaches but rather to provide a foundation for them. Our biprograms and relational assertions are easily translated to SMT-based back ends like Boogie and Why3.

Amtoft et al. [2] introduce a logic for information flow in object-based programs, using abstract locations to specify agreements in the heap. It was proposed in [8] to extend this approach to more general relational specifications, for fine-grained declassification policies. Banerjee et al. [9] showed how region-based reasoning including a frame rule can be encoded, using ghost code, with standard FOL assertions instead of an ancillary notion of abstract region. This evolved to the logic in Section 6.

Relational properties have been considered in the context of separation logic: [19] and [43] both give relational interpretations of unary separation logic that account for representation independence, using second order framing [19] or abstract predicates [43]. Extension of this work to a relational logic seems possible, but the semantics does not validate the rule of conjunction so it may not be a good basis for verification tools. Tools often rely heavily on splitting conjunctions in postconditions.

Ahmed et al. [1] address representation independence for higher order code and code pointers, using a step-indexed relational model, and prove challenging instances of contextual equivalence. Based on that work, Dreyer et al. [20] formulate a relational modal logic for proving contextual equivalence for a language that has general recursive types and general ML-style references atop System F. The logic serves to abstract from details of semantics in ways likely to facilitate interactive proofs of interesting contextual equivalences, but it includes intensional atomic propositions about steps in the transition semantics of terms. Whereas contextual equivalence means equivalent in all contexts, general relational logics can express equivalences conditioned on the initial state. For example, the assignments  $x := y.f$  and  $z.f := w$  do not commute, in general, because their effects can overlap. But they do commute under the precondition  $y \neq z$ . We can easily prove equivalence judgments such as  $(x := y.f; z.f := w \mid z.f := w; x := y.f) : \mathbb{B}(y \neq z) \wedge \mathbb{A}\{y\}'f \wedge w \doteq w \approx x \doteq x \wedge \mathbb{A}\{z\}'f$ . By contrast with [1, 34], we do not rely on embedding in higher-order logic.

## 8 Conclusion

We provide a general relational logic that encompasses the heap and includes procedures. It handles both similarly- and differently-structured programs. We use small-step semantics with the goal to leverage, in future work, our prior work on SMT-friendly heap encapsulation [40, 5, 7] for representation independence, which is not addressed in prior relational logics.<sup>1</sup>

As articulated long ago by Hoare [26] but never fully formalized in a logic of programs, reasoning about change of data representation is based on simulation relations on encapsulated state, which are necessarily preserved by client code in virtue of encapsulation. For functional correctness this corresponds to “hiding” of invariants on encapsulated data, i.e., not including the invariant in the specification used by a client. O’Hearn et al. [37] formalize this as a hypothetical or second order framing rule (which has been adapted to RL [5]). In ongoing work, the logic presented here has been extended to address encapsulation and provides a relational second order frame rule which embodies Reynolds’ abstraction theorem [39]. Whereas framing of invariants relies on write effects, framing of encapsulated relations also relies on read effects. Our ongoing work also addresses observational purity, which is known to be closely related to representation independence [26, 36].

Although we can prove equivalence for loop tiling, some array-oriented loop optimizations seem to be out of reach of the logic as currently formulated. Loop interchange changes

<sup>1</sup> With the partial exception of [1], see Section 7. Although there has been some work on observational equivalence for higher order programs, we are not aware of work dealing with general relational judgments for higher order programs.

matrix row to column order, reordering unboundedly many atomic assignments, as does loop fusion/distribution. Most prior work does not handle these examples; [47] does handle them, with a non-syntactic proof rule that involves permutations on transition steps, cf. [33].

---

## References

- 1 Amal Ahmed, Derek Dreyer, and Andreas Rossberg. State-dependent representation independence. In *ACM Symposium on Principles of Programming Languages*, 2009.
- 2 T. Amtoft, S. Bandhakavi, and A. Banerjee. A logic for information flow in object-oriented programs. In *ACM Symposium on Principles of Programming Languages*, 2006.
- 3 Anindya Banerjee and David A. Naumann. Ownership confinement ensures representation independence for object-oriented programs. *Journal of the ACM*, 52(6):894–960, 2005.
- 4 Anindya Banerjee and David A. Naumann. Stack-based access control for secure information flow. *Journal of Functional Programming*, 15(2):131–177, 2005.
- 5 Anindya Banerjee and David A. Naumann. Local reasoning for global invariants, part II: Dynamic boundaries. *Journal of the ACM*, 60(3):19:1–19:73, 2013.
- 6 Anindya Banerjee and David A. Naumann. A logical analysis of framing for specifications with pure method calls. In *Verified Software: Theories, Tools and Experiments*, volume 8471 of *LNCS*, 2014.
- 7 Anindya Banerjee, David A. Naumann, and Mohammad Nikouei. A logical analysis of framing for specifications with pure method calls. Under review for publication. Extended version of [6]. <http://www.cs.stevens.edu/~naumann/pub/readRL.pdf>, 2015.
- 8 Anindya Banerjee, David A. Naumann, and Stan Rosenberg. Expressive declassification policies and modular static enforcement. In *IEEE Symposium on Security and Privacy*, 2008.
- 9 Anindya Banerjee, David A. Naumann, and Stan Rosenberg. Regional logic for local reasoning about global invariants. In *European Conference on Object-Oriented Programming*, volume 5142 of *LNCS*, 2008.
- 10 Anindya Banerjee, David A. Naumann, and Stan Rosenberg. Local reasoning for global invariants, part I: Region logic. *Journal of the ACM*, 60(3):18:1–18:56, 2013.
- 11 Gilles Barthe, Juan Manuel Crespo, and César Kunz. Relational verification using product programs. In *Formal Methods*, volume 6664 of *LNCS*, 2011.
- 12 Gilles Barthe, Juan Manuel Crespo, and César Kunz. Beyond 2-safety: Asymmetric product programs for relational program verification. In *Logical Foundations of Computer Science, International Symposium*, volume 7734 of *LNCS*, 2013.
- 13 Gilles Barthe, Juan Manuel Crespo, and César Kunz. Product programs and relational program logics. *J. Logical and Algebraic Methods in Programming*, 2016. To appear.
- 14 Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. *ACM Trans. Program. Lang. Syst.*, 35(3):9, 2013.
- 15 Nick Benton. Simple relational correctness proofs for static analyses and program transformations. In *ACM Symposium on Principles of Programming Languages*, 2004.
- 16 Nick Benton, Martin Hofmann, and Vivek Nigam. Abstract effects and proof-relevant logical relations. In *ACM Symposium on Principles of Programming Languages*, 2014.
- 17 Nick Benton, Andrew Kennedy, Lennart Beringer, and Martin Hofmann. Relational semantics for effect-based program transformations with dynamic allocation. In *International Symposium on Principles and Practice of Declarative Programming*, 2007.
- 18 Lennart Beringer. Relational decomposition. In *Interactive Theorem Proving (ITP)*, volume 6898 of *LNCS*, 2011.
- 19 Lars Birkedal and Hongseok Yang. Relational parametricity and separation logic. *Logical Methods in Computer Science*, 4(2), 2008. doi:10.2168/LMCS-4(2:6)2008.

- 20 Derek Dreyer, Georg Neis, Andreas Rossberg, and Lars Birkedal. A relational modal logic for higher-order stateful ADTs. In *ACM Symposium on Principles of Programming Languages*, 2010.
- 21 Dennis Felsing, Sarah Grebing, Vladimir Klebanov, Philipp Rümmer, and Mattias Ulbrich. Automating regression verification. In *International Conference on Automated Software Engineering*, 2014.
- 22 Robert W. Floyd. Assigning meanings to programs. In *Proceedings of Symposia in Applied Mathematics 19*, pages 19–32. American Mathematical Society, 1967.
- 23 Benny Godlin and Ofer Strichman. Inference rules for proving the equivalence of recursive procedures. *Acta Inf.*, 45(6):403–439, 2008.
- 24 Benny Godlin and Ofer Strichman. Regression verification: proving the equivalence of similar programs. *Softw. Test., Verif. Reliab.*, 23(3):241–258, 2013.
- 25 Chris Hawblitzel, Ming Kawaguchi, Shuvendu K. Lahiri, and Henrique Rebêlo. Towards modularly comparing programs using automated theorem provers. In *International Conference on Automated Deduction*, 2013.
- 26 C. A. R. Hoare. Proofs of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.
- 27 Ioannis T. Kassios. The dynamic frames theory. *Formal Aspects of Computing*, 23(3):267–288, 2011. doi:10.1007/s00165-010-0152-5.
- 28 Máté Kovács, Helmut Seidl, and Bernd Finkbeiner. Relational abstract interpretation for the verification of 2-hypersafety properties. In *ACM Conference on Computer and Communications Security*, 2013.
- 29 Shuvendu K. Lahiri, Kenneth L. McMillan, Rahul Sharma, and Chris Hawblitzel. Differential assertion checking. In *Joint Meeting of the European Software Engineering Conference and the ACM Symposium on the Foundations of Software Engineering*, 2013.
- 30 Gary T. Leavens and Peter Müller. Information hiding and visibility in interface specifications. In *International Conference on Software Engineering*, 2007.
- 31 K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, LNCS, 2010.
- 32 Christian Mueller, Máté Kovács, and Helmut Seidl. An analysis of universal information flow based on self-composition. In *IEEE Computer Security Foundations Symposium*, 2015.
- 33 Kedar S. Namjoshi and Nimit Singhania. Loopy: Programmable and formally verified loop transformations. In *Static Analysis Symposium*, volume 9837 of LNCS, 2016.
- 34 Aleksandar Nanevski, Anindya Banerjee, and Deepak Garg. Dependent type theory for verification of information flow and access control policies. *ACM Trans. Program. Lang. Syst.*, 35(2):6, 2013. doi:10.1145/2491522.2491523.
- 35 David A. Naumann. From coupling relations to mated invariants for secure information flow. In *European Symposium on Research in Computer Security*, volume 4189 of LNCS, 2006.
- 36 David A. Naumann. Observational purity and encapsulation. *Theoretical Computer Science*, 376(3):205–224, 2007.
- 37 Peter W. O’Hearn, Hongseok Yang, and John C. Reynolds. Separation and information hiding. *ACM Transactions on Programming Languages and Systems*, 31(3):1–50, 2009.
- 38 John C. Reynolds. *The Craft of Programming*. Prentice-Hall, 1981.
- 39 John C. Reynolds. Types, abstraction, and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 1983*, pages 513–523. North-Holland, 1984.
- 40 Stan Rosenberg, Anindya Banerjee, and David A. Naumann. Decision procedures for region logic. In *Int’l Conf. on Verification, Model Checking, and Abstract Interpretation*, 2012.

## 11:16 Relational Logic with Framing and Hypotheses

- 41 Marcelo Sousa and Isil Dillig. Cartesian Hoare logic for verifying k-safety properties. In *ACM Conf. on Program. Lang. Design and Implementation*, 2016.
- 42 Tachio Terauchi and Alex Aiken. Secure information flow as a safety problem. In *International Static Analysis Symposium*, volume 3672 of *LNCS*, 2005.
- 43 Jacob Thamsborg, Lars Birkedal, and Hongseok Yang. Two for the price of one: Lifting separation logic assertions. *Logical Methods in Computer Science*, 8(3), 2012.
- 44 Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.
- 45 Hongseok Yang. Relational separation logic. *Theoretical Computer Science*, 375(1-3):308–334, 2007.
- 46 Anna Zaks and Amir Pnueli. CoVaC: Compiler validation by program analysis of the cross-product. In *Formal Methods*, volume 5014 of *LNCS*, 2008.
- 47 Lenore D. Zuck, Amir Pnueli, Benjamin Goldberg, Clark W. Barrett, Yi Fang, and Ying Hu. Translation and run-time validation of loop transformations. *Formal Methods in System Design*, 27(3):335–360, 2005.

# FO-Definable Transformations of Infinite Strings

Vrunda Dave<sup>1</sup>, Shankara Narayanan Krishna<sup>\*2</sup>, and  
Ashutosh Trivedi<sup>†3</sup>

1 Indian Institute of Technology, Bombay, India  
vrunda@cse.iitb.ac.in

2 Indian Institute of Technology, Bombay, India  
krishnas@cse.iitb.ac.in

3 Indian Institute of Technology, Bombay, India; and  
University of Colorado, Boulder, USA  
ashutosh.trivedi@colorado.edu

---

## Abstract

The theory of regular and aperiodic transformations of finite strings has recently received a lot of interest. These classes can be equivalently defined using logic (Monadic second-order logic and first-order logic), two-way machines (regular two-way and aperiodic two-way transducers), and one-way register machines (regular streaming string and aperiodic streaming string transducers). These classes are known to be closed under operations such as sequential composition and regular (star-free) choice; and problems such as functional equivalence and type checking, are decidable for these classes. On the other hand, for infinite strings these results are only known for regular transformations: Alur, Filiot, and Trivedi studied transformations of infinite strings and introduced an extension of streaming string transducers over infinite strings and showed that they capture monadic second-order definable transformations for infinite strings. In this paper we extend their work to recover connection for infinite strings among first-order logic definable transformations, aperiodic two-way transducers, and aperiodic streaming string transducers.

**1998 ACM Subject Classification** F.1.1 Models of Computation

**Keywords and phrases** Transducers, FO-definability, Infinite Strings

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.12

## 1 Introduction

The beautiful theory of regular languages is the cornerstone of theoretical computer science and formal language theory. The perfect harmony among the languages of finite words definable using abstract machines (deterministic finite automata, nondeterministic finite automata, and two-way automata), algebra (regular expressions and finite monoids), and logic (monadic second-order logic (MSO) [7]) set the stage for the generalizations of the theory to not only for the theory of regular languages of infinite words [8, 17], trees [4], partial orders [23], but more recently for the theory of regular transformations of the finite strings [6], infinite strings [3, 1], and trees [2]. For the theory of regular transformations it has been shown that abstract machines (two-way transducers [13] and streaming string transducers [6]) precisely capture the transformations definable via monadic second-order

---

\* Partly supported by CEFIPRA project AVeRTS.

† Supported by research sponsored by DARPA under agreement number FA8750-15-2-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.



© Vrunda Dave, Shankara Narayanan Krishna, and Ashutosh Trivedi;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

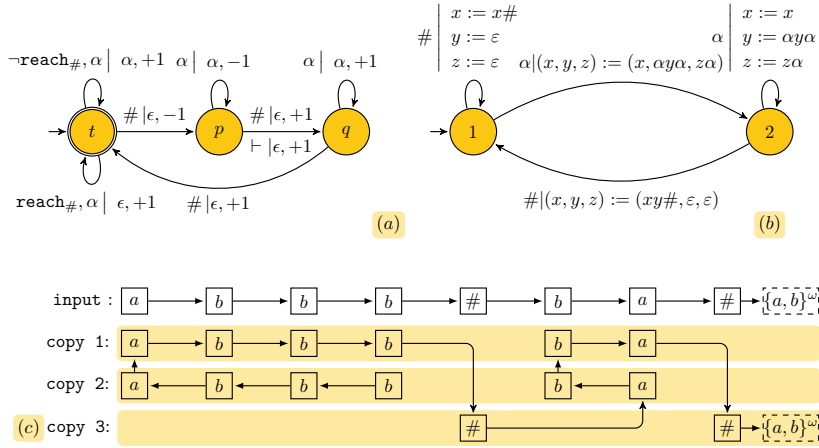
Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 12; pp. 12:1–12:14



Leibniz International Proceedings in Informatics

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





■ **Figure 1** Transformation  $f_1$  given as (a) two-way transducers with look-ahead (b) streaming string transducers with  $F(\{2\}) = xz$  is the output associated with Muller set  $\{2\}$ , and (c) FO-definable transformation for the string  $abbb\#ba\#\{a, b\}^\omega$ . Here symbol  $\alpha$  stands for both symbols  $a$  and  $b$ , and the predicate  $\text{reach}_\#$  is the lookahead that checks whether string contains a  $\#$  in future.

logic transformations [10]. For a detailed exposition on the regular theory of languages and transformations, we refer to the surveys by Thomas [23, 24] and Filiot [14], respectively.

There is an equally appealing and rich theory for first-order logic (FO) definable subclasses of regular languages. McNaughton and Papert [18] observed the equivalence between FO-definability and star-free regular expressions for finite words, while Ladner [16] and Thomas [22] extended this connection to infinite words. The equivalence of star-free regular expressions and languages defined via aperiodic monoids is due to Schützenberger [20] and corresponding extension to infinite words is due to Perrin [19]. For a detailed introduction to FO-definable language we refer the reader to Diekert and Gastin [12].

The results for the theory of FO-definable transformations are relatively recent. While Courcelle’s definition of logic based transformations [10] provides a natural basis for FO-definable transformations of finite as well as infinite words, [15] observed that over finite words, streaming string transducers [6] with an appropriate notion of aperiodicity precisely capture the same class of transformations. Carton and Dartois [9] introduced aperiodic two-way transducers for finite words and showed that it precisely captures the notion of FO-definability. We consider transformations of infinite strings and generalize these results by showing that appropriate aperiodic restrictions on two-way transducers and streaming string transducers on infinite strings capture the essence of FO-definable transformations. Let us study an example to see how the following  $\omega$ -transformation can be represented using logic, two-way transducers, and streaming string transducers.

► **Example 1 (Example Transformation).** Let  $\Sigma = \{a, b, \#\}$ . Consider an  $\omega$ -transformation  $f_1 : \Sigma^\omega \rightarrow \Sigma^\omega$  such that it replaces any maximal  $\#$ -free finite string  $u$  by  $\bar{u}u$ , where  $\bar{u}$  is the reverse of  $u$ . Moreover  $f_1$  is defined only for strings with finitely many  $\#$ ’s, e.g. for all  $w = u_1\#u_2\#\dots u_n\#v$  s.t  $u_i \in \{a, b\}^*$  and  $v \in \{a, b\}^\omega$ , we have  $f_1(w) = \bar{u}_1u_1\#\dots \#\bar{u}_nu_n\#v$ .

**Logic based transformations.** Logical descriptions of transformations of structures – as introduced by Courcelle [10] – work by introducing a fixed number of copies of the vertices of the input graph; and the domain, the labels and the edges of the output graph are defined by MSO formulae with zero, one or two free variables, respectively, interpreted over the input



graph. Figure 1(c) shows a way to express transformation  $f_1$  using three copies of the input with a) logical formula  $\phi_{\text{dom}}$  expressing the domain of the transformation, b) logical formulae  $\phi_\alpha^c(i)$  (with one free variable) for every copy  $c \in \{1, 2\}$  and letter  $\alpha \in \{a, b\}$  expressing the label of a position  $i$  for copy  $c$ , and c) logical formulae  $\phi^{c,d}(i, j)$  with two free variables expressing the edge from position  $i$  of copy  $c$  to position  $j$  of copy  $d$ . The formulae  $\phi_{\text{dom}}$ ,  $\phi_\alpha^c$ , and  $\phi^{c,d}$  are interpreted over input structure (in this paper always an infinite string), and it is easy to see that these formulae for our example can easily be expressed in MSO. In this paper we study logical transformations expressible with FO and to cover a larger class of transformations, we use natural order relation  $\prec$  for positions instead of the successor relation. We will later show that the transformation  $f_1$  indeed can be expressed using FO.

**Two-Way Transducers.** For finite string transformations, Engelfriet and Hoogeboom [13] showed that the finite-state transducers when equipped with a two-way input tape have the same expressive power as MSO transducers, and Carton and Dartois [9] recovered this result for FO transducers and two-way transducers with aperiodicity restriction. A crucial property of two-way finite-state transducers exploited in these proofs [13, 9] is the fact that transitions capable of regular (star-free) *look-ahead* (i.e., transitions that test the whole input string against a regular property) do not increase the expressiveness of regular (aperiodic) two-way transducers. However, this property does not hold in case of  $\omega$ -strings. In Figure 1(a), we show a two-way transducer characterizing transformation  $f_1$ . The transducer uses the lookahead  $\text{reach}_\#$  to check if the remaining part of the string contains a  $\#$  in future. A transition labeled  $\langle \phi, \alpha | \beta, +1 \rangle$  of the two-way transducer should be read as: if the current position on the string satisfies the look-ahead  $\phi$  and the current symbol is  $\alpha$  then output symbol  $\beta$  and move the input tape head to the right. This transducer works by first checking if the string contains a  $\#$  in the future of the current position, if so it moves its head all the way to the position before  $\#$  and starts outputting the symbols in reverse, and when it sees the end-marker or a  $\#$  it prints the string before the  $\#$ ; however, if there is no  $\#$  in future, then the transducer outputs the rest of the string. It is straightforward to verify that this transducer characterizes the transformation  $f_1$ . However, in the absence of the look-ahead a two-way transducer can not express this transformation.

**Streaming String Transducers.** Alur and Černý [6, 5] proposed a one-way finite-state transducer model, called the *streaming string transducers* (SST), that manipulates a finite set of string variables to compute its output, and showed that they have same expressive power as MSO transducers. SST, instead of appending symbols to the output tape, concurrently update all string variables using a concatenation of string variables and output symbols in a *copyless* fashion, i.e. no variable occurs more than once in each concurrent variable update. The transformation of a string is then defined using an output (partial) function  $F$  that associates states with a copyless concatenation of string variables, s.t. if the state  $q$  is reached after reading the string and  $F(q)=xy$ , then the output string is the final valuation of  $x$  concatenated with that of  $y$ . [3] generalized this by introducing a Muller acceptance condition to give an SST to characterize  $\omega$ -transitions. Figure 1(b) shows a streaming string transducer accepting the transformation  $f_1$ . It uses three string variables and concurrently prepends and/or appends these variables in a copyless fashion to construct the output. The acceptance set and the output is characterized by a Muller set (here  $\{2\}$  and its output  $xz$ ), such that if the infinitely visiting states set is  $\{2\}$  then the output is limit of the values of the concatenation  $xz$ . Again, it is easy to verify that SST in Figure 1(b) captures the transformation  $f_1$ .

**Contributions and Challenges.** Our main contributions include the definition of aperiodic streaming string transducers and aperiodic two-way transducers, and the proof of the following key theorem connecting FO and transducers for transformations of infinite strings.

► **Theorem 2.** *Let  $F : \Sigma^\omega \rightarrow \Gamma^\omega$ . Then the following assertions are equivalent:*

1.  $F$  is first-order definable.
2.  $F$  is definable by some aperiodic two-way transducer with star-free look-around.
3.  $F$  is definable by some aperiodic streaming string transducers.

We introduce the notion of transition monoids for automata, 2WST, and SST with the Muller acceptance condition; and recover the classical result proving aperiodicity of a language using the aperiodicity of the transition monoid of its underlying automaton. The equivalence between FOT and 2WST with star-free look-around (Section 4), crucially uses the transition monoid with Muller acceptance, which is necessary to show aperiodicity of the underlying language of the 2WST. On the other hand, while going from aperiodic SST to FOT (Section 5), the main difficulty is the construction of the FOT using the aperiodicity of the SST, and while going from 2WST with star-free look-around to SST (Section 6), the hard part is to establish the aperiodicity of the SST. Due to space limitation, we only provide key definitions and sketches of our results – complete proofs and related supplementary material can be found in longer version of this paper [11].

## 2 Preliminaries

A finite (infinite) string over alphabet  $\Sigma$  is a finite (infinite) sequence of letters from  $\Sigma$ . We denote by  $\epsilon$  the empty string. We write  $\Sigma^*$  for the set of finite strings,  $\Sigma^\omega$  for the set of  $\omega$ -strings over  $\Sigma$ , and  $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$  for the set of finite and  $\omega$ -strings. A language  $L$  over an alphabet  $\Sigma$  is defined as a set of strings, i.e.  $L \subseteq \Sigma^\infty$ .

For a string  $s \in \Sigma^\infty$  we write  $|s|$  for its length; note that  $|s| = \infty$  for an  $\omega$ -string  $s$ . Let  $\text{dom}(s) = \{1, 2, 3, \dots\}$  be the set of positions in  $s$ . For all  $i \in \text{dom}(s)$  we write  $s[i]$  for the  $i$ -th letter of the string  $s$ . For two  $\omega$ -strings  $s, s' \in \Sigma^\omega$ , we define the distance  $d(s, s')$  as  $\frac{1}{2^j}$  where  $j = \min\{k \mid s[k] \neq s'[k]\}$ . We say that a string  $s \in \Sigma^\omega$  is the limit of a sequence  $s_1, s_2, \dots$  of  $\omega$ -strings  $s_i \in \Sigma^\omega$  if for every  $\epsilon > 0$ , there is an index  $n_\epsilon \in \mathbb{N}$  such that for all  $i \geq n_\epsilon$ , we have that  $d(s, s_i) \leq \epsilon$ . Such a limit, if exists, is unique and is denoted as  $s = \lim_{i \rightarrow \infty} s_i$ . For example,  $b^\omega = \lim_{i \rightarrow \infty} b^i c^\omega$ .

### 2.1 Aperiodic Monoids for $\omega$ -String Languages

A monoid  $\mathcal{M}$  is an algebraic structure  $(M, \cdot, e)$  with a non-empty set  $M$ , a binary operation  $\cdot$ , and an identity element  $e \in M$  such that for all  $x, y, z \in M$  we have that  $(x \cdot (y \cdot z)) = ((x \cdot y) \cdot z)$ , and  $x \cdot e = e \cdot x$  for all  $x \in M$ . We say that a monoid  $(M, \cdot, e)$  is *finite* if the set  $M$  is finite. A monoid that we will use in this paper is the *free* monoid,  $(\Sigma^*, \cdot, \epsilon)$ , which has a set of finite strings over some alphabet  $\Sigma$  with the empty string  $\epsilon$  as the identity.

We define the notion of acceptance of a language via monoids. A morphism (or homomorphism) between two monoids  $\mathcal{M} = (M, \cdot, e)$  and  $\mathcal{M}' = (M', \cdot, e')$  is a mapping  $h : M \rightarrow M'$  such that  $h(e) = e'$  and  $h(x \cdot y) = h(x) \cdot h(y)$ . Let  $h : \Sigma^* \rightarrow M$  be a morphism from free monoid  $(\Sigma^*, \cdot, \epsilon)$  to a finite monoid  $(M, \cdot, e)$ . Two strings  $u, v \in \Sigma^*$  are said to be similar with respect to  $h$  denoted  $u \sim_h v$ , if for some  $n \in \mathbb{N} \cup \{\infty\}$ , we can factorize  $u, v$  as  $u = u_1 u_2 \dots u_n$  and  $v = v_1 v_2 \dots v_n$  with  $u_i, v_i \in \Sigma^+$  and  $h(u_i) = h(v_i)$  for all  $i$ . Two  $\omega$ -strings are  $h$ -similar if we can find factorizations  $u_1 u_2 \dots$  and  $v_1 v_2 \dots$  such that  $h(u_i) = h(v_i)$  for all  $i$ . Let  $\cong$  be the transitive closure of  $\sim_h$ .  $\cong$  is an equivalence relation. Note that since

$M$  is finite, the equivalence relation  $\cong$  is of finite index. For  $w \in \Sigma^\infty$  we define  $[w]_h$  as the set  $\{u \mid u \cong w\}$ . We say that a morphism  $h$  accepts a language  $L \subseteq \Sigma^\infty$  if  $w \in L$  implies  $[w]_h \subseteq L$  for all  $w \in \Sigma^\infty$ .

We say that a monoid  $(M, \cdot, e)$  is *aperiodic* [21] if there exists  $n \in \mathbb{N}$  such that for all  $x \in M$ ,  $x^n = x^{n+1}$ . Note that for finite monoids, it is equivalent to require that for all  $x \in M$ , there exists  $n \in \mathbb{N}$  such that  $x^n = x^{n+1}$ . A language  $L \subseteq \Sigma^\infty$  is said to be aperiodic iff it is recognized by some morphism to a finite and aperiodic monoid [11].

## 2.2 First-Order Logic for $\omega$ -String Languages

A string  $s \in \Sigma^\omega$  can be represented as a relational structure  $\Xi_s = (\text{dom}(s), \preceq^s, (L_a^s)_{a \in \Sigma})$ , called the string model of  $s$ , where  $\text{dom}(s) = \{1, 2, \dots\}$  is the set of positions in  $s$ ,  $\preceq^s$  is a binary relation over the positions in  $s$  characterizing the natural order, i.e.  $(x, y) \in \preceq^s$  if  $x \leq y$ ;  $L_a^s$ , for all  $a \in \Sigma$ , are the unary predicates that hold for the positions in  $s$  labeled with the letter  $a$ , i.e.,  $L_a^s(i)$  iff  $s[i] = a$ , for all  $i \in \text{dom}(s)$ . When it is clear from context we will drop the superscript  $s$  from the relations  $\preceq^s$  and  $L_a^s$ .

Properties of string models over the alphabet  $\Sigma$  can be formalized by first-order logic denoted by  $\text{FO}(\Sigma)$ . Formulas of  $\text{FO}(\Sigma)$  are built up from variables  $x, y, \dots$  ranging over positions of string models along with *atomic formulae* of the form  $x=y$ ,  $x \preceq y$ , and  $L_a(x)$  for all  $a \in \Sigma$  where formula  $x=y$  states that variables  $x$  and  $y$  point to the same position, the formula  $x \preceq y$  states that position corresponding to variable  $x$  is not larger than that of  $y$ , and the formula  $L_a(x)$  states that position  $x$  has the label  $a \in \Sigma$ . Atomic formulae are connected with *propositional connectives*  $\neg, \wedge, \vee, \rightarrow$ , and *quantifiers*  $\forall$  and  $\exists$  that range over node variables and we use usual semantics for them. We say that a variable is *free* in a formula if it does not occur in the scope of some quantifier. A *sentence* is a formula with no free variables. We write  $\phi(x_1, x_2, \dots, x_k)$  to denote that at most the variables  $x_1, \dots, x_k$  occur free in  $\phi$ . For a string  $s \in \Sigma^*$  and for positions  $n_1, n_2, \dots, n_k \in \text{dom}(s)$  we say that  $s$  with valuation  $\nu = (n_1, n_2, \dots, n_k)$  satisfies the formula  $\phi(x_1, x_2, \dots, x_k)$  and we write  $(s, \nu) \models \phi(x_1, x_2, \dots, x_k)$  or  $s \models \phi(n_1, n_2, \dots, n_k)$  if formula  $\phi$  with  $n_i$  as the interpretation of  $x_i$  is satisfied in the string model  $\Xi_s$ . The language defined by an FO sentence  $\phi$  is  $L(\phi) \stackrel{\text{def}}{=} \{s \in \Sigma^\omega : \Xi_s \models \phi\}$ . We say that a language  $L$  is FO-definable if there is an FO sentence  $\phi$  such that  $L = L(\phi)$ . The following is a well known result.

► **Theorem 3** ([18, 20]). *A language  $L \subseteq \Sigma^*$  is FO-definable iff it is aperiodic.*

## 2.3 Aperiodic Muller Automata for $\omega$ -String Languages

A deterministic Muller automaton (DMA) is a tuple  $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$  where  $Q$  is a finite set of states,  $q_0 \in Q$  is the initial state,  $\Sigma$  is an input alphabet,  $\delta : Q \times \Sigma \rightarrow Q$  is a transition function, and  $F \subseteq 2^Q$  are the accepting (Muller) sets. For states  $q, q' \in Q$  and letter  $a \in \Sigma$  we say that  $(q, a, q')$  is a transition of the automaton  $\mathcal{A}$  if  $\delta(q, a) = q'$  and we write  $q \xrightarrow{a} q'$ . We say that there is a run of  $\mathcal{A}$  over a finite string  $s = a_1 a_2 \dots a_n \in \Sigma^*$  from state  $p$  to state  $q$  if there is a finite sequence of transitions  $\langle (p_0, a_1, p_1), (p_1, a_2, p_2), \dots, (p_{n-1}, a_n, p_n) \rangle \in (Q \times \Sigma \times Q)^*$  with  $p = p_0$  and  $q = p_n$ . We write  $L_{p,q}$  for the set of finite strings  $w$  such that there is a run of  $\mathcal{A}$  over  $w$  from  $p$  to  $q$ . We say that there is a run of  $\mathcal{A}$  over an  $\omega$ -string  $s = a_1 a_2 \dots \in \Sigma^\omega$  if there is a sequence of transitions  $\langle (q_0, a_1, q_1), (q_1, a_2, q_2), \dots \rangle \in (Q \times \Sigma \times Q)^\omega$ . For an infinite run  $r$ , we denote by  $\Omega(r)$  the set of states that occur infinitely often in  $r$ . We say that an  $\omega$ -string  $w$  is accepted by a Muller automaton  $\mathcal{A}$  if the run of  $\mathcal{A}$  on  $w$  is such that  $\Omega(r) \in F$  and we write  $L(\mathcal{A})$  for the set of all  $\omega$ -strings accepted by  $\mathcal{A}$ .

A Muller automaton  $\mathcal{A}$  is *aperiodic* iff there exists some  $m \geq 1$  s.t.  $u^m \in L_{p,q}$  iff  $u^{m+1} \in L_{p,q}$

for all  $u \in \Sigma^*$  and  $p, q \in Q$ . Another equivalent way to define aperiodicity is using the transition monoid, which, to the best of our knowledge, has not been defined in the literature for Muller automata. Given a DMA  $\mathcal{A}=(Q, q_0, \Sigma, \Delta, \{F_1, \dots, F_n\})$ , we define the transition monoid  $\mathcal{M}_{\mathcal{A}}=(M_{\mathcal{A}}, \times, \mathbf{1})$  of  $\mathcal{A}$  as follows:  $M_{\mathcal{A}}$  is a set of  $|Q| \times |Q|$  square matrices over  $(\{0, 1\} \cup 2^Q)^n \cup \{\perp\}$ . Matrix multiplication  $\times$  is defined for matrices in  $M_{\mathcal{A}}$  with identity element  $\mathbf{1} \in M_{\mathcal{A}}$ , where  $\mathbf{1}$  is the matrix whose diagonal entries are  $(\emptyset, \emptyset, \dots, \emptyset)$  and non-diagonal entries are all  $\perp$ 's. Formally,  $M_{\mathcal{A}}=\{M_s : s \in \Sigma^*\}$  is defined using matrices  $M_s$  for strings  $s \in \Sigma^*$  s.t.  $M_s[p][q]=\perp$  if there is no run from  $p$  to  $q$  over  $s$  in  $\mathcal{A}$ . Otherwise, let  $P$  be the set of states (excluding  $p$  and  $q$ ) witnessed in the unique run from  $p$  to  $q$ . Then  $M_s[p][q] = (x_1, \dots, x_n) \in (\{0, 1\} \cup 2^Q)^n$  where (1)  $x_i = 0$  iff  $\exists t \in P \cup \{p, q\}, t \notin F_i$ ; (2)  $x_i = 1$  iff  $P \cup \{p, q\} = F_i$ , and (3)  $x_i = P \cup \{p, q\}$  iff  $P \cup \{p, q\} \subset F_i$ . It is easy to see that  $M_\epsilon = \mathbf{1}$ , since  $\epsilon$  takes a state to itself and nowhere else. The operator  $\times$  is simply matrix multiplication for matrices in  $M_{\mathcal{A}}$ , however we need to define addition  $\oplus$  and multiplication  $\odot$  for elements  $(\{0, 1\} \cup 2^Q)^n \cup \{\perp\}$  of the matrices. We have  $\alpha_1 \odot \alpha_2 = \perp$  if  $\alpha_1 = \perp$  or  $\alpha_2 = \perp$ , and if  $\alpha_1 = (x_1, \dots, x_n)$  and  $\alpha_2 = (y_1, \dots, y_n)$  then  $\alpha_1 \odot \alpha_2 = (z_1, \dots, z_n)$  s.t.:

$$z_i = \begin{cases} 0 & \text{if } x_i = 0 \text{ or } y_i = 0 \\ 1 & \text{if } (x_i = y_i = 1) \text{ or if } (x_i, y_i \subset F_i \text{ and } x_i \cup y_i = F_i) \\ 1 & \text{if } (x_i = 1 \text{ and } y_i \subset F_i) \text{ or } (y_i = 1 \text{ and } x_i \subset F_i) \\ x_i \cup y_i & \text{if } x_i, y_i \subset F_i \text{ and } x_i \cup y_i \subset F_i \end{cases} \quad (\star)$$

Due to determinism, we have that for every matrix  $M_s$  and every state  $p$  there is at most one state  $q$  such that  $M_s[p][q] \neq \perp$  and hence the only addition rule we need to introduce is  $\alpha \oplus \perp = \perp \oplus \alpha = \alpha$ . It is easy to see that  $(M_{\mathcal{A}}, \times, \mathbf{1})$  is a monoid (a proof is deferred to the [11]). It is straightforward to see that a Muller automaton is aperiodic if and only if its transition monoid is aperiodic. [11] gives a proof showing that a language  $L \subseteq \Sigma^\omega$  is aperiodic iff there is an aperiodic DMA accepting it.

### 3 Aperiodic Transformations

In this section we formally introduce three models to express FO-transformations, and prepare the machinery required to prove their expressive equivalence in the rest of the paper.

#### 3.1 First-Order Logic Definable Transformations

Courcelle [10] initiated the study of structure transformations using MSO logic. His main idea was to define a transformation  $(w, w') \in R$  by defining the string model of  $w'$  using a finite number of copies of positions of the string model of  $w$ . The existence of positions, various edges, and position labels are then given as MSO( $\Sigma$ ) formulas. We study a restriction of his formalism to use first-order logic to express string transformations.

► **Definition 4.** An *FO string transducer* is a tuple  $T=(\Sigma, \Gamma, \phi_{\text{dom}}, C, \phi_{\text{pos}}, \phi_{\preceq})$  where:

- $\Sigma$  and  $\Gamma$  are finite input and output alphabets;
- $\phi_{\text{dom}}$  is a closed FO( $\Sigma$ ) formula characterizing the domain of the transformation;
- $C=\{1, 2, \dots, n\}$  is a finite index set;
- $\phi_{\text{pos}}=\{\phi_\gamma^c(x) : c \in C \text{ and } \gamma \in \Gamma\}$  is a set of FO( $\Sigma$ ) formulae with a free variable  $x$ ;
- $\phi_{\preceq}=\{\phi_{\preceq}^{c,d}(x, y) : c, d \in C\}$  is a set of FO( $\Sigma$ ) formulae with two free variables  $x$  and  $y$ .

The transformation  $\llbracket T \rrbracket$  defined by  $T$  is as follows. A string  $s$  with  $\Xi_s = (\text{dom}(s), \preceq, (L_a)_{a \in \Sigma})$  is in the domain of  $\llbracket T \rrbracket$  if  $s \models \phi_{\text{dom}}$  and the output string  $w$  with structure

$M = (D, \preceq^M, (L_\gamma^M)_{\gamma \in \Gamma})$  is such that

- $D = \{v^c : v \in \text{dom}(s), c \in C \text{ and } \phi^c(v)\}$  is the set of positions where

$$\phi^c(v) \stackrel{\text{def}}{=} \bigvee_{\gamma \in \Gamma} \phi_\gamma^c(v);$$

- $\preceq^M \subseteq D \times D$  is the ordering relation between positions and it is such that for  $v, u \in \text{dom}(s)$  and  $c, d \in C$  we have that  $v^c \preceq^M u^d$  if  $w \models \phi_{\preceq}^{c,d}(v, u)$ ; and
- for all  $v^c \in D$  we have that  $L_\gamma^M(v^c)$  iff  $\phi_\gamma^c(v)$ .

Observe that the output is unique and therefore FO transducers implement functions. A string  $s \in \Sigma^\omega$  can be represented by its string-graph with  $\text{dom}(s) = \{i \in \mathbb{N}\}$ ,  $\preceq = \{(i, j) \mid i \leq j\}$  and  $L_a(i)$  iff  $s[i] = a$  for all  $i$ . From now on, we denote the string-graph of  $s$  as  $s$  only. We say that an FO transducer is a *string-to-string* transducer if its domain is restricted to string graphs and the output is also a string graph. We say that a string-to-string transformation is FO-definable if there exists an FO transducer implementing the transformation. We write FOT for the set of FO-definable string-to-string  $\omega$ -transformations.

► **Example 5.** Figure 1(c) shows a transformation for an FOT that implements the transformation  $f_1 : \Sigma^* \# \{a, b\}^\omega \rightarrow \Sigma^\omega$ , where  $\Sigma = \{a, b, \#\}$ , by replacing every maximal  $\#$  free string  $u$  with  $\bar{u}u$ . Let  $\text{is\_string}_\#$  be an FO formula that defines a string that contains a  $\#$ , and let  $\text{reach}_\#(x)$  be an FO formula that is true at a position which has a  $\#$  at a later position. To define the FOT formally, we have  $\phi_{\text{dom}} = \text{is\_string}_\#$ ,  $\phi_\gamma^1(x) = \phi_\gamma^2(x) = L_\gamma(x) \wedge \neg L_\#(x) \wedge \text{reach}_\#(x)$ , since we only keep the non  $\#$  symbols that can “reach” a  $\#$  in the input string in the first two copies.  $\phi_\gamma^3(x) = L_\#(x) \vee (\neg L_\#(x) \wedge \neg \text{reach}_\#(x))$ , since we only keep the  $\#$ 's, and the infinite suffix from where there are no  $\#$ 's. The full list of formulae  $\phi^{i,j}$  can be seen in [11].

### 3.2 Two-way Transducers (2WST)

A 2WST is a tuple  $T = (Q, \Sigma, \Gamma, q_0, \delta, F)$  where  $\Sigma, \Gamma$  are respectively the input and output alphabet,  $q_0$  is the initial state,  $\delta$  is the transition function and  $F \subseteq 2^Q$  is the acceptance set. The transition function is given by  $\delta : Q \times \Sigma \rightarrow Q \times \Gamma^* \times \{1, 0, -1\}$ . A configuration of the 2WST is a pair  $(q, i)$  where  $q \in Q$  and  $i \in \mathbb{N}$  is the current position of the input string. A run  $r$  of a 2WST on a string  $s \in \Sigma^\omega$  is a sequence of transitions  $(q_0, i_0=0) \xrightarrow{a_1/c_1.dir} (q_1, i_1) \xrightarrow{a_2/c_2.dir} (q_2, i_2) \cdots$  where  $a_i \in \Sigma$  is the input letter read and  $c_i \in \Gamma^*$  is the output string produced during a transition and  $i_j$ s are the positions updated during a transition for all  $j \in \text{dom}(s)$ .  $dir$  is the direction,  $\{1, 0, -1\}$ . W.l.o.g. we can consider the outputs to be over  $\Gamma \cup \{\epsilon\}$ . The output  $out(r)$  of a run  $r$  is simply a concatenation of the individual outputs, i.e.  $c_1 c_2 \cdots \in \Gamma^\infty$ . We say that the transducer reads the whole string  $s$  when  $\sup \{i_n \mid 0 \leq n < |r|\} = \infty$ . The output of  $s$ , denoted  $T(s)$  is defined as  $out(r)$  only if  $\Omega(r) \in F$  and  $r$  reads the whole string  $s$ . We write  $\llbracket T \rrbracket$  for the transformation captured by  $T$ .

**Transition Monoid.** The transition monoid of a 2WST  $T = (Q, \Sigma, \Gamma, q_0, \delta, \{F_1, \dots, F_n\})$  is the transition monoid of its underlying automaton. However, since the 2WST can read their input in both directions, the transition monoid definition must allow for reading the string starting from left side and leaving at the left (left-left) and similar other behaviors (left-right, right-left and right-right). Following [9], we define the behaviors  $\mathcal{B}_{xy}(w)$  of a string  $w$  for  $x, y \in \{\ell, r\}$ .  $\mathcal{B}_{\ell r}(w)$  is a set consisting of pairs  $(p, q)$  of states such that starting in state  $p$  in the left side of  $w$  the transducer leaves  $w$  in right side in state  $q$ . In the example in figure 1(a), we have  $\mathcal{B}_{\ell r}(ab\#) = \{(t, t), (p, t), (q, t)\}$  and  $\mathcal{B}_{rr}(ab\#) = \{(q, t), (t, t), (p, q)\}$ . Two words  $w_1, w_2$  are “equivalent” if their left-left, left-right, right-left and right-right behaviors are same.

That is,  $\mathcal{B}_{xy}(w_1) = \mathcal{B}_{xy}(w_2)$  for  $x, y \in \{\ell, r\}$ . The transition monoid of  $T$  is the conjunction of the 4 behaviors, which also keeps track, in addition, the set of states witnessed in the run, as shown for the deterministic Muller automata earlier. For each string  $w \in \Sigma^*$ ,  $x, y \in \{\ell, r\}$ , and states  $p, q$ , the entries of the matrix  $M_u^{xy}[p][q]$  are of the form  $\perp$ , if there is no run from  $p$  to  $q$  on word  $u$ , starting from the side  $x$  of  $u$  and leaving it in side  $y$ , and is  $(x_1, \dots, x_n)$  otherwise, where  $x_i$  is defined exactly as in section 2.3. For equivalent words  $u_1, u_2$ , we have  $M_{u_1}^{xy}[p][q] = M_{u_2}^{xy}[p][q]$  for all  $x, y \in \{\ell, r\}$  and states  $p, q$ . Addition and multiplication of matrices are defined as in the case of Muller automata. See [11] for more details. Note that behavioral composition is quite complex, due to left-right movements. In particular, it can be seen from the example that  $\mathcal{B}_{\ell r}(ab\#a\#) = \mathcal{B}_{\ell r}(ab\#)\mathcal{B}_{\ell\ell}(a\#)\mathcal{B}_{rr}(ab\#)\mathcal{B}_{\ell r}(a\#)$ . Since we assume that the 2WST  $T$  is deterministic and completely reads the input string  $\alpha \in \Sigma^\omega$ , we can find a unique factorization  $\alpha = [\alpha_0 \dots \alpha_{p_1}][\alpha_{p_1+1} \dots \alpha_{p_2}] \dots$  such that the run of  $\mathcal{A}$  on each  $\alpha$ -block progresses from left to right, and each  $\alpha$ -block will be processed completely. That is, one can find a unique sequence of states  $q_{p_1}, q_{p_2}, \dots$  such that the 2WST starting in initial state  $q_0$  at the left of the block  $\alpha_0 \dots \alpha_{p_1}$  leaves it at the right in state  $q_{p_1}$ , starts the next block  $\alpha_{p_1+1} \dots \alpha_{p_2}$  from the left in state  $q_{p_1}$  and leaves it at the right in state  $q_{p_2}$  and so on.

We consider the languages  $L_{pq}^{xy}$  for  $x, y \in \{\ell, r\}$ , where  $\ell, r$  respectively stand for left and right.  $L_{pq}^{\ell\ell}$  stands for all strings  $w$  such that, starting at state  $p$  at the left of  $w$ , one leaves the left of  $w$  in state  $q$ . Similarly,  $L_{pq}^{r\ell}$  stands for all strings  $w$  such that starting at the right of  $w$  in state  $p$ , one leaves the left of  $w$  in state  $q$ . In figure 1(a), note that starting on the right of  $ab\#$  in state  $t$ , we leave it on the right in state  $t$ , while we leave it on the left in state  $p$ . So  $ab\# \in L_{tt}^{rr}, L_{tp}^{r\ell}$ . Also,  $ab\# \in L_{pq}^{rr}$ .

A 2WST is said to be *aperiodic* iff for all strings  $u \in \Sigma^*$ , all states  $p, q$  and  $x, y \in \{\ell, r\}$ , there exists some  $m \geq 1$  such that  $u^m \in L_{pq}^{xy}$  iff  $u^{m+1} \in L_{pq}^{xy}$ .

**Star-Free Lookaround.** We wish to introduce aperiodic 2WST that are capable of capturing FO-definable transformations. However, as we discussed earlier (see page 3 in the paragraph on two-way transducers) 2WST without look-ahead are strictly less expressive than MSO transducers. To remedy this we study aperiodic 2WSTs enriched with star-free look-ahead (star-free look-back can be assumed for free).

An aperiodic 2WST with star-free look-around ( $2WST_{sf}$ ) is a tuple  $(T, A, B)$  where  $A$  is an aperiodic Muller look-ahead automaton and  $B$  is an aperiodic look-behind automaton, resp., and  $T = (\Sigma, \Gamma, Q, q_0, \delta, F)$  is an aperiodic 2WST as defined earlier except that the transition function  $\delta : Q \times Q_B \times \Sigma \times Q_A \rightarrow Q \times \Gamma \times \{-1, 0, +1\}$  may consult look-ahead and look-behind automata to make its decisions. Let  $s \in \Sigma^\omega$  be an input string, and  $L(A, p)$  be the set of infinite strings accepted by  $A$  starting in state  $p$ . Similarly, let  $L(B, r)$  be the set of finite strings accepted by  $B$  starting in state  $r$ . We assume that  $2WST_{sf}$  are *deterministic* i.e. for every string  $s \in \Sigma^\omega$  and every input position  $i \leq |s|$ , there is exactly one state  $p \in Q_A$  and one state  $r \in Q_B$  such that  $s(i)s(i+1) \dots \in L(A, p)$  and  $s(0)s(1) \dots s(i-1) \in L(B, r)$ . If the current configuration is  $(q, i)$  and  $\delta(q, r, s(i), p) = (q', z, d)$  is a transition, such that the string  $s(i)s(i+1) \dots \in L(A, p)$  and  $s(0)s(1) \dots s(i-1) \in L(B, r)$ , then  $2WST_{sf}$  writes  $z \in \Gamma$  on the output tape and updates its configuration to  $(q', i+d)$ . Figure 1(a) shows a 2WST with star-free look-ahead  $\text{reach}_\#(x)$  capturing the transformation  $f_1$  (details in [11]).

### 3.3 Streaming $\omega$ -String Transducers (SST)

Streaming string transducers(SSTs) manipulate a finite set of string variables to compute their output. In this section we introduce aperiodic SSTs for infinite strings. Let  $\mathcal{X}$



be a finite set of variables and  $\Gamma$  be a finite alphabet. A substitution  $\sigma$  is defined as a mapping  $\sigma : \mathcal{X} \rightarrow (\Gamma \cup \mathcal{X})^*$ . A valuation is defined as a substitution  $\sigma : \mathcal{X} \rightarrow \Gamma^*$ . Let  $\mathcal{S}_{\mathcal{X},\Gamma}$  be the set of all substitutions  $[\mathcal{X} \rightarrow (\Gamma \cup \mathcal{X})^*]$ . Any substitution  $\sigma$  can be extended to  $\hat{\sigma} : (\Gamma \cup \mathcal{X})^* \rightarrow (\Gamma \cup \mathcal{X})^*$  in a straightforward manner. The composition  $\sigma_1\sigma_2$  of two substitutions  $\sigma_1$  and  $\sigma_2$  is defined as the standard function composition  $\hat{\sigma}_1\sigma_2$ , i.e.  $\hat{\sigma}_1\sigma_2(x) = \hat{\sigma}_1(\sigma_2(x))$  for all  $x \in \mathcal{X}$ . We say that a string  $u \in (\Gamma \cup \mathcal{X})^*$  is *copyless* (or linear) if each  $x \in \mathcal{X}$  occurs at most once in  $u$ . A substitution  $\sigma$  is copyless if  $\hat{\sigma}(u)$  is copyless, for all linear  $u \in (\Gamma \cup \mathcal{X})^*$ .

► **Definition 6.** A *streaming  $\omega$ -string transducer* (SST) is a tuple  $T = (\Sigma, \Gamma, Q, q_0, \delta, \mathcal{X}, \rho, F)$

- $\Sigma$  and  $\Gamma$  are finite input and output alphabets;
- $Q$  is a finite set of states with initial state  $q_0$ ;
- $\delta : Q \times \Sigma \rightarrow Q$  is a transition function and  $\mathcal{X}$  is a finite set of variables;
- $\rho : (Q \times \Sigma) \rightarrow \mathcal{S}_{\mathcal{X},\Gamma}$  is a variable update function to copyless substitutions such that any variable  $x$  occurs at most once on the right hand side of a simultaneous substitution;
- $F : 2^Q \rightarrow \mathcal{X}^*$  is an output function such that for all  $P \in \text{dom}(F)$  the string  $F(P)$  is copyless of form  $x_1 \dots x_n$ , and for  $q, q' \in P$  and  $a \in \Sigma$  s.t.  $q' = \delta(q, a)$  we have
  - $\rho(q, a)(x_i) = x_i$  for all  $i < n$  and  $\rho(q, a)(x_n) = x_n u$  for some  $u \in (\Gamma \cup \mathcal{X})^*$ .

The concept of a run of an SST is defined in an analogous manner to that of a Muller automaton. The sequence  $\langle \sigma_{r,i} \rangle_{0 \leq i \leq |r|}$  of substitutions induced by a run  $r = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \dots$  is defined inductively as the following:  $\sigma_{r,i} = \sigma_{r,i-1} \rho(q_{i-1}, a_i)$  for  $0 < i \leq |r|$  and  $\sigma_{r,0} = x \in X \mapsto \varepsilon$ . The output  $T(r)$  of an infinite run  $r$  of  $T$  is defined only if  $F(r)$  is defined and equals  $T(r) \stackrel{\text{def}}{=} \lim_{i \rightarrow \infty} \langle \sigma_{r,i}(F(r)) \rangle$ , when the limit exists. If not, we pad  $\perp^\omega$  to the obtained finite string to get  $\lim_{i \rightarrow \infty} \langle \sigma_{r,i}(F(r)) \perp^\omega \rangle$  as the infinite output string.

The assumptions on the output function  $F$  in the definition of an SST ensure that this limit exists whenever  $F(r)$  is defined. Indeed, when a run  $r$  reaches a point from where it visits only states in  $P$ , these assumptions enforce the successive valuations of  $F(P)$  to be an increasing sequence of strings by the prefix relation. The padding by unique letter  $\perp$  ensures that the output is always an  $\omega$ -string. The output  $T(s)$  of a string  $s$  is then defined as the output  $T(r)$  of its unique run  $r$ . The transformation  $\llbracket T \rrbracket$  defined by an SST  $T$  is the partial function  $\{(s, T(s)) : T(s) \text{ is defined}\}$ . See [11] for an example. We remark that for every SST  $T = (\Sigma, \Gamma, Q, q_0, \delta, \mathcal{X}, \rho, F)$ , its domain is always an  $\omega$ -regular language defined by the Muller automaton  $(\Sigma, Q, q_0, \delta, \text{dom}(F))$ , which can be constructed in linear time. However, the range of an SST may not be  $\omega$ -regular. For instance, the range of the SST-definable transformation  $a^n \#^\omega \mapsto a^n b^n \#^\omega$  ( $n \geq 0$ ) is not  $\omega$ -regular.

**Aperiodic Streaming String Transducers.** We define the notion of aperiodic SSTs by introducing an appropriate notion of transition monoid for transducers. The transition monoid of an SST  $T$  is based on the effect of a string  $s$  on the states as well as on the variables. The effect on variables is characterized by, what we call, flow information that is given as a relation that describes the number of copies of the content of a given variable that contribute to another variable after reading a string  $s$ .

Let  $T = (\Sigma, \Gamma, Q, q_0, \delta, \mathcal{X}, \rho, F)$  be an SST. Let  $s$  be a string in  $\Sigma^*$  and suppose that there exists a run  $r$  of  $T$  on  $s$ . Recall that this run induces a substitution  $\sigma_r$  that maps each variable  $X \in \mathcal{X}$  to a string  $u \in (\Gamma \cup \mathcal{X})^*$ . For string variables  $X, Y \in \mathcal{X}$ , states  $p, q \in Q$ , and  $n \in \mathbb{N}$  we say that  $n$  copies of  $Y$  flow to  $X$  from  $p$  to  $q$  if there exists a run  $r$  on  $s \in \Sigma^*$  from  $p$  to  $q$ , and  $Y$  occurs  $n$  times in  $\sigma_r(X)$ . We extend the notion of transition monoid for the Muller automata as defined in Section 2 for the transition monoid for SSTs to equip it with variables. Formally, the transition monoid  $\mathcal{M}_T = (M_T, \times, \mathbf{1})$  of an

## 12:10 FO-Definable Transformations of Infinite Strings

SST  $T = (\Sigma, \Gamma, Q, q_0, \delta, \mathcal{X}, \rho, \{F_1, \dots, F_n\})$  is such that  $M_T$  is a set of  $|Q \times \mathcal{X}| \times |Q \times \mathcal{X}|$  square matrices over  $(\mathbb{N} \times (\{0, 1\} \cup 2^Q)^n) \cup \{\perp\}$  along with matrix multiplication  $\times$  defined for matrices in  $M_T$  and identity element  $\mathbf{1} \in M_T$  is the matrix whose diagonal entries are  $(1, (\emptyset, \emptyset, \dots, \emptyset))$  and non-diagonal entries are all  $\perp$ 's. Formally  $M_T = \{M_s : s \in \Sigma^*\}$  is defined using matrices  $M_s$  for strings  $s \in \Sigma^*$  s.t.  $M_s[(p, X)][(q, Y)] = \perp$  if there is no run from state  $p$  to state  $q$  over  $s$  in  $T$ , otherwise  $M_s[(p, X)][(q, Y)] = (k, (x_1, \dots, x_n)) \in (\mathbb{N} \times (\{0, 1\} \cup 2^Q)^n)$  where  $x_i$  is defined exactly as in section 2.3, and  $k$  copies of variable  $X$  flow to variable  $Y$  from state  $p$  to state  $q$  after reading  $s$ .

We write  $(p, X) \rightsquigarrow_\alpha^u (q, Y)$  for  $M_u[(p, X)][(q, Y)] = \alpha$ .

It is easy to see that  $M_\epsilon = \mathbf{1}$ . The operator  $\times$  is simply matrix multiplication for matrices in  $M_T$ , however we need to define addition  $\oplus$  and multiplication  $\odot$  for elements  $(\{0, 1\} \cup 2^Q)^n \cup \{\perp\}$  of the matrices. We have  $\alpha_1 \odot \alpha_2 = \perp$  if  $\alpha_1 = \perp$  or  $\alpha_2 = \perp$ , and if  $\alpha_1 = (k_1, (x_1, \dots, x_n))$  and  $\alpha_2 = (k_2, (y_1, \dots, y_n))$  then  $\alpha_1 \odot \alpha_2 = (k_1 \times k_2, (z_1, \dots, z_n))$  s.t. for all  $1 \leq i \leq n$   $z_i$  are defined as in  $(\star)$  from Section 2.3. Note that due to determinism of the SSTs we have that for every matrix  $M_s$  and every state  $p$  there is at most one state  $q$  such that  $M_s[p][q] \neq \perp$  and hence the only addition rules we need to introduce is  $\alpha \oplus \perp = \perp \oplus \alpha = \alpha$ ,  $0 \oplus 0 = 0$ ,  $1 \oplus 1 = 1$  and  $\kappa \oplus \kappa = \kappa$  for  $\kappa \subseteq Q$ . It is easy to see that  $(M_T, \times, \mathbf{1})$  is a monoid and we give a proof in [11]. We say that the transition monoid  $M_T$  of an SST  $T$  is 1-bounded if in all entries  $(j, (x_1, \dots, x_n))$  of the matrices of  $M_T$ ,  $j \leq 1$ . A streaming string transducer is *aperiodic* if its transition monoid is aperiodic.

### 4 FOTs $\equiv$ Aperiodic 2WST<sub>sf</sub>

► **Theorem 7.** *A transformation  $f : \Sigma^\omega \rightarrow \Gamma^\omega$  is FOT-definable if and only if it is definable using an aperiodic two way transducer with star-free look-around.*

**Proof (Sketch).** This proof is in two parts.

■ **Aperiodic 2WST<sub>sf</sub>  $\subseteq$  FOT.** We first show that given an aperiodic 2WST<sub>sf</sub>  $\mathcal{A}$ , we can effectively construct an FOT that captures the same transduction as  $\mathcal{A}$  over infinite words. Let  $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \delta, F)$  be an aperiodic 2WST<sub>sf</sub>, where each transition outputs at most one letter. Note that this is without loss of generality, since we can output any longer string by having some extra states. Given  $\mathcal{A}$ , we construct the FOT  $T = (\Sigma, \Gamma, \phi_{dom}, C, \phi_{pos}, \phi_{\prec})$  that realizes the transduction of  $\mathcal{A}$ . The formula  $\phi_{dom}$  is the conjunction of formulae `is_string` and  $\varphi$  where  $\varphi$  is a FO formula that captures the set of accepted strings of  $\mathcal{A}$  (obtained by proving  $L(\mathcal{A})$  is aperiodic [11]) and `is_string` is a FO formula that specifies that the input graph is a string (see [11]). The copies of the FOT are the states of  $\mathcal{A}$ . For any two positions  $x, y$  of the input string, and any two copies  $q, q'$ , we need to define  $\phi_{\prec}^{q, q'}$ . This is simply describing the behaviour of  $\mathcal{A}$  on the substring from position  $x$  to position  $y$  of the input string  $u$ , assuming at position  $x$ , we are in state  $q$ , and reach state  $q'$  at position  $y$ . The following lemma (proof in [11]) gives an FO formula  $\psi_{q, q'}(x, y)$  describing this.

► **Lemma 8.** *Let  $\mathcal{A}$  be an aperiodic 2WST<sub>sf</sub> with the Muller acceptance condition. Then for all pairs of states  $q, q'$ , there exists an FO formula  $\psi_{q, q'}(x, y)$  such that for all strings  $s \in \Sigma^\omega$  and a pair of positions  $x, y$  of  $s$ ,  $s \models \psi_{q, q'}(x, y)$  iff there is a run from state  $q$  starting at position  $x$  of  $s$  that reaches position  $y$  of  $s$  in state  $q'$ .*

An edge exists between position  $x$  of copy  $q$  and position  $y$  of copy  $q'$  iff the input string  $u \models \psi_{q, q'}(x, y)$ . The formulae  $\phi_q^x(x)$  for each copy  $q$  specifies the output at position  $x$  in state  $q$ . We have to capture that position  $x$  is reached from the initial position in state  $q$ ,



and also the possible outputs produced while in state  $q$  at  $x$ . The transition function  $\delta$  gives us these symbols. The formula  $\bigvee_{\delta(q,a)=(q',dir,\gamma)} L_a(x)$  captures the possible output symbols. To state that we reach  $q$  at position  $x$ , we say  $\exists y[\mathbf{first}(y) \wedge \psi_{q_0,q}(y, x)]$ . The conjunction of these two formulae gives  $\phi_\gamma^q(x)$ . This completes the FOT  $T$ .

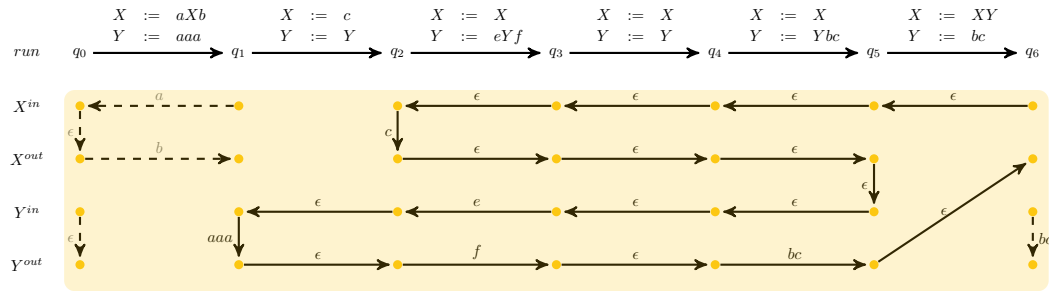
- **FOT  $\subseteq$  Aperiodic 2WST<sub>sf</sub>.** Given an FOT, we show that we can construct an aperiodic 2WST with star-free look-around capturing the same transduction over  $\omega$ -words. For this, we first show that given an FOT, we can construct 2WST enriched with FO instructions that captures the same transduction as the FOT. The idea of the proof follows [13], where one first defines an intermediate model of aperiodic 2WST with FO instructions instead of look-around. Then we show  $\text{FOT} \subseteq \text{2WST}_{fo} \subseteq \text{2WST}_{sf}$ , to complete the proof.

The omitted details can be found in [11]. ◀

## 5 Aperiodic SST $\subseteq$ FOT

► **Lemma 9.** *A transformation is FO-definable if it is aperiodic-SST definable.*

We show that every aperiodic 1-bounded SST definable transformation is definable using FO-transducers. A crucial component in the proof of this lemma is to show that the variable flow in the aperiodic 1-bounded SST is FO-definable ([11]). To construct the FOT, we make use of the output structure for SST. It is an intermediate representation of the output, and the transformation of any input string into its SST-output structure will be shown to be FO-definable. For any SST  $T$  and string  $s \in \text{dom}(T)$ , the SST-output structure of  $s$  is a relational structure  $G_T(s)$  obtained by taking, for each variable  $X \in \mathcal{X}$ , two copies of  $\text{dom}(s)$ , respectively denoted by  $X^{in}$  and  $X^{out}$ . For notational convenience we assume that these structures are labeled on the edges. A pair  $(X, i)$  is *useful* if the content of variable  $X$  before reading  $s[i]$  will be part of the output after reading the whole string  $s$ . This structure satisfies the following *invariants*: for all  $i \in \text{dom}(s)$ , (1) the nodes  $(X^{in}, i)$  and  $(X^{out}, i)$  exist only if  $(X, i)$  is useful, and (2) there is a directed path from  $(X^{in}, i)$  to  $(X^{out}, i)$  whose labels are same as variable  $X$  computed by  $T$  after reading  $s[i]$ .



We define SST-output structures formally in [11], however, the illustration above shows an SST-output structure. We show only the variable updates. Dashed arrows represent variable updates for useless variables, and therefore does not belong the SST-output structure. The path from  $(X^{in}, 6)$  to  $(X^{out}, 6)$  gives the contents of  $X$  ( $ceaaafbc$ ) after 6 steps. We write  $O_T$  for the set of strings appearing in right-hand side of variable updates.

We next show that the transformation that maps an  $\omega$ -string  $s$  into its output structure is FO-definable, whenever the SST is 1-bounded and aperiodic. Using the fact that variable flow is FO-definable, we show that for any two variables  $X, Y$ , we can capture in FO, a path from  $(X^d, i)$  to  $(Y^e, j)$  for  $d, e \in \{in, out\}$  in  $G_T(s)$  and all positions  $i, j$ .

► **Lemma 10.** *Let  $T$  be an **aperiodic, 1-bounded** SST  $T$ . For all  $X, Y \in \mathcal{X}$  and all  $d, d' \in \{in, out\}$ , there exists an  $FO[\Sigma]$ -formula  $path_{X,Y,d,d'}(x, y)$  with two free variables such that for all strings  $s \in \text{dom}(T)$  and all positions  $i, j \in \text{dom}(s)$ ,  $s \models path_{X,Y,d,d'}(i, j)$  iff there exists a path from  $(X^d, i)$  to  $(Y^{d'}, j)$  in  $G_T(s)$ .*

The proof of Lemma 10 is in longer version [11]. As seen in [11] (in Proposition 4) one can write a formula  $\phi_q(x)$  (to capture the state  $q$  reached) and formula  $\psi_P^{Rec}$  (to capture the recurrence of a Muller set  $P$ ) in an accepting run after reading a prefix. For each variable  $X \in \mathcal{X}$ , we have two copies  $X^{in}$  and  $X^{out}$  that serve as the copy set of the FOT. As given by the SST output-structure, for each step  $i$ , state  $q$  and symbol  $a$ , a copy is connected to copies in the previous step based on the updates  $\rho(q, a)$ . The full details of the FOT construction handling the Muller acceptance condition of the SST are in [11].

## 6 Aperiodic $2WST_{sf} \subset$ Aperiodic SST

We show that given an aperiodic  $2WST \mathcal{A} = (\Sigma, \Gamma, Q, q_0, \delta, F)$  with star-free look around over  $\omega$ -words, we can construct an aperiodic SST  $\mathcal{T}$  that realizes the same transformation.

► **Lemma 11.** *For every transformation definable with an aperiodic  $2WST$  with star-free look around, there exists an equivalent aperiodic 1-bounded SST.*

**Proof.** While the idea of the construction is similar to [3], the main challenge is to eliminate the star-free look-around for infinite strings from the SST, preserving aperiodicity. As an intermediate model we introduce streaming  $\omega$ -string transducers with star-free look-around  $SST_{sf}$  that can make transitions based on some star-free property of the input string. We first show that for every aperiodic  $2WST_{sf}$  one can obtain an aperiodic  $SST_{sf}$ , and then prove that the star-free look arounds can be eliminated from the  $SST_{sf}$ .

■ ( $2WST_{sf} \subset SST_{sf}$ ). One of the key observations in the construction is that a  $2WST_{sf}$  can move in either direction, while  $SST_{sf}$  cannot. Since we start with a deterministic  $2WST_{sf}$  that reads the entire input string, it is clear that if a cell  $i$  is visited in a state  $q$ , then we never come back to that cell in the same state. We keep track in each cell  $i$ , with current state  $q$ , the state  $f(q)$  the  $2WST_{sf}$  will be in, when it moves into cell  $i + 1$  for the first time. The  $SST_{sf}$  will move from state  $q$  in cell  $i$  to state  $f(q)$  in cell  $i + 1$ , keeping track of the output produced in the interim time; that is, the output produced between  $q$  in cell  $i$  and  $f(q)$  in cell  $i + 1$  must be produced by the  $SST_{sf}$  during the move. This output is stored in a variable  $X_q$ . The state of the  $SST_{sf}$  at each point of time thus comprises of a pair  $(q, f)$  where  $q$  is the current state of the  $2WST_{sf}$ , and  $f$  is the function which computes the state that  $q$  will evolve into, when moving to the right, the first time. In each cell  $i$ , the state of the SST will coincide with the state the  $2WST_{sf}$  is in, when reading cell  $i$  for the first time. In particular, in the  $SST_{sf}$ , we define  $\delta'((q, f), r, a, p) = (f'(q), f')$  where  $f'(q) = f'(f(t))$  if in the  $2WST_{sf}$  we have  $\delta(q, r, a, p) = (t, \gamma, -1)$ .  $f'(q)$  gives the state in which the  $2WST_{sf}$  will move to the right of the current cell, but clearly this depends on  $f(t)$ , the state in which the  $2WST_{sf}$  will move to the right from the previous cell. The variables of the  $SST_{sf}$  are of the form  $X_q$ , where  $q$  is the current state of the  $SST_{sf}$ . Update of  $X_q$  depends on whether the  $2WST_{sf}$  moves left, right or stays in state  $q$ . For example,  $X_q$  is updated as  $X_t \rho(X_{f(t)})$  if in the  $2WST$ ,  $\delta(q, r, a, p) = (t, \gamma, -1)$  and  $f(t)$  is defined. The definition is recursive, and  $X_t$  handles the output produced from state  $t$  in cell  $i - 1$ . We allow all subsets of  $Q$  as Muller sets of the  $SST_{sf}$ , and keep any checks on these, as part of the look-ahead.

A special variable  $O$  is used to define the output of the Muller sets, by simply updating it as  $O := O\rho(X_q)$  corresponding to the current state  $q$  of the  $2WST_{sf}$  (and  $(q, f)$  is the state of the  $SST_{sf}$ ). The details of the correctness of construction are in [11].

- ( $SST_{sf} \subset SST$ ). An aperiodic SST with star-free lookahead is a tuple  $(T, B, A)$  where  $A = (P_A, \Sigma, \delta_A, P_f)$  is an aperiodic, deterministic Muller automaton called a look-ahead automaton,  $B = (P_B, \Sigma, \delta_B)$  is an aperiodic automaton called the look-behind automaton, and  $T$  is a tuple  $(\Sigma, \Gamma, Q, q_0, \delta, \mathcal{X}, \rho, F)$  where  $\Sigma, \Gamma, Q, q_0, \mathcal{X}, \rho,$  and  $F$  are defined in the same fashion as for  $\omega$ -SSTs, and  $\delta : Q \times P_B \times \Sigma \times P_A \rightarrow Q$  is the transition function. On a string  $a_1a_2 \dots$ , while processing symbol  $a_i$ , we have in the  $SST_{sf}$ ,  $\delta((q, p_B, p_A), a_i) = q'$ , (and the next transition is  $\delta((q', p'_B, p'_A), a_{i+1})$ ) if (i) the prefix  $a_1a_2 \dots a_i \in L(p_A)$ , (ii) the suffix  $a_{i+1}a_{i+2} \dots \in L(p_B)$ , where  $L(p_A)$  ( $L(p_B)$ ) denotes the language accepted starting in state  $p_A$  ( $p_B$ ). We further assume that the look-aheads are mutually exclusive, i.e. for all symbols  $a \in \Sigma$ , all states  $q \in Q$ , and all transitions  $q' = \delta(q, r, a, p)$  and  $q'' = \delta(q, r', a, p')$ , we have that  $L(A_p) \cap L(A_{p'}) = \emptyset$  and  $L(B_r) \cap L(B_{r'}) = \emptyset$ . In [11], we show that for any input string, there is at most one useful, accepting run in the  $SST_{sf}$ , while in Lemma 29 in [11], we show that adding (aperiodic) look-arounds to SST does not increase their expressiveness.

The proof sketch is now complete. ◀

## 7 Conclusion

We extended the notion of aperiodicity from finite string transformations to that on infinite strings. We have shown a way to generalize transition monoids for deterministic Muller automata to streaming string transducers and two-way finite state transducers that capture the FO definable global transformations. An interesting and natural next step is to investigate LTL-definable transformations, their connection with FO-definable transformations, and their practical applications in verification and synthesis.

---

### References

- 1 R. Alur, A. Durand-Gasselín, and A. Trivedi. From monadic second-order definable string transformations to transducers. In *LICS*, pages 458–467, 2013.
- 2 Rajeev Alur and Loris D’Antoni. *Automata, Languages, and Programming: 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, chapter Streaming Tree Transducers, pages 42–53. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- 3 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science, LICS’12*, pages 65–74, Washington, DC, USA, 2012. IEEE Computer Society.
- 4 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, May 2009.
- 5 Rajeev Alur and Pavol Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL’11*, pages 599–610. ACM, 2011.
- 6 Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1–12, Dagstuhl, Germany, 2010.

- Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2010.1.
- 7 J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6(1–6):66–92, 1960.
  - 8 J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Int. Congr. for Logic Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, Stanford, 1962.
  - 9 Olivier Carton and Luc Dartois. Aperiodic two-way transducers and fo-transductions. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, pages 160–174, 2015.
  - 10 B. Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75, 1994.
  - 11 Vrunda Dave, Shankara Narayanan Krishna, and Ashutosh Trivedi. Fo-definable transformations of infinite strings. *CoRR*, abs/1607.04910, 2016. URL: <http://arxiv.org/abs/1607.04910>.
  - 12 V. Diekert and P. Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives*, pages 261–306. Amsterdam University Press, 2008.
  - 13 J. Engelfriet and H. J. Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Logic*, 2:216–254, 2001.
  - 14 Emmanuel Filiot. *Logic and Its Applications: 6th Indian Conference, ICLA 2015, Mumbai, India, January 8-10, 2015. Proceedings*, chapter Logic-Automata Connections for Transformations, pages 30–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
  - 15 Emmanuel Filiot, Shankara Narayanan Krishna, and Ashutosh Trivedi. First-order definable string transformations. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 147–159, 2014.
  - 16 Richard E. Ladner. Application of model theoretic games to discrete linear orders and finite automata. *Information and Control*, 33(4):281–303, 1977.
  - 17 R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Inform. Contr.*, 9:521–530, 1966.
  - 18 R. McNaughton and S. Papert. Counter-free automata. *M.I.T. Research Monograph*, 65, 1971. With an appendix by William Henneman.
  - 19 Dominique Perrin. *Mathematical Foundations of Computer Science 1984: Proceedings, 11th Symposium Praha, Czechoslovakia September 3–7, 1984*, chapter Recent results on automata and infinite words, pages 134–148. Springer Berlin Heidelberg, 1984.
  - 20 M. P. Schuetzenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.
  - 21 H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, 1994.
  - 22 Wolfgang Thomas. Star-free regular sets of  $\omega$ -sequences. *Information and Control*, 42(2):148–156, 1979.
  - 23 Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455. Springer, 1996.
  - 24 Wolfgang Thomas. Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In *Structures in Logic and Computer Science: A Selection of Essays in Honor of A. Ehrenfeucht, Lecture*, pages 118–143. Springer-Verlag, 1997.

# Aperiodicity of Rational Functions Is PSPACE-Complete\*

Emmanuel Filiot<sup>1</sup>, Olivier Gauwin<sup>2</sup>, and Nathan Lhote<sup>2</sup>

- 1 Université Libre de Bruxelles, Belgium
- 2 Université de Bordeaux, LaBRI, CNRS, France
- 3 Université Libre de Bruxelles, Belgium; and  
Université de Bordeaux, LaBRI, CNRS, France

---

## Abstract

It is known that a language of finite words is definable in monadic second-order logic – MSO – (resp. first-order logic – FO –) iff it is recognized by some finite automaton (resp. some aperiodic finite automaton). Deciding whether an automaton  $A$  is equivalent to an aperiodic one is known to be PSPACE-complete. This problem has an important application in logic: it allows one to decide whether a given MSO formula is equivalent to some FO formula. In this paper, we address the aperiodicity problem for functions from finite words to finite words (transductions), defined by finite transducers, or equivalently by bimachines, a transducer model studied by Schützenberger and Reutenauer. Precisely, we show that the problem of deciding whether a given bimachine is equivalent to some aperiodic one is PSPACE-complete.

**1998 ACM Subject Classification** F.4.3 Mathematical Logic and Formal Languages

**Keywords and phrases** Rational word transductions, decision problem, aperiodicity, bimachines

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.13

## 1 Introduction

### Rational languages and the aperiodicity problem

The theory of rational languages (of finite words) is robust, due to many characterizations coming from different domains, such as computation, logic and algebra. For instance, it is well-known that a language is rational iff it is recognized by some finite automaton, iff it is definable in monadic second-order logic with one successor (MSO), iff its right syntactic congruence (also known as Myhill-Nerode congruence) has finite index. The latter algebraic characterization is closely related to the existence of a unique minimal deterministic automaton for every rational language, the states of which are the classes of the right syntactic congruence. Connections between computation, logic and algebra have been established for subclasses of rational languages. Perhaps the most important example, based on seminal works by Schützenberger [21], McNaughton and Papert [16], is the class of aperiodic languages, characterized by *aperiodic* automata, *first-order* logic (FO), and *aperiodic* right syntactic congruences. See also [23] for other classes.

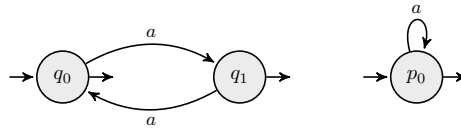
Thanks to the (effective) logic-automata connections, results in logic can be obtained from results in automata, which are well-suited for algorithmic analysis. An important example

---

\* This work was partially supported by the *ExStream* project (ANR-13-JS02-0010), the ARC project *Transform* (Federation Wallonia-Brussels) and the Belgian FNRS CDR project *Flare*. Emmanuel Filiot is research associate at F.R.S.-FNRS.



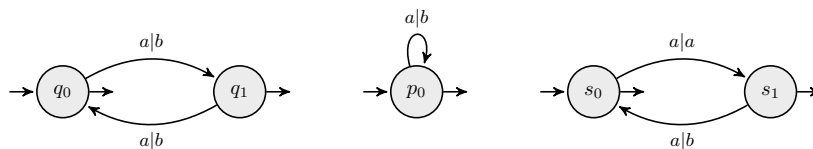
which motivates this paper is the FO in MSO definability problem: given an MSO formula, is it equivalent to some FO formula, over finite strings? In automata-theoretic terms, this amounts to decide whether a given automaton  $\mathcal{A}$  is equivalent to an aperiodic one. We illustrate this problem on an example, by considering the following deterministic automata which both recognize  $a^*$  (all states are final, and all states but  $q_1$  are initial):



Roughly, an automaton is aperiodic if for some  $n$ , for all words  $u$ ,  $u^n$  and  $u^{n+1}$  behave the same with respect to their effect on states. For instance, in the left automaton, any word has one of the two following behaviours: either sending  $q_0$  to  $q_0$  and  $q_1$  to  $q_1$ , or  $q_0$  to  $q_1$  and  $q_1$  to  $q_0$ . This automaton is not aperiodic because  $a^n$  and  $a^{n+1}$  have necessarily two different behaviours, for all  $n \geq 0$ . However, this left automaton is equivalent to the right one, which is aperiodic. In general, it is not easy to see whether some automaton  $\mathcal{A}$  is equivalent to an aperiodic one, and this problem is known to be PSPACE-complete when  $\mathcal{A}$  is deterministic. To decide it, the connection between automata and algebra plays an important role. Indeed, since aperiodic automata and aperiodic right congruences both characterize the same class of languages, it suffices to (1) minimize  $\mathcal{A}$  into the unique minimal automaton  $\mathcal{A}_m$  (which is an effective representation of the right syntactic congruence of  $L(\mathcal{A})$ ) and (2) decide whether  $\mathcal{A}_m$  is aperiodic. It is well-known that step (1) is in PTIME since  $\mathcal{A}$  is deterministic, and step (2) is known to be in PSPACE [22], and this is optimal [5]. In this paper, our goal is to extend this decidability result to functions of finite words, called transductions.

**Rational transductions and the aperiodicity problem**

A transduction is a function of finite words. Rational transductions are the transductions realized by finite automata with outputs, called *transducers* [2]. As an example, consider the three following transducers:

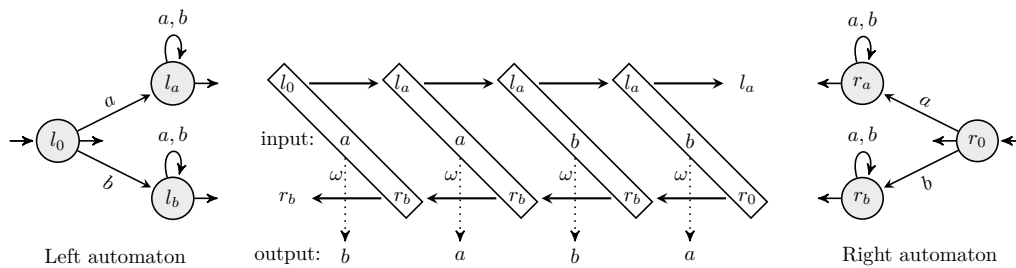


The left one maps any word of the form  $a^n$  to  $b^n$ . The middle one realizes the same transduction, and the right one maps any word of the form  $a^{2n}$  to  $(ab)^n$ , and any  $a^{2n+1}$  to  $(ab)^n a$ . *Aperiodic rational transductions* are the transductions realized by transducers with aperiodic underlying input automata. E.g., the transducer on the left is not aperiodic, but is equivalent to the middle one, which is aperiodic. Hence both transducers realize an aperiodic rational transduction. However, the transducer on the right is not aperiodic, and is not equivalent to any aperiodic transducer. The left and right transducers are almost the same, but one realizes an aperiodic rational transduction while the other does not. It shows that to decide whether a transducer is equivalent to an aperiodic one, outputs must be taken into account as well, reasoning only on the underlying input automata is not sufficient.

The *aperiodicity problem* asks, given some effective representation of a rational transduction, whether this transduction is aperiodic. It has been shown in [11] that two-way transducers (resp. aperiodic two-way transducers [14, 4]) are expressively equivalent to

MSOT (resp. FOT), a formalism introduced by Courcelle in the general context of graph transductions [7]. This equivalence carries over to the subclass of rational functions (resp. aperiodic rational functions) by considering MSOT (resp. FOT) with the natural restriction of *order-preserving* [3, 12]. As for languages, solving the aperiodicity problem also solves the logic definability problem FOT in MSOT (resp. order-preserving FOT in order-preserving MSOT). As we have seen, for rational languages, the aperiodicity checking procedure heavily relies on the existence of a unique minimal deterministic automaton. However in the setting of transductions, determinism is not sufficient to capture all rational transductions. In transducer theory, the notion of determinism is called *sequentiality*, a transducer being sequential if its underlying input automaton is deterministic. Consider the transduction *swap* that swaps the first and last letter of a word, *i.e.* maps any word of the form  $\sigma w\beta$ , where  $\sigma, \beta$  are symbols and  $w$  is a word, to  $\beta w\sigma$ . If the alphabet has more than one symbol, the transduction *swap* cannot be realized by a sequential transducer, although it can be easily shown that it is rational. The reason is that any transducer realizing it should guess in advance the last symbol  $\beta$ , by using non-determinism.

To overcome this issue, it has been shown that any rational transduction is the composition of a (left) sequential and a right sequential transduction [10]. In other words, any rational transduction can be realized by composing a sequential transducer that reads input words from right to left, and a sequential transducer that reads words from left to right. This idea has been captured in a single deterministic device called *bimachine*, introduced by Schützenberger [20] and studied by Eilenberg [9], and Reutenauer and Schützenberger [17]. Intuitively, a bimachine is made of two deterministic automata  $\mathcal{L}$  and  $\mathcal{R}$ , and some output function  $\omega$ , and works as follows:  $\mathcal{R}$  processes an input word  $w$  from right to left and annotates it with its states. Symmetrically,  $\mathcal{L}$  processes  $w$  from left to right and annotates it with its states. Finally, the output function  $\omega$  is applied to any triple  $(r, \sigma, l)$  of the annotated word, where  $r$  is a state of  $\mathcal{R}$ ,  $\sigma$  is an input symbol of  $w$ , and  $l$  is a state of  $\mathcal{L}$ . For example, consider again the transduction *swap* on the alphabet  $\{a, b\}$ . It is realized by the following bimachine with a left deterministic automaton that remembers whether the prefix read so far is empty (state  $l_0$ ), starts with  $a$  (state  $l_a$ ) or starts with  $b$  (state  $l_b$ ). Symmetrically, the deterministic right automaton remembers information about suffixes. Finally, the output function  $\omega$  maps any triple of the form  $(l_0, \sigma, r_\beta)$  or  $(l_\beta, \sigma, r_0)$  to  $\beta$ , and any other triple  $(l, \sigma, r)$  to  $\sigma$ , for  $\sigma, \beta \in \{a, b\}$ . An execution on  $aabb$  is illustrated on the next figure:



**Contributions**

A bimachine is aperiodic if its two left and right automata are aperiodic. Aperiodic bimachines define exactly aperiodic rational transductions [18]. In this paper, our main result is to provide optimal complexity (PSPACE-complete) of the aperiodicity problem for rational transductions represented by bimachines.

We detail our contributions more precisely. In language theory, solving the aperiodicity problem relies on the existence of a unique deterministic automaton. For transductions,



there is no unique minimal (deterministic) bimachine in general, but a canonical bimachine attached to every rational transduction has been defined by Reutenauer and Schützenberger [17], and this machine can be effectively constructed from a transducer or a bimachine realizing the transduction. As a first contribution, we show that a rational transduction is aperiodic iff its canonical bimachine is aperiodic. As a consequence, this gives an algorithm to solve the aperiodicity problem: (1) construct the canonical bimachine and (2) check whether its left and right automata are aperiodic, using the algorithm of [5].

Unfortunately, step (1) cannot be done in PTIME and this is unavoidable: the canonical bimachine may be exponentially bigger than the initial bimachine. Instead of constructing the canonical bimachine, we show that it is sufficient to construct another minimal bimachine of polynomial size, which is aperiodic iff the function it realizes is aperiodic rational. This other bimachine is constructed via a PTIME generalization of a minimization procedure for automata to bimachines. This yields in the end a PSPACE algorithm, whose correctness is proved based on the aperiodicity of the canonical bimachine. The lower bound is immediate as it is already the case of deterministic automata.

### Comparison with [13]

The aperiodicity problem was already shown to be decidable in [13], however with a more general procedure working for any (decidable) variety of congruences (e.g. the class of commutative congruences). More precisely, it is shown in [13] that the following problem is decidable: given a transducer, is it equivalent to some transducer whose transition congruence belongs to some decidable variety  $\mathbf{V}$ . It is shown that a transduction is in  $\mathbf{V}$  iff one of the minimal bimachines is in  $\mathbf{V}$ , and hence decidability comes as follows: construct the set of all minimal bimachines (shown to be finite) and test whether one of them belongs to  $\mathbf{V}$ . Instantiated by the variety of aperiodic congruences, this solves the aperiodicity problem, however with non-optimal complexity (several exponentials). Moreover, it is shown in [13] that the canonical bimachine of Reutenauer and Schützenberger does not necessarily preserve the equalities of a variety in general. In this paper, we show instead that for aperiodic congruences, the canonical bimachine is necessarily aperiodic if the transduction is, a result which is crucial to obtain an optimal procedure.

A last improvement compared to [13] is the following: in [13], we defined a rational transduction to be aperiodic (and more generally in a variety  $\mathbf{V}$ ) if it is realized by an *unambiguous* aperiodic transducer (or an unambiguous  $\mathbf{V}$ -transducer). This definition was motivated by the fact that unambiguous transducers already capture all rational functions. For a general variety  $\mathbf{V}$ , this left open the problem of whether any transduction realized by a  $\mathbf{V}$ -transducer is realizable by an unambiguous  $\mathbf{V}$ -transducer. In this paper, we close this problem for the case of aperiodicity: as we show, a transduction is realized by some aperiodic transducer (not necessarily unambiguous) iff its canonical bimachine is aperiodic, and any aperiodic bimachine can be turned into an aperiodic unambiguous transducer.

## 2 Rational languages and transductions

### Words and languages

An alphabet  $\Sigma$  is a finite set of symbols, and a word over  $\Sigma$  is an element of the free monoid  $\Sigma^*$  whose neutral element is denoted by  $\epsilon$ . For  $w \in \Sigma^*$ , we write  $|w|$  for its length and in particular,  $|\epsilon| = 0$ . For a non-empty word  $w$  and  $i \in \{1, \dots, |w|\}$ , we denote by  $w[i]$  the  $i$ th symbol of  $w$ . For  $u, v \in \Sigma^*$ , we write  $u \preceq v$  if  $u$  is a prefix of  $v$  and in this case we denote by  $u^{-1}v$  the unique word  $v'$  such that  $v = uv'$ .



By  $u \wedge v$  we denote the *longest common prefix* of any two words  $u$  and  $v$ , and by  $\|u, v\|$  the value  $|u| + |v| - 2|u \wedge v|$ . It is well-known that  $\|\cdot, \cdot\|$  defines a distance. Finally, a *language*  $L \subseteq \Sigma^*$  is a set of words.

### Finite automata

A *finite automaton* (or just automaton for short) over an alphabet  $\Sigma$  is a tuple  $\mathcal{A} = (Q, I, F, \Delta)$  where  $Q$  is a finite set of states,  $I \subseteq Q$  (resp.  $F \subseteq Q$ ) is a set of initial (resp. final) states, and  $\Delta \subseteq Q \times \Sigma \times Q$  is a transition relation. A *run*  $r$  of an automaton  $\mathcal{A} = (Q, I, F, \Delta)$  on a word  $w \in \Sigma^*$  of length  $n$  is a word  $r = q_0 \dots q_n$  over  $Q$  such that  $(q_i, w[i+1], q_{i+1}) \in \Delta$  for all  $i \in \{0, \dots, n-1\}$ . The run  $r$  is accepting if  $q_0 \in I$  and  $q_n \in F$ . A word is accepted by  $\mathcal{A}$  if there exists an accepting run of  $\mathcal{A}$  over it. The *language recognized by  $\mathcal{A}$*  is the set  $\llbracket \mathcal{A} \rrbracket$  of words accepted by  $\mathcal{A}$ . We write  $p \xrightarrow{w}_{\mathcal{A}} q$  (or simply  $p \xrightarrow{w} q$ ) whenever there exists a run  $r$  on  $w$  such that  $r[1] = p$  and  $r[|r|] = q$ . An automaton  $\mathcal{A}$  is *deterministic* if  $|I| = 1$  and for any two rules  $(p, \sigma, q_1), (p, \sigma, q_2) \in \Delta$ , it holds that  $q_1 = q_2$ . It is *unambiguous* if for any word there exists at most one accepting run of  $\mathcal{A}$  on it. It is *complete* if for every state  $p \in Q$  and symbol  $\sigma \in \Sigma$ ,  $(p, \sigma, q) \in \Delta$  for some  $q \in Q$ .

### Congruences and recognizability

Equivalently, rational languages can be defined as the languages recognized by congruences of finite index. We present these notions. Let  $\Sigma$  be an alphabet and let  $\sim$  be an equivalence relation on  $\Sigma^*$ . We say that  $\sim$  is a *right congruence* (resp. *left congruence*) if it satisfies  $u \sim v \Rightarrow u\sigma \sim v\sigma$  (resp.  $u \sim v \Rightarrow \sigma u \sim \sigma v$ ) for all  $u, v \in \Sigma^*, \sigma \in \Sigma$ . A *congruence* is both a left and right congruence. For  $u \in \Sigma^*$ , the equivalence class of  $u$  is denoted by  $[u]_{\sim}$  (or  $[u]$  if  $\sim$  is clear from the context), and  $\Sigma^*/_{\sim} = \{[u]_{\sim} \mid u \in \Sigma^*\}$  is called the quotient of  $\Sigma^*$  by  $\sim$ . We say that  $\sim$  has *finite index* if  $\Sigma^*/_{\sim}$  is finite. Concatenation naturally extends to congruence classes as follows: for all  $u, v \in \Sigma^*$ ,  $[u]_{\sim}[v]_{\sim} = [uv]_{\sim}$ . Since  $\sim$  is a congruence, the latter is well-defined. With this operation,  $\Sigma^*/_{\sim}$  forms a monoid whose neutral element is  $[\epsilon]_{\sim}$ .

► **Example 1.** We will extensively use the following examples of congruences in this paper: the *syntactic congruence*  $\equiv_L$  of a language  $L$ , the *transition congruence*  $\approx_{\mathcal{A}}$  of an automaton  $\mathcal{A}$  with set of states  $Q$  and if  $\mathcal{A}$  is deterministic with initial state  $q_0$ , the *right transition congruence*  $\sim_{\mathcal{A}}$ . They are defined as follows, for any two words  $u, v \in \Sigma^*$

$$\begin{aligned} u \equiv_L v &\Leftrightarrow (\forall x, y \in \Sigma^*, \quad xuy \in L \Leftrightarrow xvy \in L) \\ u \approx_{\mathcal{A}} v &\Leftrightarrow (\forall p, q \in Q, \quad p \xrightarrow{u}_{\mathcal{A}} q \Leftrightarrow p \xrightarrow{v}_{\mathcal{A}} q) \\ u \sim_{\mathcal{A}} v &\Leftrightarrow (\forall q \in Q, \quad q_0 \xrightarrow{u}_{\mathcal{A}} q \Leftrightarrow q_0 \xrightarrow{v}_{\mathcal{A}} q) \end{aligned}$$

In particular, if  $\mathcal{A}$  is deterministic and complete, then  $[u]_{\sim_{\mathcal{A}}}$  can be identified with the state of  $\mathcal{A}$  reached by  $u$  from the initial state. In this paper, we often make this identification, implicitly assuming that  $\mathcal{A}$  is complete<sup>1</sup>, and rather denote  $[u]_{\mathcal{A}}$  instead of  $[u]_{\sim_{\mathcal{A}}}$ .

A language  $L \subseteq \Sigma^*$  is *recognized by a congruence*  $\sim$  if it is equal to the union of some equivalence classes of  $\Sigma^*/_{\sim}$ , i.e.  $L = \{u \in \Sigma^* \mid [u] \in P\}$  for some  $P \subseteq \Sigma^*/_{\sim}$ . E.g.,  $L$  is recognized by  $\equiv_L$ , by taking  $P = L/\equiv_L$ , and any language  $L$  is rational iff it is recognized by a congruence of finite index (see for instance [23]).

<sup>1</sup> Any automaton can be made complete in polynomial-time.

Equivalence relations can be compared with respect to their granularity: if  $\sim_1, \sim_2$  are two equivalence relations on  $\Sigma^*$ , we say that  $\sim_1$  is *finer* than  $\sim_2$  (or  $\sim_2$  is *coarser* than  $\sim_1$ ), if for all  $u, v \in \Sigma^*$  such that  $u \sim_1 v$ , it holds  $u \sim_2 v$ . We write  $\sim_1 \sqsubseteq \sim_2$  to mean that  $\sim_1$  is finer than  $\sim_2$ . E.g.,  $\equiv_L$  is the coarsest congruence recognizing  $L$ , for any language  $L$ . In this paper, we also compare deterministic automata  $\mathcal{A}_1, \mathcal{A}_2$  with respect to their right transition congruence, by saying that  $\mathcal{A}_1$  is *finer* than  $\mathcal{A}_2$  if  $\sim_{\mathcal{A}_1} \sqsubseteq \sim_{\mathcal{A}_2}$ . We write  $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$  when  $\mathcal{A}_1$  is finer than  $\mathcal{A}_2$ . For example, the minimal deterministic automaton recognizing a language  $L$  is the coarsest deterministic automaton recognizing  $L$  with respect to  $\sqsubseteq$ .

### Rational transductions and finite transducers

A *transduction*  $f$  over a finite alphabet  $\Sigma$  is a partial function from  $\Sigma^*$  to  $\Sigma^*$ , whose domain is denoted by  $\text{dom}(f)$ . We are interested in the class of rational transductions, defined by finite transducers. A *finite transducer*<sup>2</sup> (or just transducer for short) over an alphabet  $\Sigma$  is a tuple  $\mathcal{T} = (\mathcal{A}, o, i, t)$  where  $\mathcal{A} = (Q, I, F, \Delta)$  is a finite automaton,  $o : \Delta \rightarrow \Sigma^*$  is the *output function*,  $i : I \rightarrow \Sigma^*$  is the *initial function* and  $t : F \rightarrow \Sigma^*$  is the *final function*. The transducer  $\mathcal{T}$  realizes a binary relation  $\llbracket \mathcal{T} \rrbracket \subseteq \Sigma^* \times \Sigma^*$  defined as follows. Let  $r = q_0 \dots q_n$  be a run of  $\mathcal{A}$  on some word  $u$ . We write  $q_0 \xrightarrow{u|v}_{\mathcal{T}} q_n$  (or simply  $q_0 \xrightarrow{u|v} q_n$ ) whenever  $q_0 \xrightarrow{u}_{\mathcal{A}} q_n$  and  $v = o(q_0, u[0], q_1) \dots o(q_{n-1}, u[n], q_n)$ . If  $r$  is an accepting run and  $w = i(q_0)vt(q_n)$  then we say that  $(u, w)$  is *realized* by  $\mathcal{T}$ , call  $u$  an *input* word and  $w$  an *output* word. The *relation realized* by  $\mathcal{T}$  is the set  $\llbracket \mathcal{T} \rrbracket = \{(u, w) \mid (u, w) \text{ is realized by } \mathcal{T}\}$ .

A transducer  $\mathcal{T} = (\mathcal{A}, o, i, t)$  is *functional* if it realizes a transduction (a function). This property is decidable in PTIME (see [1] for instance).  $\mathcal{T}$  is called *unambiguous* (resp. *sequential*) if  $\mathcal{A}$  is unambiguous (resp. deterministic). In both cases  $\llbracket \mathcal{T} \rrbracket$  is a transduction and we denote  $(u, w) \in \llbracket \mathcal{T} \rrbracket$  by  $\llbracket \mathcal{T} \rrbracket(u) = w$ . The class of *rational transductions* (resp. *sequential transductions*) is defined as the class of transductions realized by finite transducers (resp. sequential transducers). It is also known (see [2]) that a transduction is rational iff it is realized by some unambiguous transducer.

### Aperiodicity

We define the notion of aperiodicity for congruences, automata, transducers and rational transductions. First, a congruence  $\sim$  on  $\Sigma^*$  is *aperiodic* if for some  $n > 0$  and all words  $w \in \Sigma^*$ , we have  $w^n \sim w^{n+1}$ . Aperiodicity is stable by taking coarser congruences, *i.e.* if  $\sim_1 \sqsubseteq \sim_2$  and  $\sim_1$  is aperiodic, then so is  $\sim_2$ . A deterministic automaton  $\mathcal{A}$  is aperiodic if  $\approx_{\mathcal{A}}$  is aperiodic. In other words,  $\mathcal{A}$  is aperiodic if for some  $n > 0$ , for all words  $w$  and states  $p, q$ , we have  $p \xrightarrow{w^n} q$  iff  $p \xrightarrow{w^{n+1}} q$ . Deciding whether a deterministic automaton is aperiodic is PSPACE-complete [5]. Since the minimal deterministic automaton  $\mathcal{A}_L$  recognizing a language  $L$  is the coarsest automaton recognizing  $L$ , and aperiodicity is stable by taking coarser congruences,  $\mathcal{A}_L$  is aperiodic iff there is some aperiodic deterministic automaton recognizing  $L$ . Therefore, deciding whether a deterministic automaton is *equivalent* to some aperiodic one is also PSPACE-complete, because minimizing a deterministic automaton can be done in PTIME. Finally, a transducer  $\mathcal{T} = (\mathcal{A}, o, i, t)$  is *aperiodic* if  $\mathcal{A}$  is aperiodic, and a transduction  $f$  is *aperiodic rational* (resp. *aperiodic sequential*) if it is realized by some aperiodic transducer (resp. aperiodic sequential transducer).

<sup>2</sup> This type of transducer is sometimes called *real-time transducer* [19]. In the general case, a transition of a transducer may be labelled by any word, even empty. However such a transducer is equivalent to a real-time one if it realizes a function.

### 3 Bimachines and minimization

In this section we define bimachines, and associated operators *Left* and *Right*. Then we define canonical bimachines, that will be used in Section 4 to prove that the minimal bimachine  $Left(Right(\mathcal{B}))$  is aperiodic iff the transduction realized by the bimachine  $\mathcal{B}$  is.

#### 3.1 Bimachines

A bimachine is a model of computation, as expressive as (functional) transducers, introduced by Schützenberger in [20]. It is composed of two automata, an automaton reading words deterministically backwards, called a right automaton, and a classical deterministic automaton called here a left automaton. The right automaton acts as a deterministic look-ahead. An output function produces words based on the current symbol, and the states of the left and right automata.

More precisely, a *right automaton*  $\mathcal{R} = (Q, I, F, \Delta)$  is an automaton such that  $I$  is a singleton, and transitions are backward deterministic: for any transitions  $(p_1, \sigma, q), (p_2, \sigma, q) \in \Delta$  it holds that  $p_1 = p_2$ . The only difference with a (classical) automaton lies in the notion of accepting runs (and therefore in the notion of recognized language): a run  $r$  is accepting if  $r[1]$  is final and  $r[|r|]$  is initial. Therefore, a right automaton can be thought of as an automaton reading words backwards. We write  $s_2 \xleftarrow{\mathcal{R}} s_1$  instead of  $s_2 \xrightarrow{\mathcal{R}} s_1$  (with the same meaning) to emphasize that  $\mathcal{R}$  is a right automaton, and graphically any transition  $(q, \sigma, p) \in \Delta$  is depicted with an arrow from  $p$  to  $q$ . For instance, the accepting run on  $ba$  of the right automaton of the bimachine depicted in Section 1 is  $r_b r_a r_0$ . The *left transition congruence*  $\sim_{\mathcal{R}}$  of  $\mathcal{R}$  is defined by  $u \sim_{\mathcal{R}} v \Leftrightarrow (\forall r \in R, r \xleftarrow{\mathcal{R}} r_0 \Leftrightarrow r \xleftarrow{\mathcal{R}} r_0)$ . Implicitly assuming that  $\mathcal{R}$  is complete, we identify  $[u]_{\sim_{\mathcal{R}}}$  (also just denoted by  $[u]_{\mathcal{R}}$ ) with the state  $r \in R$  such that  $r \xleftarrow{\mathcal{R}} r_0$ . We say that  $\mathcal{R}$  is finer than a right automaton  $\mathcal{R}'$  (written  $\mathcal{R} \sqsubseteq \mathcal{R}'$ ) if  $\sim_{\mathcal{R}} \sqsubseteq \sim_{\mathcal{R}'}$ . A *left automaton* is a deterministic automaton, called 'left' to emphasize its role in the context of bimachines.

A *bimachine* over an alphabet  $\Sigma$  is a tuple  $\mathcal{B} = (\mathcal{L}, \mathcal{R}, \omega, \lambda, \rho)$  where  $\mathcal{L} = (L, \{l_0\}, F_{\mathcal{L}}, \Delta_{\mathcal{L}})$  is a left automaton,  $\mathcal{R} = (R, \{r_0\}, F_{\mathcal{R}}, \Delta_{\mathcal{R}})$  is a right automaton,  $\omega : L \times \Sigma \times R \rightarrow \Sigma^*$  is the *output function*,  $\lambda : F_{\mathcal{R}} \rightarrow \Sigma^*$  is the *left final function* and  $\rho : F_{\mathcal{L}} \rightarrow \Sigma^*$  is the *right final function*. Both automata  $\mathcal{L}$  and  $\mathcal{R}$  must recognize the same language, *i.e.*  $\llbracket \mathcal{L} \rrbracket = \llbracket \mathcal{R} \rrbracket$ .

We now define the *transduction*  $\llbracket \mathcal{B} \rrbracket$  realized by  $\mathcal{B}$ . First, we extend the function  $\omega$  to  $L \times \Sigma^* \times R$  as follows: for all states  $r \in R$  and  $l \in L$ , all  $u \in \Sigma^*$  and  $\sigma \in \Sigma$ ,  $\omega(l, \epsilon, r) = \epsilon$ , and  $\omega(l, u\sigma, r) = \omega(l, u, r')\omega(l', \sigma, r)$  where  $l \xrightarrow{\mathcal{L}} l'$  and  $r' \xleftarrow{\mathcal{R}} r$ . Now, the domain  $\text{dom}(\mathcal{B})$  of  $\mathcal{B}$  is  $\llbracket \mathcal{L} \rrbracket = \llbracket \mathcal{R} \rrbracket$ . For all  $u \in \llbracket \mathcal{L} \rrbracket$ , if  $l_0 \xrightarrow{\mathcal{L}} l$  for some  $l \in F_{\mathcal{L}}$  and  $r \xleftarrow{\mathcal{R}} r_0$  for some  $r \in F_{\mathcal{R}}$ , the image of  $u$  by  $\mathcal{B}$  is defined by  $\llbracket \mathcal{B} \rrbracket(u) = \lambda(r)\omega(l_0, u, r_0)\rho(l)$ .

#### 3.2 Left and right bimachine minimization

Sequential transducers can be minimized by producing the outputs as early as possible [6]. No such procedure exist for transducers in general. For bimachines, however, a similar procedure is proposed in [17]. This one applies the ‘‘as early as possible’’ principle, but parameterized by the look-ahead information of the right automaton. We describe this minimization, exhibit some useful properties, and provide a PTIME minimization algorithm.

Let  $f$  be a rational transduction realized by a bimachine  $\mathcal{B}$  whose right automaton is  $\mathcal{R}$ . Our goal here is to construct a bimachine  $Left_f(\mathcal{B}) = (Left_f(\mathcal{R}), \mathcal{R}, \omega, \lambda, \rho)$ , which realizes  $f$  and has the minimal left automaton among bimachines realizing  $f$  with  $\mathcal{R}$  as its right automaton. We first give the construction of a minimal (wrt  $\mathcal{R}$ ) left automaton

$Left_f(\mathcal{R})$  (or simply  $Left(\mathcal{R})$  when it is clear from context). For simplicity, we will write  $[w]_{\mathcal{R}}$  instead of  $[w]_{\sim_{\mathcal{R}}}$  for any word  $w \in \Sigma^*$ . For any two words  $w$  and  $u$ , we define <sup>3</sup>  $\widehat{f}_{[w]_{\mathcal{R}}}(u) = \bigwedge \{f(uv) \mid v \in [w]_{\mathcal{R}} \cap u^{-1}\text{dom}(f)\}$ .

This word is the longest possible output upon reading  $u$ , knowing that the suffix is in  $[w]_{\mathcal{R}}$ . The states of  $Left_f(\mathcal{R})$  will be the classes of the right congruence  $\sim_L$  defined by  $u \sim_L v$  if for any letter  $\sigma$ , any  $w, z \in \Sigma^*$  we have:

- $uz \in \text{dom}(f) \Leftrightarrow vz \in \text{dom}(f)$
- $\widehat{f}_{[\epsilon]_{\mathcal{R}}}(uz)^{-1}f(uz) = \widehat{f}_{[\epsilon]_{\mathcal{R}}}(vz)^{-1}f(vz)$ , if  $uz, vz \in \text{dom}(f)$
- $\widehat{f}_{[\sigma w]_{\mathcal{R}}}(uz)^{-1}\widehat{f}_{[w]_{\mathcal{R}}}(uz\sigma) = \widehat{f}_{[\sigma w]_{\mathcal{R}}}(vz)^{-1}\widehat{f}_{[w]_{\mathcal{R}}}(vz\sigma)$

Intuitively, the second condition ensures that after reading  $uz$  and  $vz$ ,  $Left_f(\mathcal{R})$  outputs the same word by the final output function, and the third condition ensures that the output produced on  $\sigma$  is the same after reading  $uz$  and  $vz$ . From  $\sim_L$  we define the automaton  $Left_f(\mathcal{R}) = (\Sigma^*/\sim_L, \{[\epsilon]_{\sim_L}\}, F, \Delta)$  where  $F = \{[w]_{\sim_L} \mid w \in \text{dom}(f)\}$  and  $\Delta = \{([w]_{\sim_L}, \sigma, [w\sigma]_{\sim_L}) \mid \sigma \in \Sigma, w \in \Sigma^*\}$ . Finally, the output functions are defined by:  $\omega([u]_{\sim_L}, \sigma, [v]_{\mathcal{R}}) = \widehat{f}_{[\sigma v]_{\mathcal{R}}}(u)^{-1}\widehat{f}_{[v]_{\mathcal{R}}}(u\sigma)$ ,  $\lambda([v]_{\mathcal{R}}) = \widehat{f}_{[v]_{\mathcal{R}}}(\epsilon)$  and  $\rho([u]_{\sim_L}) = \widehat{f}_{[\epsilon]_{\mathcal{R}}}(u)^{-1}f(u)$ .

Symmetrically one can define  $Right_f(\mathcal{L})$  (and hence  $Right_f(\mathcal{B})$ ).

The correctness of these constructions was shown in [17], *i.e.*  $Left(\mathcal{B})$  and  $Right(\mathcal{B})$  both realize  $f$ . The following proposition shows that  $Left(\mathcal{R})$  and  $Right(\mathcal{L})$  are minimal automata for which the bimachines (with fixed  $\mathcal{R}$  and  $\mathcal{L}$  respectively) realize  $f$ .<sup>4</sup>

► **Proposition 2.** *If  $\mathcal{B} = (\mathcal{L}, \mathcal{R}, \omega, \lambda, \rho)$  is a bimachine, then  $\mathcal{L} \sqsubseteq Left(\mathcal{R})$  and  $\mathcal{R} \sqsubseteq Right(\mathcal{L})$ .*

One contribution of this paper is to show that the left automaton can be minimized in PTIME (for a fixed right automaton), and symmetrically for the right automaton.

► **Theorem 3.** *Let  $\mathcal{B}$  be a bimachine. One can compute  $Left(\mathcal{B})$  (and  $Right(\mathcal{B})$ ) in PTIME.*

**Proof.** Let  $\mathcal{B} = (\mathcal{L}, \mathcal{R}, \omega, \lambda, \rho)$  be a bimachine realizing a function  $f$  with automata  $\mathcal{L} = (Q_{\mathcal{L}}, l_0, F_{\mathcal{L}}, \Delta_{\mathcal{L}})$  and  $\mathcal{R} = (Q_{\mathcal{R}}, r_0, F_{\mathcal{R}}, \Delta_{\mathcal{R}})$ . W.l.o.g. we assume that  $\mathcal{L}$  is complete (otherwise we complete it in polynomial time). The algorithm works in two steps: (i) make the output production earliest, (ii) run state minimization on the left automaton.

**Step 1: making the bimachine earliest.** We construct  $\mathcal{B}' = (\mathcal{L}, \mathcal{R}, \omega', \lambda', \rho')$  a bimachine realizing  $f$  with the same automata, but with the earliest leftmost possible outputs:

$$\omega'([u]_{\mathcal{L}}, \sigma, [v]_{\mathcal{R}}) = \widehat{f}_{[\sigma v]_{\mathcal{R}}}(u)^{-1}\widehat{f}_{[v]_{\mathcal{R}}}(u\sigma) \quad \rho'([u]_{\mathcal{L}}) = \widehat{f}_{[\epsilon]_{\mathcal{R}}}(u)^{-1}f(u) \quad \lambda'([v]_{\mathcal{R}}) = \widehat{f}_{[v]_{\mathcal{R}}}(\epsilon).$$

These functions are well-defined as they do not depend on the choice of the representatives  $u$  (see the proof of Proposition 2). We have to show that these output values can be computed in polynomial time. The algorithm is very close to the ones described in [6], for sequential functions, which is why we only give the main ideas. We first remark, as in the proof of Proposition 2, that for any words  $u, v$ ,  $\widehat{f}_{[v]_{\mathcal{R}}}(u) = \lambda([uv]_{\mathcal{R}})\omega([\epsilon]_{\mathcal{L}}, u, [v]_{\mathcal{R}})\beta([u]_{\mathcal{L}}, [v]_{\mathcal{R}})$  with:

$$\beta([u]_{\mathcal{L}}, [v]_{\mathcal{R}}) = \bigwedge \{w \mid \exists x \in [v]_{\mathcal{R}} \cap u^{-1}\text{dom}(f), \omega([u]_{\mathcal{L}}, x, [\epsilon]_{\mathcal{R}})\rho([ux]_{\mathcal{L}}) = w\}$$

As in [6], to compute the values  $\beta([u]_{\mathcal{L}}, [v]_{\mathcal{R}})$  we can take the directed graph of the automaton  $\mathcal{L} \times \mathcal{R}$  with the outputs of  $\omega$  labelling the edges. In order to account for the functions  $\lambda$  and  $\rho$  we add two vertices, a source  $s$  pointing to the initial states with the edges labelled

<sup>3</sup> As a convention, the longest common prefix of an empty set of words is the empty word, that is:  $\bigwedge \emptyset = \epsilon$ .

<sup>4</sup> It was already shown in [17] for total functions, we extend it with a similar proof to our setting, that is, when the function is not total, and the automata of the bimachine both recognize  $\text{dom}(f)$ .

accordingly by the values of the initial function, and a target vertex  $t$  with edges pointing to it from the final states with the edges labelled by the final function. The value of  $\beta([u]_{\mathcal{L}}, [v]_{\mathcal{R}})$  can be seen as the longest common prefix of the labels of all the paths starting in  $([u]_{\mathcal{L}}, [v]_{\mathcal{R}})$  and ending in the target vertex  $t$ . These values can be computed in polynomial time [6].

**Step 2: minimizing the left automaton.** Now we describe a minimization algorithm inspired by Moore's minimization algorithm for DFA. It consists in computing successively finer equivalence relations  $\sim_0, \sim_1, \dots$  over the states of  $\mathcal{L}$ . The only difference is in the initial partition, for which we have to make sure that outputs are compatible. More precisely, for all  $l, l' \in Q_{\mathcal{L}}$  and  $i \geq 0$ , we let:

$$\begin{aligned} l \sim_0 l' & \text{ if } l \in F_{\mathcal{L}} \Leftrightarrow l' \in F_{\mathcal{L}}, \rho'(l) = \rho'(l'), \text{ and } \forall r, \sigma, \omega'(l, \sigma, r) = \omega'(l', \sigma, r) \\ l \sim_{i+1} l' & \text{ if } l \sim_i l' \text{ and } \forall \sigma, l.\sigma \sim_i l'.\sigma \end{aligned}$$

where  $l.\sigma$  denotes the state reached by  $\mathcal{L}$  after reading the letter  $\sigma$  from  $l$ . We extend this notation to words in the natural way:  $l.u$  is the state such that  $l \xrightarrow{u}_{\mathcal{L}} l.u$  (it is unique as  $\mathcal{L}$  is deterministic and complete).

Since  $\sim_{i+1} \sqsubseteq \sim_i$  for all  $i \geq 0$ , this sequence converges after at most  $|Q_{\mathcal{L}}|$  steps to an equivalence relation that we denote by  $\sim_*$ . Moreover,  $\sim_0$  can be computed in PTIME from  $\mathcal{B}'$ , and each  $\sim_i$  can be computed in PTIME from  $\sim_{i-1}$ , for  $i > 0$ .

We extend the relations  $\sim_i$  to  $\Sigma^*$  as follows:  $u \sim_i v$  if  $l_0.u \sim_i l_0.v$ . We show in the long version of this paper that  $\sim_* = \sim_L$  (remind that  $\sim_L$  is the right congruence associated with  $\mathcal{R}$  and used to define  $Left(\mathcal{R})$ ). To give an idea of the proof, we first show by induction on  $i \geq 0$  that for all  $u, v \in \Sigma^*$ ,  $u \sim_i v$  iff for all  $z \in \Sigma^i$ , all  $w \in \Sigma^*$  and all  $\sigma \in \Sigma$ , we have (i)  $uz \in \text{dom}(f)$  iff  $vz \in \text{dom}(f)$ , (ii)  $\rho'([uz]_{\mathcal{L}}) = \rho'([vz]_{\mathcal{L}})$  (if  $uz, vz \in \text{dom}(f)$ ), and (iii)  $\omega'([uz]_{\mathcal{L}}, \sigma, [w]_{\mathcal{R}}) = \omega'([vz]_{\mathcal{L}}, \sigma, [w]_{\mathcal{R}})$ . This implies that  $u \sim_* v$  iff the properties (i)–(iii) holds for all  $z \in \Sigma^*$ . Finally, we conclude by noticing that  $\rho'([uz]_{\mathcal{L}}) = \hat{f}_{[\epsilon]_{\mathcal{R}}}(uz)^{-1} f(uz)$ , and  $\omega'([uz]_{\mathcal{L}}, \sigma, [w]_{\mathcal{R}}) = \hat{f}_{[\sigma w]_{\mathcal{R}}}(uz)^{-1} \hat{f}_{[w]_{\mathcal{R}}}(uz\sigma)$ .

Clearly, if  $l \sim_* l'$ , then for all  $\sigma \in \Sigma$ ,  $l.\sigma \sim_* l'.\sigma$ . Moreover, if  $l \in F_{\mathcal{L}}$ , then since  $\sim_* \sqsubseteq \sim_0$ , we also get  $l' \in F_{\mathcal{L}}$ , and conversely. Therefore one can define the left automaton  $\mathcal{L}/\sim_* = (Q_{\mathcal{L}}/\sim_*, F_{\mathcal{L}}/\sim_*, [l_0]_{\sim_*}, \delta)$  where  $\delta([l]_{\sim_*}, \sigma) = [l.\sigma]_{\sim_*}$ , and we have  $\llbracket \mathcal{L}/\sim_* \rrbracket = \llbracket \mathcal{L} \rrbracket$ .

Finally,  $\sim_* = \sim_L$  implies that the bimachine  $Left_f(\mathcal{B})$  is isomorphic to the bimachine  $(\mathcal{L}/\sim_*, \mathcal{R}, \omega'', \lambda', \rho'')$  where  $\omega''([u]_{\sim_*}, \sigma, [v]_{\mathcal{R}}) = \omega'([u]_{\mathcal{L}}, \sigma, [v]_{\mathcal{R}})$  and  $\rho''([u]_{\sim_*}) = \rho'([u]_{\mathcal{L}})$ . Note that these output functions are well-defined since, by definition,  $\sim_*$  is compatible with the output functions  $\lambda', \omega', \rho'$ . Moreover,  $\mathcal{L}/\sim_*$  can be computed in PTIME since  $\sim_*$  can be computed in PTIME, and the output functions  $\omega'', \rho''$  can as well be computed in PTIME, which concludes the proof. A last remark is that a Hopcroft-like minimization algorithm [15], initialized with  $\sim_0$  as well, would be more efficient, but with a more involved proof. ◀

### 3.3 A minimal and canonical bimachine

For a given function  $f$  and two bimachines  $\mathcal{B}_1$  and  $\mathcal{B}_2$  realizing it, we say that  $\mathcal{B}_1$  is *finer* than  $\mathcal{B}_2$ , denoted again by  $\mathcal{B}_1 \sqsubseteq \mathcal{B}_2$ , if we have both  $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$  and  $\mathcal{R}_1 \sqsubseteq \mathcal{R}_2$ , with  $\mathcal{L}_i$  and  $\mathcal{R}_i$  being the left and the right automata of bimachine  $\mathcal{B}_i$ , respectively. We say that a bimachine is *minimal* if there is no coarser bimachine realizing the same transduction. There is no unique minimal bimachine in general [17]. In this section, we explain the construction of a minimal and canonical bimachine associated with a rational transduction, the main result of [17]. It relies on (a) a canonical right automaton, that we detail hereafter, and (b) the construction of a minimal left automaton from a right automaton, as described in Section 3.2.

## 13:10 Aperiodicity of Rational Functions Is PSPACE-Complete

The construction of a *canonical right automaton* is based on a left congruence associated with a function that measures the effect of suffixes on the translation of prefixes. The *left congruence* of a transduction  $f$  on  $\Sigma$  is defined  $\forall u, v \in \Sigma^*$  by  $u \prec_f v$  if:

- $\forall w \in \Sigma^*, wu \in \text{dom}(f) \Leftrightarrow vw \in \text{dom}(f)$  and
- $\sup\{\|f(wu), f(vw)\| \mid wu, vw \in \text{dom}(f)\} < \infty$

This congruence has finite index if  $f$  is rational [17]. The converse does not hold but if additionally  $f^{-1}$  preserves language rationality, then  $f$  is rational. For the rest of this section  $[w]$  denotes the class of  $w$  in  $\Sigma^*/\prec_f$ . The *canonical right automaton* for  $f$  is  $\mathcal{R}_f = (\Sigma^*/\prec_f, \{\epsilon\}, F, \Delta)$  where  $F = \{[w] \mid w \in \text{dom}(f)\}$  and  $\Delta = \{([\sigma w], \sigma, [w]) \mid \sigma \in \Sigma, w \in \Sigma^*\}$ .

► **Remark.** We can define symmetrically the *right congruence* of  $f$  by  $u \rightarrow_f v$  if  $\forall w, uw \in \text{dom}(f) \Leftrightarrow vw \in \text{dom}(f)$  and  $\sup\{\|f(uw), f(vw)\| \mid uw, vw \in \text{dom}(f)\} < \infty$  and based on this right congruence, define the *canonical left automaton*  $\mathcal{L}_f$ .

The automata  $\mathcal{R}_f$  and  $\mathcal{L}_f$  are coarser than any right (resp. left) automaton of a bimachine realizing  $f$ . This was shown in [13] but only for the case of total functions. The proof is similar in the general case and we give it in the long version of this paper for completeness.

► **Proposition 4.** *Let  $f$  be a transduction, and let  $\mathcal{B} = (\mathcal{L}, \mathcal{R}, \omega, \lambda, \rho)$  be a bimachine realizing  $f$ . Then  $\mathcal{L} \sqsubseteq \mathcal{L}_f$  and  $\mathcal{R} \sqsubseteq \mathcal{R}_f$ .*

The *canonical bimachine* [17] associated with a rational transduction  $f$  is the bimachine  $\mathcal{B}_f = (\text{Left}_f(\mathcal{R}_f), \mathcal{R}_f, \omega_f, \lambda_f, \rho_f)$  where:

$$\begin{aligned} \omega_f([u]_{\sim_L}, \sigma, [v]_{\mathcal{R}_f}) &= \widehat{f}_{[\sigma v]_{\mathcal{R}_f}}(u)^{-1} \widehat{f}_{[v]_{\mathcal{R}_f}}(u\sigma) \\ \lambda_f([v]_{\mathcal{R}_f}) &= \widehat{f}_{[v]_{\mathcal{R}_f}}(\epsilon) \\ \rho_f([u]_{\sim_L}) &= \widehat{f}_{[\epsilon]_{\mathcal{R}_f}}(u)^{-1} f(u) \end{aligned}$$

By its definition, the bimachine  $\mathcal{B}_f$  is canonical, *i.e.* does not depend on any description of  $f$ . It is also minimal: indeed, suppose that  $f$  is realized by a bimachine  $\mathcal{B} = (\mathcal{L}, \mathcal{R}, \omega, \lambda, \rho)$  such that  $\mathcal{B}_f \sqsubseteq \mathcal{B}$ , *i.e.*  $\text{Left}(\mathcal{R}_f) \sqsubseteq \mathcal{L}$  and  $\mathcal{R}_f \sqsubseteq \mathcal{R}$ . Then  $\omega$  (and similarly  $\lambda, \rho$ ) can be restricted to  $\omega'([u]_{\mathcal{L}}, \sigma, [v]_{\mathcal{R}_f}) = \omega([u]_{\mathcal{L}}, \sigma, [v]_{\mathcal{R}})$ , which is well-defined since  $\mathcal{R}_f \sqsubseteq \mathcal{R}$ , so that the bimachine  $(\mathcal{L}, \mathcal{R}_f, \omega', \lambda', \rho')$  realizes  $f$ . By Proposition 2 we get  $\mathcal{L} \sqsubseteq \text{Left}(\mathcal{R}_f)$ . Moreover, by Proposition 4, we also have  $\mathcal{R} \sqsubseteq \mathcal{R}_f$ .

Finally,  $\mathcal{B}_f$  is computable when  $f$  is given by a bimachine or a transducer [17].

## 4 Characterization of aperiodic transductions

In this section we show that to decide if a transduction given by a bimachine  $\mathcal{B}$  is aperiodic, one only needs to minimize  $\mathcal{B}$ , *i.e.* to construct  $\text{Left}(\text{Right}(\mathcal{B}))$ , which yields a minimal bimachine, and check its aperiodicity (Section 4.2). To prove the correctness of this procedure, we rely on the following characterization proved in Section 4.1: a transduction  $f$  is aperiodic if and only if the canonical bimachine  $\mathcal{B}_f$  is aperiodic. This is in contrast with other varieties for which the canonical bimachine does not preserve membership in general [13]. The latter characterization does not yield an optimal algorithm since the canonical bimachine may be exponentially larger than the initial bimachine.

### 4.1 Characterization through canonical bimachine

In this section, given a rational transduction  $f$  over some alphabet  $\Sigma$ , we show that it is aperiodic iff the canonical bimachine  $\mathcal{B}_f$  associated with  $f$ , defined in the previous section,



is aperiodic. It relies on two important facts: (i) the left congruence  $\leftarrow_f$  of an aperiodic transduction is aperiodic (Proposition 5), (ii) any aperiodic rational transduction  $f$  can be decomposed into  $f = \ell \circ \text{label}_{\mathcal{R}_f}$  such that  $\ell$  is realized by some aperiodic sequential transducer, and  $\text{label}_{\mathcal{R}_f}$  annotates every input position  $i$  with the class of the suffix from  $i$  by  $\leftarrow_f$  (Proposition 6). From this decomposition, one can construct an aperiodic bimachine whenever  $f$  is aperiodic. First, one shows that  $\mathcal{R}_f$  is aperiodic when  $f$  is too:

► **Proposition 5.** *Let  $f$  be a transduction realizable by an aperiodic transducer then the congruence  $\leftarrow_f$ , and so the automaton  $\mathcal{R}_f$ , are aperiodic.*

A right-sequential transducer is a transducer whose underlying input automaton is a right automaton. A right-sequential transduction is a function realized by a right-sequential transducer. A bimachine can be seen as the composition of a right-sequential transduction annotating the word with states of the right automaton, and a (left-) sequential transduction obtained from the left automaton and the output function  $\omega$ . We show that any aperiodic rational transduction  $f$  can be decomposed into  $\ell \circ \text{label}_{\mathcal{R}_f}$  such that  $\ell$  can be realized by a sequential aperiodic transducer, and  $\text{label}_{\mathcal{R}_f}$  annotates the input word with states of  $\mathcal{R}_f$ .

More precisely, let  $\mathcal{R}$  be a right automaton over  $\Sigma$  with set of states  $Q$ , and let  $\Sigma_{\mathcal{R}} = \{\sigma_q \mid \sigma \in \Sigma, q \in Q\}$ . We define the rational transduction  $\text{label}_{\mathcal{R}} : \Sigma^* \rightarrow \Sigma_{\mathcal{R}}^*$ , called the labelling function of  $\mathcal{R}$ , which labels words in  $\Sigma^*$  by states of  $\mathcal{R}$ . It is defined by the right-sequential transducer  $\mathcal{T} = (\mathcal{R}, \sigma, \bar{\epsilon}, \bar{\epsilon})$  where  $\bar{\epsilon}$  denotes the constant function which maps any element to  $\epsilon$ , and with  $\sigma(p, \sigma, q) = \sigma_q$ .

► **Proposition 6.** *Let  $f$  be an aperiodic rational transduction. There exists a transduction  $\ell$  such that  $f = \ell \circ \text{label}_{\mathcal{R}_f}$  and  $\ell$  is realized by a sequential aperiodic transducer.*

**Proof.** We first show that there exists a *sequential* transduction  $\ell$  such that  $f = \ell \circ \text{label}_{\mathcal{R}_f}$ . This sequential transduction is realized by the left automaton of the canonical bimachine  $\mathcal{B}_f$  combined with the output function  $\omega_f$ . Then, we show that  $\ell$  is *aperiodic* rational, by constructing an aperiodic transducer realizing it, obtained by taking the product of any aperiodic transducer realizing  $f$  (which exists by assumption) and  $\mathcal{R}_f$ , and by ensuring that the information  $[u]$  occurring on symbols  $\sigma_{[u]}$  is consistent with the information  $[u]$  occurring on the states of the product, for all words  $u$ . Finally, any aperiodic and sequential transduction can be realized by a transducer which is both sequential *and* aperiodic (*i.e.* sequentialization preserves aperiodicity [13]). Details can be found in the long version. ◀

We can now show our characterization of aperiodic rational transductions:

► **Theorem 7.** *A rational function  $f$  is aperiodic iff its canonical bimachine is aperiodic.*

**Proof.** It is known that any bimachine can be transformed into an equivalent (unambiguous) transducer whose underlying automaton is the product of the left and the right automata [13, 17]. Roughly, the transducer has to guess the state of the right automaton, and unambiguity is implied by the fact that the transitions of the right automaton are backward deterministic. The product of two aperiodic automata being aperiodic, this shows the 'if' direction.

We now show the 'only if' direction. By Proposition 5,  $f$  can be decomposed into  $f = \ell \circ \text{label}_{\mathcal{R}_f}$  such that  $\ell$  is realized by some aperiodic sequential transducer  $\mathcal{T}_m = (\mathcal{A}_m, \sigma_m, i_m, t_m)$ . Based on this decomposition we construct an aperiodic bimachine  $\mathcal{B} = (\mathcal{D}, \mathcal{R}_f, \omega, \lambda, \rho)$  realizing  $f$ . This will allow to conclude. Indeed, by Proposition 2 we have  $\mathcal{D} \sqsubseteq \text{Left}(\mathcal{R}_f)$ , and since  $\mathcal{D}$  is aperiodic, so is  $\text{Left}(\mathcal{R}_f)$ . Since  $\mathcal{R}_f$  is aperiodic as well by Proposition 5, it implies that  $\mathcal{B}_f$  is aperiodic. Let us now construct  $\mathcal{D}$ ,  $\omega$ ,  $\lambda$  and  $\rho$ .

## 13:12 Aperiodicity of Rational Functions Is PSPACE-Complete

Recall that the input alphabet of  $\mathcal{T}_m$  (and  $\mathcal{A}_m$ ) is  $\Sigma_Q$  where  $Q = \Sigma^*/\sphericalangle_f$ . To construct  $\mathcal{B}$ , it is tempting to think that it suffices to take the projection of  $\mathcal{A}_m$  on  $\Sigma$  as left automaton,  $\mathcal{R}_f$  as right automaton, and to define the output function  $\omega$  by  $\omega(p, \sigma, [u]) = w$  where  $p \xrightarrow{\sigma_{[u]}|w} q$  is the transition of  $\mathcal{T}_m$  on  $\sigma_{[u]}$  with output  $w$ . The problem is that the projection of  $\mathcal{A}_m$  on  $\Sigma$  is not a deterministic automaton in general, and by determinizing it, one loses the information of which transition of  $\mathcal{T}_m$  should be applied. We propose a solution that overcomes this issue, by integrating the information of  $\mathcal{R}_f$  in the state of the left automaton. Let us detail this construction. We take  $\mathcal{T}_m$  and project input letters to their  $\Sigma$  component, hence we obtain a transducer realizing  $f$  which is unambiguous, since  $\mathcal{R}_f$  has backward deterministic transitions. Let  $\tilde{\mathcal{T}}_m$  denote the obtained transducer, and  $\tilde{\mathcal{A}}_m$  its underlying (unambiguous) automaton. We let  $\mathcal{D}$  be the automaton obtained by determinization, by subset construction, of the product automaton  $\tilde{\mathcal{A}}_m \times \mathcal{R}_f$ . States of  $\mathcal{D}$  are therefore of the form  $2^{Q_m \times \Sigma^*/\sphericalangle_f}$ . The output function  $\omega$  is defined by:

$$\omega(\{(p_1, [u_1]), \dots, (p_n, [u_n])\}, \sigma, [v]) = o_m(p_i, \sigma)$$

such that  $[\sigma v] = [u_i]$ . The state  $p_i$  is unique since  $\mathcal{T}_m$  is unambiguous. Indeed, let us assume by contradiction that there are two distinct such states  $p_i, p_j$ . This would mean that  $[u_i] = [u_j]$  and since  $\mathcal{R}_f$  has backward deterministic transitions, for a word  $w$  which reaches both  $p_i$  and  $p_j$  in  $\tilde{\mathcal{A}}_m$ , we have a labelled word  $z$  such that  $z[k] = w[k]_c$  for  $k \in \{1, \dots, |w|\}$  and  $c$  the class of the word  $w[k+1] \dots w[|w|]u_i$ . Thus we obtain  $q_0 \xrightarrow{z}_{\mathcal{A}_m} p_i$  and  $q_0 \xrightarrow{z}_{\mathcal{A}_m} p_j$  which is in contradiction with the deterministic nature of  $\mathcal{A}_m$ . We define the final output functions naturally:  $\lambda([u]) = i_m(q_{o,m})$  and  $\rho(\{(p_1, [u_1]), \dots, (p_n, [u_n])\}) = \iota(p_i)$  such that  $[u_i] = [\epsilon]$  (again, it is unique by unambiguity of  $\mathcal{T}_m$ ).

It remains to show that  $\mathcal{B}$  is aperiodic. Aperiodicity of  $\mathcal{R}_f$  is obtained by Proposition 5. Aperiodicity of  $\mathcal{D}$  is shown in the long version of this paper, as a consequence of the aperiodicity of  $\mathcal{T}_m$ ,  $\mathcal{R}_f$  and the fact that subset construction preserves aperiodicity. ◀

► **Remark.** This theorem gives an algorithm to decide aperiodicity of a rational function: computing the canonical bimachine and checking that it is aperiodic. However, computing the left automaton  $Left(\mathcal{R}_f)$  may cause an exponential blow-up. Consider for example, the transduction  $f : \Sigma^* \rightarrow \Sigma^*$  defined for  $w \in \Sigma^*$ ,  $w_n \in \Sigma^n$  by  $f(w w_n) = w_n$  and for  $|w| < n$  by  $f(w) = w$ . Since the distance between the image of two words is bounded by  $2n$ , the left congruence of  $f$  is trivial, so the canonical bimachine of  $f$  is just a sequential transducer and needs  $O(\Sigma^n)$  states to remember the last  $n$  letters of an input word. However this transduction can be realized by a right-sequential transducer with only  $n$  states, but one could define a symmetrical example ( $f(w_n w) = w_n$ ) where the bimachine obtained from the canonical left automaton  $\mathcal{L}_f$  witnesses an exponential blow-up.

Another consequence is that any aperiodic transduction admits an aperiodic *unambiguous* transducer realizing it. This problem is open for arbitrary varieties.

► **Corollary 8.** *Every aperiodic transduction can be realized by an unambiguous aperiodic transducer.*

**Proof.** As explained in the proof of the ‘if’ direction of Theorem 7, from any bimachine one can construct an equivalent unambiguous transducer by taking the product of the left and right automata of the bimachine, which is aperiodic if the bimachine is aperiodic too. ◀



## 4.2 Characterization through bimachine minimization and main result

In this section, we prove the main result of this paper, *i.e.* that aperiodicity is PSPACE-complete for functions realized by bimachines. We show that for a bimachine  $\mathcal{B}$  it suffices to construct  $Left(Right(\mathcal{B}))$  (or  $Right(Left(\mathcal{B}))$ ) and to check aperiodicity of the resulting bimachine. The first step can be done in PTIME and the second step in PSPACE. The operations  $Left(Right(\mathcal{B}))$  and  $Right(Left(\mathcal{B}))$  are called *bimachine minimization*, as they indeed yield minimal bimachines, as shown by the following proposition:

► **Proposition 9.** *Let  $\mathcal{B}$  be a bimachine realizing a transduction  $f$ . Then  $Left(Right(\mathcal{B}))$  and  $Right(Left(\mathcal{B}))$  are minimal bimachines realizing  $f$ .*

**Proof.** Based on successive applications of Proposition 2 and given in the long version. ◀

The following result is a key towards the main contribution:

► **Proposition 10.** *Let  $f$  be a transduction realized by a bimachine  $\mathcal{B} = (\mathcal{L}, \mathcal{R}, \omega, \lambda, \rho)$ . Then  $f$  is aperiodic iff  $Left(Right(\mathcal{B}))$  is aperiodic iff  $Right(Left(\mathcal{B}))$  is aperiodic.*

**Proof.** First, we start by some observation: if there are two bimachines realizing  $f$  with  $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$  as right automata, then  $Left_f(\mathcal{A}_2) \sqsubseteq Left_f(\mathcal{A}_1)$ . Indeed, if  $\mathcal{A}_1$  provides more (*i.e.* finer) information than  $\mathcal{A}_2$  on suffixes, then the two equalities of the definition of the right congruence used to define  $Left_f(\mathcal{A}_1)$  (see Section 3.2) are “easier” to satisfy since the set of suffixes  $v$  taken into account in the definition of  $\hat{f}_{[w]\mathcal{A}_1}$  is included in the set of suffixes used in the definition of  $\hat{f}_{[w]\mathcal{A}_2}$  for all words  $w$ . Symmetrically, if there are two bimachines realizing  $f$  with  $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$  as left automata, then  $Right_f(\mathcal{A}_2) \sqsubseteq Right_f(\mathcal{A}_1)$ .

By Proposition 4 we have  $\mathcal{L} \sqsubseteq \mathcal{L}_f$  and  $\mathcal{R} \sqsubseteq \mathcal{R}_f$ , but also  $Right(\mathcal{L}) \sqsubseteq \mathcal{R}_f$  since Proposition 4 holds for any bimachine realizing  $f$ , and there is a bimachine realizing  $f$  with  $Right(\mathcal{L})$  as right automaton (the bimachine  $Right(\mathcal{B})$ ). Therefore by the observation, we get  $Right(\mathcal{L}_f) \sqsubseteq Right(\mathcal{L})$  and  $Left(\mathcal{R}_f) \sqsubseteq Left(Right(\mathcal{L}))$ .

By Theorem 7, if  $f$  is aperiodic, then  $\mathcal{B}_f = (Left(\mathcal{R}_f), \mathcal{R}_f, \omega, \lambda, \rho)$  is aperiodic. Therefore,  $Left(Right(\mathcal{L}))$  is aperiodic. Symmetrically, exactly as shown in Theorem 7, it can be shown that  $\mathcal{L}_f$  and  $Right(\mathcal{L}_f)$  are aperiodic if  $f$  is aperiodic, which implies that  $Right(\mathcal{L})$  is aperiodic. In conclusion,  $Left(Right(\mathcal{B}))$  is aperiodic. ◀

We can now prove the main result of this paper:

► **Theorem 11.** *The problem of deciding whether a bimachine  $\mathcal{B}$  realizes an aperiodic rational transduction is PSPACE-complete.*

**Proof.** To get the upper-bound, by Proposition 10, it suffices to (i) construct  $Right(\mathcal{B})$ , (ii) construct  $Left(Right(\mathcal{B}))$ , and (iii) test whether the left and right automata of the bimachine  $Left(Right(\mathcal{B}))$  are aperiodic.

By Theorem 3, steps (i) and (ii) can be done in PTIME, while step (iii) can be done in PSPACE by [5].

The lower bound is obtained from the problem of deciding whether the transition congruence of a minimal deterministic finite automaton is aperiodic, which is PSPACE-hard [5]. The details can be found in the long version of this paper. ◀

## 5 Perspectives

In [8] it is proved that deciding whether a regular language given by a *non-deterministic* automaton is aperiodic is also PSPACE-complete. As a future work, we want to obtain tight complexity for the following problem: given a (non-deterministic) transducer, does it define an aperiodic transduction? Based on the techniques of this paper, the latter problem could be shown to be in  $2^{\text{EXPTIME}}$ , since obtaining the canonical bimachine causes two exponential blow-ups: one for the canonical right automaton and one for the determinization of the transducer over the enriched alphabet. It is however yet unclear whether the techniques of [8] can be combined with the ones of this paper to lower this upper bound.

---

### References

- 1 Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theor. Comput. Sci.*, 292(1):45–63, 2003.
- 2 Jean Berstel and Luc Boasson. Transductions and context-free languages. *Ed. Teubner*, pages 1–278, 1979.
- 3 Mikołaj Bojańczyk. Transducers with origin information. In *International Colloquium on Automata, Languages, and Programming (ICALP), Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 2014. doi:10.1007/978-3-662-43951-7\_3.
- 4 Olivier Carton and Luc Dartois. Aperiodic two-way transducers and fo-transductions. In *EACSL Annual Conference on Computer Science Logic (CSL)*, volume 41 of *LIPICs*, pages 160–174. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPICs.CSL.2015.160.
- 5 Sang Cho and Dung T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theor. Comput. Sci.*, 88(1):99–116, 1991. doi:10.1016/0304-3975(91)90075-D.
- 6 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(1):131–143, 2003. doi:10.1016/S0304-3975(01)00219-5.
- 7 Bruno Courcelle. Monadic second-order definable graph transductions: A survey. *Theor. Comput. Sci.*, 126(1):53–75, 1994. doi:10.1016/0304-3975(94)90268-2.
- 8 Volker Diekert and Paul Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.
- 9 Samuel Eilenberg. *Automata, Languages, and Machines. Volume A*. Pure and Applied Mathematics. Academic press, 1974.
- 10 Calvin C. Elgot and Jorge E. Mezei. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965. doi:10.1147/rd.91.0047.
- 11 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001. doi:10.1145/371316.371512.
- 12 Emmanuel Filiot. Logic-automata connections for transformations. In *Indian Conference on Logic and Its Applications (ICLA)*, volume 8923 of *Lecture Notes in Computer Science*, pages 30–57. Springer, 2015. doi:10.1007/978-3-662-45824-2\_3.
- 13 Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. First-order definability of rational transductions: An algebraic approach. In *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 2016.
- 14 Emmanuel Filiot, Shankara Narayanan Krishna, and Ashutosh Trivedi. First-order definable string transformations. In *International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 29 of *LIPICs*, pages 147–159.

- Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.147.
- 15 John E. Hopcroft. An  $N \log N$  Algorithm for Minimizing States in a Finite Automaton. Technical report, Stanford University, Stanford, CA, USA, 1971.
  - 16 Robert McNaughton and Seymour Papert. *Counter-free automata*. M.I.T. Press, 1971.
  - 17 Christophe Reutenauer and Marcel-Paul Schützenberger. Minimization of rational word functions. *SIAM J. Comput.*, 20(4):669–685, 1991. doi:10.1137/0220042.
  - 18 Christophe Reutenauer and Marcel-Paul Schützenberger. Variétés et fonctions rationnelles. *Theor. Comput. Sci.*, 145(1&2):229–240, 1995. doi:10.1016/0304-3975(94)00180-Q.
  - 19 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
  - 20 Marcel-Paul Schützenberger. A remark on finite transducers. *Information and Control*, 4(2-3):185–196, 1961.
  - 21 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.
  - 22 Jacques Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985. doi:10.1016/S0019-9958(85)80058-9.
  - 23 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1994.



# Homomorphism Problems for First-Order Definable Structures\*

Bartek Klin<sup>1</sup>, Sławomir Lasota<sup>2</sup>, Joanna Ochremiak<sup>3</sup>, and Szymon Toruńczyk<sup>4</sup>

- 1 University of Warsaw, Warsaw, Poland
- 2 University of Warsaw, Warsaw, Poland
- 3 Universitat Politècnica de Catalunya, Barcelona, Spain
- 4 University of Warsaw, Warsaw, Poland

---

## Abstract

We investigate several variants of the homomorphism problem: given two relational structures, is there a homomorphism from one to the other? The input structures are possibly infinite, but definable by first-order interpretations in a fixed structure. Their signatures can be either finite or infinite but definable. The homomorphisms can be either arbitrary, or definable with parameters, or definable without parameters. For each of these variants, we determine its decidability status.

**1998 ACM Subject Classification** F.4.1 [Mathematical Logic] Model theory, F.4.3 [Formal Languages] Decision problems

**Keywords and phrases** Sets with atoms, first-order interpretations, homomorphism problem

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.14

## 1 Introduction

First-order definable sets, although usually infinite, can be finitely described and are therefore amenable to algorithmic manipulation. Definable sets (we drop the qualifier *first-order* in what follows) are parametrized by a fixed underlying relational structure  $\mathcal{A}$  whose elements are called *atoms*. We shall assume that the first-order theory of  $\mathcal{A}$  is decidable. To simplify the presentation, unless stated otherwise, let  $\mathcal{A}$  be a countable set  $\{1, 2, 3, \dots\}$  equipped with the equality relation only; we shall call this the *pure set*.

► **Example 1.** Let

$$V = \{ \{a, b\} \mid a, b \in \mathcal{A}, a \neq b \},$$
$$E = \{ (\{a, b\}, \{c, d\}) \mid a, b, c, d \in \mathcal{A}, a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d \}.$$

Both  $V$  and  $E$  are definable sets (over  $\mathcal{A}$ ), as they are constructed from  $\mathcal{A}$  using (possibly nested) set-builder expressions with first-order guards ranging over  $\mathcal{A}$ . In general, we allow finite unions in the definitions, and finite tuples (as above) are allowed for notational convenience. Precise definitions are given in Section 2. The pair  $G = (V, E)$  is also a definable set, in fact, a definable graph. It is an infinite Kneser graph (a generalization of the famous Petersen graph): its vertices are all two-element subsets of  $\mathcal{A}$ , and two such subsets are adjacent iff they are disjoint.

---

\* This work is supported by Poland's National Science Centre grant 2012/07/B/ST6/01497, and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement ERC-2014-CoG 648276 AUTAR).



The graph  $G$  is  $\emptyset$ -definable: its definition does not refer to any particular elements of  $\mathcal{A}$ . In general, one may refer to a finite set of parameters  $S \subseteq \mathcal{A}$  to describe an  $S$ -definable set. For instance, the set  $\{a \mid a \in \mathcal{A}, a \neq \underline{1} \wedge a \neq \underline{2}\}$  is  $\{\underline{1}, \underline{2}\}$ -definable. Definable sets are those which are  $S$ -definable for some finite  $S \subseteq \mathcal{A}$ .

Although definable relational structures correspond (up to isomorphism) to first-order interpretations well-known from logic and model theory [19], we prefer to use a different definition since standard set-theoretic notions directly translate into this setting. For example, a definable function  $f : X \rightarrow Y$  is simply a function whose domain  $X$ , codomain  $Y$ , and graph  $\Gamma(f) \subseteq X \times Y$  are definable sets. A relational structure is definable if its universe, signature, and interpretation function that maps each relation symbol to a relation on the universe, are definable. Finally, a definable homomorphism between definable structures over the same signature is a definable mapping between their universes that is a homomorphism, i.e., preserves every relation in the signature. All hereditarily finite sets (finite sets, whose elements are finite, and so on, recursively) are definable, and every finite relational structure over a finite signature is (isomorphic to) a definable one.

The classical *homomorphism problem* is the problem of determining whether there exists a homomorphism from a given finite source structure  $\mathbb{A}$  to a given finite target structure  $\mathbb{B}$ . This is also known as the Constraint Satisfaction Problem, and is clearly decidable (and NP-complete). The precise computational complexity has been thoroughly studied in the literature in many variants. The case when the target structure is fixed (and is called a *template*) is of particular interest, as it expresses many natural computational problems (such as  $k$ -colorability, 3-SAT, or solving systems of linear equations over a finite field). The famous Feder-Vardi conjecture states that for every fixed template  $\mathbb{B}$ , the corresponding constraint satisfaction problem  $\text{CSP}(\mathbb{B})$  is either solvable in polynomial time or NP-complete [18].

In this paper, we consider the homomorphism problem for definable structures: given two definable structures  $\mathbb{A}, \mathbb{B}$ , does there exist a homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$ ? Note that definable structures can be meaningfully considered as instances of a computational problem since they are finitely described with the set-builder notation and first-order formulas in the language of  $\mathcal{A}$ .

We remark that in the pure set  $\mathcal{A}$  with equality, every first-order formula is effectively equivalent to a quantifier-free formula. Thus, as long as complexity issues are ignored and decidability is the only concern, we can safely restrict to quantifier-free formulas.

► **Example 2.** The graph  $G$  from Example 1 does not map homomorphically to a clique of 3 vertices, which is another way of saying that  $G$  is not 3-colorable. In fact,  $G$  does not map homomorphically to any finite clique (the finite subgraph of  $G$  using only atoms  $\underline{1}, \dots, \underline{2n}$  has chromatic number at least  $n$ , as it contains an  $n$ -clique). However,  $G$  maps homomorphically to the (easily definable) infinite clique on the set  $\mathcal{A}$ , by any injective mapping from  $V$  to  $\mathcal{A}$ . No such homomorphism is definable, as there is no definable injective function from  $V$  to  $\mathcal{A}$ , even with parameters.

We consider several variants of the homomorphism problem:

- *Finite vs. infinite signature.* In the most general form, we allow the signature of both input structures to be infinite, but definable. In a restricted variant, the signature is required to be finite.
- *Finite vs. infinite structures.* In general, both input structures can be infinite, definable. In a restricted variant, one of the two input structures may be assumed to be finite.
- *Definability of homomorphisms.* In the general setting, we ask the question whether there exists an arbitrary homomorphism between the input structures. In other variants, the homomorphism is required to be definable, or to be  $\emptyset$ -definable.

- *Restrictions on homomorphisms.* Most often we ask about any homomorphism, but one may also ask about existence of a homomorphism that is injective, strong, or an embedding.
- *Fixing one structure.* In the uniform variant, both the source and the target structures are given on input. We also consider non-uniform variants, when one of the two structures is fixed.
- *Structured atoms.* In the basic setting, the underlying structure  $\mathcal{A}$  is the pure set, i.e., has no structure other than equality. One can also consider sets definable over other structures. For instance, if the underlying structure is  $(\mathbb{Q}, \leq)$ , the definitions of definable sets can refer to the relation  $\leq$ .

**Contribution.** For most combinations of these choices we determine the decidability status of the homomorphism problem. The decidability border turns out to be quite subtle and sometimes counterintuitive. The following theorem samples some of the opposing results proved in this paper:

► **Theorem 3.** *Let  $\mathcal{A}$  be the pure set. Given two definable structures  $\mathbb{A}, \mathbb{B}$  over a finite signature,*

1. *it is decidable whether there is a  $\emptyset$ -definable homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$ ,*
2. *it is undecidable (but semidecidable) whether there is a definable homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$ ,*
3. *it is decidable whether there is a homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$ ,*
4. *it is undecidable (but co-semidecidable) whether a given  $\emptyset$ -definable partial mapping between the universes of  $\mathbb{A}$  and  $\mathbb{B}$  extends to a homomorphism.*

In a previous paper [21], the constraint satisfaction problem is considered for source structures definable over the pure set, or more generally over  $(\mathbb{Q}, \leq)$ . We denote this problem by  $\text{CSP}_{\text{def}}(\mathbb{B})$ . The results from [21], together with the polynomial time reduction to the finite-template CSP which we provide here, imply complexity results for different variants of the constraint satisfaction problem:

► **Theorem 4.** *For any definable template  $\mathbb{B}$  over a finite signature:*

1. *the problem  $\text{CSP}(\mathbb{B})$  is in NP,*
2. *the problem  $\text{CSP}_{\text{def}}(\mathbb{B})$  is in NEXPTIME.*

**Related work.** Some of the variants considered in this paper are closely related to previous work.

Bodirsky, Pinsker and coauthors [2, 8, 10] consider fixed infinite templates over finite signatures, and finite source structures given on input. They usually consider the template  $\mathbb{B}$  to be a reduct of a fixed structure  $\mathcal{A}$  with good properties, in particular, with a decidable first-order theory. Reducts are special cases of definable structures: a structure  $\mathbb{B}$  is a reduct of  $\mathcal{A}$  if  $\mathbb{B}$  is  $\emptyset$ -definable over  $\mathcal{A}$  and both have the same domains. In general, if the template  $\mathbb{B}$  is definable over a structure  $\mathcal{A}$  with decidable first-order theory, then  $\mathbb{B}$  itself has decidable first-order theory. It follows that the existence of a homomorphism from a given *finite* source structure  $\mathbb{A}$  is trivially decidable, as it can be expressed as an existential formula evaluated in  $\mathbb{B}$ . In this case, the interesting problem is to analyse precise complexity bounds. Templates for which a complete complexity classification was obtained (modulo the Feder-Vardi conjecture) include all reducts of countably infinite homogeneous graphs [3, 9, 12, 6], of  $(\mathbb{Q}, \leq)$  [4], and of the integers with the successor function  $(\mathbb{Z}, +1)$  [5]. One of the key tools used in these results is the notion of a *canonical mapping*. The construction of a canonical

mapping relies on Ramsey theory, most conveniently applied through the use of the result of Kechris, Pestov, and Todorćević concerning extremely amenable groups [20].

For finite templates, it is shown in [21] that the complexity analysis of  $\text{CSP}_{\text{def}}(\mathbb{B})$  can be reduced (with an exponential blowup) to the case of finite input structures. For example, 3-colorability of definable graphs is decidable and NEXPTIME-complete, because 3-colorability of finite graphs is NP-complete. A more general decidability result concerns *locally finite templates*, i.e., definable, possibly infinite templates (over definable, possibly infinite signatures) where every relation contains only finitely many tuples. The decidability proof also employs Ramsey theory, applied through the use of Pestov's theorem concerning the topological dynamics of the group  $\text{Aut}(\mathbb{Q}, \leq)$ , which is a special case of the Kechris-Pestov-Todorćević result. As we shall demonstrate here, for infinite signatures the local finiteness restriction is crucial and adding even a single infinite definable relation may lead to undecidability.

This paper, as well as [21], is part of a programme aimed at generalizing classical decision problems and computation models such as automata [15], Turing machines [16] and programming languages [14, 13, 22, 24], to sets with atoms. For other applications of sets with atoms (called there *nominal sets*) in computing, see [26].

**Motivation.** Testing existence of homomorphisms is at the core of many decision problems in combinatorics and logic. As shown in [11], decidability of pp-definability of a definable relation  $R$  in a definable structure  $\mathbb{A}$  can be reduced to deciding the existence of homomorphisms between definable structures. Another application is 0-1 laws, and deciding whether a sentence  $\phi$  of the form  $\exists R. \exists^* \forall^* \psi$  is satisfied with high probability in a finite random graph. In [23], after showing that the problem is equivalent to testing if  $\phi$  holds in the infinite random graph, the authors give a complex Ramsey argument based on [25] to prove the decidability of the latter. The second step can be alternatively achieved by reducing to several instances of the homomorphism problem from structures definable over the ordered random graph (which is a Ramsey structure by [25], see Section 5) to finite target structures. Finally, in [21] the homomorphism problem for locally finite definable templates is used to test whether the logic IFP captures PTime over a certain class of finite structures, generalizing the Cai-Fürer-Immerman construction [17].

## 2 Preliminaries

Throughout this section, fix a countable relational structure  $\mathcal{A}$  of *atoms*. We assume that the signature of  $\mathcal{A}$  is finite. We shall now introduce definable sets, following [21].

**Definable sets.** An *expression* is either a variable from some fixed infinite set, or a formal finite union (including the empty union  $\emptyset$ ) of *set-builder expressions* of the form

$$\{ e \mid a_1, \dots, a_n \in \mathcal{A}, \phi \}, \quad (1)$$

where  $e$  is an expression,  $a_1, \dots, a_n$  are (bound) variables, and  $\phi$  is a first-order formula over the signature of  $\mathcal{A}$  and over the set of variables. Free variables in (1) are those free variables of  $e$  and of  $\phi$  which are not among  $a_1, \dots, a_n$ .

For an expression  $e$  with free variables  $V$ , any valuation  $\text{val} : V \rightarrow \mathcal{A}$  defines in an obvious way a value  $X = e[\text{val}]$ , which is either an atom or a set, formally defined by induction on the structure of  $e$ . We then say that  $X$  is a *definable set with atoms*, and that it is *defined* by  $e$  with  $\text{val}$ . Note that one set  $X$  can be defined by many different expressions. When we



want to emphasize those atoms that are in the image of the valuation  $\text{val} : V \rightarrow \mathcal{A}$ , we say that the finite set  $S = \text{val}(V) \subseteq \mathcal{A}$  *supports*  $X$ , or that  $X$  is *S-definable*.

As syntactic sugar, we allow atoms to occur directly in set expressions. For example, what we write as the  $\{\underline{1}\}$ -definable set  $\{a \mid a \in \mathcal{A}, a \neq \underline{1}\}$  is formally defined by the expression  $\{a \mid a \in \mathcal{A}, a \neq b\}$ , together with a valuation mapping  $b$  to  $\underline{1}$ . Similarly, the set  $\{\underline{1}, \underline{2}\}$  is  $\{\underline{1}, \underline{2}\}$ -definable as a union of two singleton sets.

► **Remark.** To improve readability, it will be convenient to use standard set-theoretic encodings to allow a more flexible syntax. In particular, ordered pairs and tuples can be encoded e.g. by Kuratowski pairs,  $(x, y) = \{\{x, y\}, \{x\}\}$ . We will also consider as definable infinite families of symbols, such as  $\{R_x : x \in X\}$ , where  $R$  is a symbol and  $X$  is a definable set. Formally, such a family can be encoded as the set of ordered pairs  $\{R\} \times X$ , where the symbol  $R$  is represented by some  $\emptyset$ -definable set, e.g.  $\emptyset$  or  $\{\emptyset\}$ , etc. Here we use the fact that definable sets are closed under Cartesian products.

**Closure properties.** The following lemma is proved routinely by induction on the nesting of set-builder expressions.

► **Lemma 5.** *Definable sets are effectively closed under:*

- Boolean combinations  $\cap, \cup, -$  and Cartesian products,
- images and inverse images under definable functions,
- quotients under definable equivalence relations,
- intersections and unions of definable families,
- the operations (below,  $x \in y$  and  $x \subseteq y$  are interpreted as false if  $y$  is an atom):  
 $V, W \mapsto \{(v, w) \mid v \in V, w \in W, v \in w\},$   
 $V, W \mapsto \{(v, w) \mid v \in V, w \in W, v \subseteq w\}.$

This implies that the set-builder notation (1) may be safely generalized by allowing bound variables to range not only over  $\mathcal{A}$  but also over other definable sets, and allowing in  $\phi$  quantifiers of the form  $\exists v \in V$  or  $\forall v \in V$ , for  $V$  a definable set presented by an expression. One may also use binary predicates  $=, \in, \subseteq$  and binary operations  $\cup, \cap, -, \times$ . The resulting sets will still be definable. As an example, if  $V$  and  $W$  are definable sets, then so is

$$\{(v, w) \mid v \in V, w \in W, v \subseteq w \wedge \exists a \in \mathcal{A} \exists b \in \mathcal{A} (a, b) \in v\}.$$

Suppose that the first-order theory of  $\mathcal{A}$  is decidable (this applies in particular to the pure set). Then it is straightforward to prove that the validity of first-order sentences generalized as above, such as  $\forall v \in V \exists w \in W v \subseteq w$  where  $V$  and  $W$  are definable sets presented by expressions, is also decidable. This demonstrates that definable sets are suitable for effectively performing set-theoretic manipulations and tests.

**Definable relational structures.** For any object in the set-theoretic universe (a relation, a function, a logical structure, etc.), it makes sense to ask whether it is definable. For example, a definable relation on  $X, Y$  is a relation  $R \subseteq X \times Y$  which is a definable set of pairs, and a definable function  $X \rightarrow Y$  is a function whose graph is definable. Along the same lines, a definable relational signature is a definable set of *symbols*  $\Sigma$ , together with a partition  $\Sigma = \Sigma_1 \uplus \Sigma_2 \uplus \dots \uplus \Sigma_l$  into definable subsets, for  $l \in \mathbb{N}$ . We say that  $\sigma$  has *arity*  $r$  if  $\sigma \in \Sigma_r$ .

For a signature  $\Sigma$ , a definable  $\Sigma$ -structure  $\mathbb{A}$  consists of a definable universe  $A$  and a definable interpretation function which assigns a relation  $\sigma^{\mathbb{A}} \subseteq A^r$  to each relation symbol  $\sigma \in \Sigma$  of arity  $r$ . (We denote structures using blackboard font, and their universes using the corresponding symbol in italics). More explicitly, such a structure can be represented

## 14:6 Homomorphism Problems for First-Order Definable Structures

by the tuple  $\mathbb{A} = (A, I_1, \dots, I_l)$  where  $I_r = \{(\sigma, a_1, \dots, a_r) \mid \sigma \in \Sigma_r, (a_1, \dots, a_r) \in \sigma^{\mathbb{A}}\}$  is a definable set for  $r = 1, \dots, l$  (where  $l$  is the maximal arity in  $\Sigma$ ).

► **Remark.** A definable  $\Sigma$ -structure  $\mathbb{A} = (A, I_1, \dots, I_l)$ , for  $\Sigma$  finite or infinite, can be seen as a definable structure over a finite signature, denoted  $\mathbb{A}_\Sigma$  and defined as follows. The universe of  $\mathbb{A}_\Sigma$  is  $A \uplus \Sigma$ , and its relations are  $I_1, \dots, I_l$ , of arity  $2, \dots, l+1$ , respectively. The signature is finite, with just  $l$  symbols. Then homomorphisms between  $\Sigma$ -structures  $\mathbb{A}$  and  $\mathbb{B}$  correspond to those homomorphisms between  $\mathbb{A}_\Sigma$  and  $\mathbb{B}_\Sigma$  that are the identity on  $\Sigma$ .

► **Example 6.** The graph  $G$  from Example 1 can be seen as a structure over a signature with a single binary relation symbol  $E$ . To give an example of an infinite, definable signature, extend  $G$  to a structure  $\mathbb{A}$  by infinitely many unary predicates representing the neighborhoods of each vertex of  $G$ . To this end, define the signature  $\Sigma = \{E\} \cup \{N_v \mid v \in V\}$ , where  $V$  is the vertex set of  $G$  and  $N$  is a symbol (cf. Remark 2). The interpretation of  $N_v$  is specified by the set  $I_1 = \{(N_v, w) \mid (v, w) \in E\}$  (where  $E$  is defined by the expression from Example 1), which is definable by Lemma 5.

► **Lemma 7.** *For every  $S$ -definable set  $X$  there is an  $S$ -definable surjective function  $f : Y \rightarrow X$ , where  $Y$  is an  $S$ -definable subset of  $\mathcal{A}^k$ , for some  $k \in \mathbb{N}$ . Moreover,  $f$  and  $Y$  can be computed from  $X$ .*

► **Remark.** By Lemma 7, definable structures over finite signatures coincide, up to definable isomorphism, with structures that admit a *first-order interpretation with parameters* in  $\mathcal{A}$ , in the sense of model theory [19].

**Representing the input.** Definable relational structures can be input to algorithms, as they are finitely presented by expressions defining the signature, the universe, and the interpretation function. If the input is an  $S$ -definable set  $X$ , defined by an expression  $e$  with valuation  $\text{val} : V \rightarrow S$  with  $V = \{v_1, \dots, v_n\}$  the free variables of  $e$ , then we also need to represent the tuple  $\text{val}(v_1), \dots, \text{val}(v_n)$  of elements of  $S$ . For the pure set  $\mathcal{A}$ , these elements can be represented as  $\underline{1}, \underline{2}, \dots$

### 3 Homomorphism problems

To simplify the presentation, we now drop some of the generality of the previous section. In this section let  $\mathcal{A}$  be the pure set. In Section 5 we shall discuss generalizations of our results to underlying structures other than the pure set.

#### 3.1 $\emptyset$ -definable homomorphism problem

Let's start with the following warm-up decision problem:

**Problem:**  $\emptyset$ -DEFINABLE HOMOMORPHISM

**Input:**  $\emptyset$ -definable structures  $\mathbb{A}$  and  $\mathbb{B}$  over a  $\emptyset$ -definable signature  $\Sigma$ .

**Decide:** Is there an  $\emptyset$ -definable homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$ ?

It is not hard to prove the following theorem, which gives (1) of Theorem 3:

► **Theorem 8** ([21]).  $\emptyset$ -DEFINABLE HOMOMORPHISM *is decidable.*

We sketch a proof here in order to illustrate the good algorithmic properties of definable sets, and to emphasize the contrast with later undecidability results.

**Proof sketch.** Our aim is to decide if two given  $\emptyset$ -definable  $\Sigma$ -structures  $\mathbb{A} = (A, I_1, \dots, I_l)$  and  $\mathbb{B} = (B, J_1, \dots, J_l)$  admit an  $\emptyset$ -definable homomorphism. The signature  $\Sigma$  is assumed to be part of the input (also, it can be computed from  $\mathbb{A}$  or from  $\mathbb{B}$ ).

We will use the following facts that hold for the pure set  $\mathcal{A}$ , but also for many other structures with decidable first-order theories.

► **Lemma 9.** *For each number  $n \in \mathbb{N}$ , there are finitely (doubly exponentially) many first-order formulas with  $n$  free variables, up to equivalence in  $\mathcal{A}$ . Moreover, they can be computed from  $n$ .*

The following lemma is a consequence.

► **Lemma 10.** *An  $\emptyset$ -definable set  $X$  has only finitely many  $\emptyset$ -definable subsets, and expressions defining these subsets can be enumerated from an expression defining  $X$ .*

Indeed, for each definable set  $X$  represented by a single set-builder expression of the form (1), replace  $\phi$  by each (up to equivalence) quantifier-free formula  $\psi$  with the same free variables, such that  $\psi \rightarrow \phi$ .

To verify existence of an  $\emptyset$ -definable homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$ , apply Lemma 10 to  $X = A \times B$  and for every  $\emptyset$ -definable subset  $R \subseteq A \times B$ , test the validity of the first-order formula

$$\forall a \in A \exists! b \in B \ R(a, b)$$

ensuring that  $R$  is a graph of a function; and, for  $i = 1 \dots l$ , test the validity of the formula

$$\forall a_1, \dots, a_i \in A \forall b_1, \dots, b_i \in B \forall \rho \in \Sigma_i \bigwedge_{1 \leq j \leq i} R(a_j, b_j) \wedge I_i(\rho, a_1, \dots, a_i) \rightarrow J_i(\rho, b_1, \dots, b_i)$$

ensuring that the function is a homomorphism. ◀

In a similar vein one can decide the existence of homomorphisms that are injective, strong, or are embeddings (i.e. injective and strong), as all these properties are first-order definable.

The assumption that the structures  $\mathbb{A}$  and  $\mathbb{B}$  are  $\emptyset$ -definable is inessential in Theorem 8; the crucial assumption is that a homomorphism we ask for is required to be  $\emptyset$ -definable. In fact, a similar argument as above works even if the two given structures are definable instead of  $\emptyset$ -definable, and a homomorphism is allowed to be definable with  $n$  parameters, for a number  $n \in \mathbb{N}$  given on input.

### 3.2 (Definable) homomorphism problem

In more relaxed versions of the homomorphism problem, we ask for a homomorphism that is definable without any bound on the number of parameters:

**Problem:** DEFINABLE HOMOMORPHISM

**Input:** Definable structures  $\mathbb{A}$  and  $\mathbb{B}$  over a definable signature  $\Sigma$ .

**Decide:** Is there a definable homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$ ?

Or we may make no restriction on a homomorphism at all:

**Problem:** HOMOMORPHISM

**Input:** Definable structures  $\mathbb{A}$  and  $\mathbb{B}$  over a definable signature  $\Sigma$ .

**Decide:** Is there a homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$ ?

These problems appear similar, but they are of rather different nature. On one hand, DEFINABLE HOMOMORPHISM is recursively enumerable, by an argument similar to the proof sketch of Theorem 8: if a definable homomorphism exists then one can find it by searching for homomorphisms definable with  $n$  parameters, for increasing values of  $n$ . On the other hand HOMOMORPHISM is co-recursively enumerable, by a compactness argument: if  $\mathbb{A}$  does *not* map homomorphically to  $\mathbb{B}$  then some finite substructure of  $\mathbb{A}$  does not map to  $\mathbb{B}$  either, and one can detect this by enumerating all finite substructures of  $\mathbb{A}$  and using Theorem 11 below.

► **Remark.** We might also consider natural variants of (DEFINABLE) HOMOMORPHISM, where one asks about existence of an injective homomorphism, or a strong homomorphism, or an embedding. Theorems 11–16, stated below, apply to all these variants as well.

Below we show that both DEFINABLE HOMOMORPHISM and HOMOMORPHISM are undecidable in general. However, when one of the input structures has finite universe, both problems are decidable:

► **Theorem 11.** DEFINABLE HOMOMORPHISM and HOMOMORPHISM are decidable if one of the input structures has a finite universe.

On the other hand, the general version of the homomorphism problem is undecidable:

► **Theorem 12.** HOMOMORPHISM is undecidable, even if one of the input structures is fixed.

The fixed input structure is understood existentially; in particular, there exists a definable structure  $\mathbb{B}$  such that it is undecidable, for a given definable structure  $\mathbb{A}$  over the same signature, whether there is a homomorphism  $\mathbb{A} \rightarrow \mathbb{B}$ .

Theorem 12 is proved by a reduction from a classical quarter-plane tiling problem [1]. The following example illustrates a phenomenon used in the proof: a homomorphism can determine an infinite ordered sequence of atoms, and thus to enumerate coordinates within the quarter-plane.

► **Example 13.** Consider a signature with a single binary relation symbol  $R$ . For a chosen atom  $a_0 \in \mathcal{A}$ , define structures  $\mathbb{A}$  and  $\mathbb{B}$  over this signature as follows:

$$A = \mathcal{A} \quad R^{\mathbb{A}} = \neq \quad B = \mathcal{A} - \{a_0\} \quad R^{\mathbb{B}} = \neq$$

Note that  $\mathbb{A}$  is  $\emptyset$ -definable and  $\mathbb{B}$  is  $\{a_0\}$ -definable. Considered as graphs,  $\mathbb{A}$  and  $\mathbb{B}$  are isomorphic to the countably infinite clique. However, no homomorphism  $h : \mathbb{A} \rightarrow \mathbb{B}$  is definable. To see this, suppose towards contradiction that an  $S$ -definable homomorphism  $h$  actually exists for some finite  $S$ . We will exploit the fact that the  $S$ -definition of  $h$  is necessarily invariant under every bijection  $\pi$  of atoms such that  $\pi(a) = a$  for all  $a \in S$ .

Since  $\mathbb{A}$  is a clique and  $\mathbb{B}$  has no self-loops,  $h$  must be injective. Pick the atom  $a_1 = h(a_0)$ . Clearly  $a_1 \neq a_0$ , since  $a_0 \notin \mathbb{B}$ . This means that  $a_1 \in S$ ; indeed, if  $a_1 \notin S$  then the  $S$ -definition of (the graph of)  $h$  would be invariant under a renaming  $\pi$  of atoms with  $\pi(a_0) = a_0$  and  $\pi(a_1) \neq a_1$ , which cannot be since  $h$  is a function. Now consider  $a_2 = h(a_1)$ . Again,  $a_2 \neq a_0$ . Moreover we have  $a_2 \neq a_1$ , since  $a_1 \neq a_0$  and  $h$  is injective. Moreover,  $a_2 \in S$  by the same argument as for  $a_1$ . This proceeds by induction, showing that infinitely many distinct atoms must belong to  $S$ , which contradicts the finiteness of  $S$ .

More importantly, each homomorphism  $h : \mathbb{A} \rightarrow \mathbb{B}$  determines an infinite sequence of distinct atoms  $a_0, a_1, a_2, \dots$  such that  $h(a_i) = a_{i+1}$  for each  $i \in \mathbb{N}$ .

As it turns out, DEFINABLE HOMOMORPHISM is even harder to decide than HOMOMORPHISM:

► **Theorem 14.** DEFINABLE HOMOMORPHISM *is undecidable even if*

- (i) *a source structure  $\mathbb{A}$  over a finite signature is fixed; or*
- (ii) *a target structure  $\mathbb{B}$  is fixed.*

Theorem 14 yields (2) of Theorem 3, and is proved by reduction from periodic and ultimately periodic variants of the tiling problem.

Example 13 shows a situation where definable homomorphisms do not exist, but non-definable ones do, and each of them induces an infinite sequence of atoms. In the following example definable homomorphisms do exist, and each of them determines a finite cycle of atoms. This observation is the core of the proof of Theorem 14, much as Example 13 is the core of Theorem 12.

► **Example 15.** Consider a signature with a single binary relation symbol  $R$ . Define structures  $\mathbb{A}$  and  $\mathbb{B}$  over this signature as follows (for readability we write  $ab$  to denote an ordered pair  $(a, b)$ ):

$$\begin{aligned} A &= \mathcal{A} & B &= \{ab \mid a, b \in \mathcal{A}, a \neq b\} \\ R^{\mathbb{A}} &= \neq & R^{\mathbb{B}} &= \{(ab, cd) \mid a, b, c, d \in \mathcal{A}, a \neq b, c \neq d, a \neq c\} \end{aligned}$$

Note that there are many non-definable homomorphisms from  $\mathbb{A}$  to  $\mathbb{B}$ . For example, for any enumeration  $a_0, a_1, a_2, \dots$  of all atoms, one may put  $h(a_n) = a_n a_{n+1}$  for each  $n \in \mathbb{N}$ .

However, definable homomorphisms  $h : \mathbb{A} \rightarrow \mathbb{B}$  also exist. For example, there is an  $S$ -definable one for  $S = \{\underline{1}, \underline{2}, \underline{3}\}$ :

$$h(x) = x\underline{1} \quad h(\underline{1}) = \underline{1}\underline{2} \quad h(\underline{2}) = \underline{2}\underline{3} \quad h(\underline{3}) = \underline{3}\underline{1}$$

where  $x \notin S$ . Note how the values of  $h$  on  $S$  encode a cycle of atoms of length 3. This is a general phenomenon. Indeed, consider any  $S$ -definable homomorphism  $h : \mathbb{A} \rightarrow \mathbb{B}$ , for some finite  $S = \{a_1, \dots, a_n\} \subseteq \mathcal{A}$ . Denote  $e_i = h(a_i)$  for  $i = 1..n$ . Each  $e_i$  is of the form  $a_j a_k$  for some  $1 \leq j \neq k \leq n$ . Indeed, if some  $e_i = bc$  (or  $e_i = cb$ ) for some  $b \notin S$ , then the  $S$ -definition of (the graph of)  $h$  would be invariant under a renaming  $\pi$  of atoms with  $\pi(a_i) = a_i$  and  $\pi(b) \neq b$ , which cannot be since  $h$  is a function.

One may view the  $e_i$  as edges of a directed graph with nodes  $\{a_1, \dots, a_n\}$ . This graph has  $n$  nodes,  $n$  edges, no self-loops, and, looking at the definition of  $R^{\mathbb{B}}$ , no two distinct edges have the same source. In other words, the graph is the graph of a function without fixpoints on  $\{a_1, \dots, a_n\}$ , therefore it contains a cycle of length at least 2. In other words, there is a subset of  $S$  of size at least 2 that is mapped to a set of the form  $\{a_i a_j, a_j a_k, \dots, a_m a_i\}$ .

The two negative results in Theorems 12 and 14 are complemented by a positive one:

► **Theorem 16.** HOMOMORPHISM *is decidable for finite signatures.*

This gives (3) of Theorem 3. Theorem 16 is implicit in the work of Bodirsky, Pinsker and Tsankov [11], where it is proved in a special case when  $\mathbb{A} = \mathbb{B}^n$ , for  $n \geq 1$ , and  $\mathbb{B}$  is a reduct of a finitely bounded Ramsey structure  $\mathcal{A}$  (cf. Section 5). Our self-contained proof of Theorem 16, given in Section 4, instead of using the machinery of *canonical mappings* goes by a direct reduction to the case when the target structure is finite, which is decidable as shown in [21]. Our reduction slightly generalizes a reduction due to Bodirsky and Mottet [7] in the special case of the target structure being a reduct of  $\mathcal{A}$  (both reductions need  $\mathcal{A}$  to be a finitely bounded homogeneous structure).

Theorems 8–16 settle the decidability landscape for the homomorphism problem almost entirely. One remaining open problem is the decidability status of DEFINABLE HOMOMORPHISM for a fixed target structure  $\mathbb{B}$  over a finite signature.

### 3.3 Homomorphism extension problem

Theorem 16 may be a little surprising in light of Theorem 12. Indeed, Remark 2 allows one to view an arbitrary definable  $\Sigma$ -structure as a definable structure  $\mathbb{A}_\Sigma$  over a finite signature. Homomorphisms  $\mathbb{A} \rightarrow \mathbb{B}$  correspond to those homomorphisms  $\mathbb{A}_\Sigma \rightarrow \mathbb{B}_\Sigma$  that are the identity on the subset  $\Sigma$  of the universe of  $\mathbb{A}_\Sigma$ . Thus by Theorem 12 we obtain undecidability, even for finite signatures, of the following slight generalization of HOMOMORPHISM:

**Problem:** HOMOMORPHISM EXTENSION

**Input:** Definable structures  $\mathbb{A}$  and  $\mathbb{B}$  over  $\Sigma$  and a definable partial mapping  $f : A \rightarrow B$ .

**Decide:** Is there a homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$  extending  $f$ ?

The above remark proves (4) of Theorem 3:

► **Theorem 17.** HOMOMORPHISM EXTENSION *is undecidable for finite signatures.*

## 4 Homomorphism problem for finite signatures

Throughout this section, we assume that  $\Sigma$  is a finite signature. For simplicity, assume that  $\mathcal{A}$  is the pure set; in Section 5 we discuss how the results generalize to other underlying structures. We consider the homomorphism problem for structures over  $\Sigma$  which are definable over  $\mathcal{A}$ . For simplicity, we assume that the input structures  $\mathbb{A}$  and  $\mathbb{B}$  are  $\emptyset$ -definable – the proof easily generalizes to arbitrary definable structures over a finite signature.

Here is the main result of this section:

► **Theorem 18.** *Given a  $\emptyset$ -definable structure  $\mathbb{B}$  over a finite signature, one can compute a finite structure  $\mathbb{B}'$  such that:*

- $\text{CSP}(\mathbb{B})$  *is polynomial-time reducible to*  $\text{CSP}(\mathbb{B}')$ ,
- $\text{CSP}_{\text{def}}(\mathbb{B})$  *is polynomial-time reducible to*  $\text{CSP}_{\text{def}}(\mathbb{B}')$ .

Note that Theorem 18 implies Theorem 16, as finite structures  $\mathbb{B}'$  are a special case of *locally finite* ones, and decidability of the homomorphism problem for locally finite target structures has been shown in [21]. Moreover, Theorem 18 implies Theorem 4, since as shown in [21], for every finite template  $\mathbb{B}'$ , the complexity of  $\text{CSP}_{\text{def}}(\mathbb{B}')$  is exponentially larger than the complexity of  $\text{CSP}(\mathbb{B}')$ .

Theorem 18 is a slight extension of results implicit in the work of Bodirsky, Pinsker and Tsankov [11] that provided a decision procedure for testing the existence of a homomorphism from  $\mathbb{A} = \mathbb{B}^n$  to  $\mathbb{B}$ , where  $\mathbb{B}$  is assumed to be a reduct of  $\mathcal{A}$  (with further assumptions about  $\mathcal{A}$ , which apply also in our case, as discussed in Section 5). Instead, we allow both  $\mathbb{A}$  and  $\mathbb{B}$  to be arbitrary definable structures over  $\mathcal{A}$  (in particular, they need not be reducts). Our reduction in Theorem 18 is based on a reduction due to Bodirsky and Mottet [7], generalized to the case of definable structures rather than reducts (recall from Remark 2 that according to our definition, definable structures correspond to structures which interpret in  $\mathcal{A}$  via first-order interpretations).

**Proof of Theorem 18.** The remaining part of this section is devoted to demonstrating Theorem 18. In the sequel we fix a  $\emptyset$ -definable structure  $\mathbb{B}$  over the signature  $\Sigma$  (assumed to be finite). We show how to effectively construct a finite structure  $\mathbb{B}'$  as described in the theorem.

First we observe that without loss of generality we may assume that the universe  $B$  of  $\mathbb{B}$  is a subset of  $\mathcal{A}^k$ , for some  $k$ . To see this, apply Lemma 7 to obtain  $\bar{B} \subseteq \mathcal{A}^k$  and a surjection  $g : \bar{B} \rightarrow B$ , both definable and computable from  $\mathbb{B}$ . Then compute a definable structure  $\bar{\mathbb{B}}$

with universe  $\bar{B}$  over the same signature as  $\mathbb{B}$ , where every relation symbol is interpreted in  $\bar{\mathbb{B}}$  as the inverse image under  $g$  of its interpretation in  $\mathbb{B}$ . Finally, observe that there is a homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$  if, and only if there is a homomorphism from  $\mathbb{A}$  to  $\bar{\mathbb{B}}$ . Thus from now on we assume that  $B \subseteq \mathcal{A}^k$ , for some  $k$ .

We now define some notation. For  $n \in \mathbb{N}$ , denote  $\{1, \dots, n\}$  by  $[n]$ . For a set  $C$ , numbers  $m, n \in \mathbb{N}$  and an injective, monotone function  $i : [m] \rightarrow [n]$ , consider a projection mapping  $\pi_i : C^n \rightarrow C^m$  onto  $m$  coordinates induced by  $i$  in the obvious way, i.e.,  $\pi_i(c_1, \dots, c_n) = (c_{i(1)}, \dots, c_{i(m)})$ . Let  $\mathbb{C}$  be a structure with universe  $C$  and let  $r \geq 2$  be an integer at least as large as the maximal arity of the relations in  $\mathbb{C}$ . We define a structure  $\mathbb{C}^{\leq r}$  with universe  $C^{\leq r} = C \cup C^2 \cup \dots \cup C^r$ , as follows. If  $R$  is a relation symbol of arity  $k$  in the signature of  $\mathbb{C}$ , then the signature of  $\mathbb{C}^{\leq r}$  contains a *unary* symbol  $U_R$ . If  $S \subseteq C^k$  is the interpretation of  $R$  in  $\mathbb{C}$ , then the interpretation of  $U_R$  in  $\mathbb{C}^{\leq r}$  is the set  $S \subseteq C^k \subseteq C^{\leq r}$ , treated as a unary relation. Moreover, for  $m \leq n \leq r$  and each monotone injection  $i : [m] \rightarrow [n]$ , there is a binary *projection relation*  $\Pi_i \subseteq C^n \times C^m$  in  $\mathbb{C}^{\leq r}$  which is the graph of the projection  $\pi_i$ .

We use standard notions of group actions and orbits. The group  $\text{Aut}(\mathcal{A})$  acts on  $\mathcal{A}^k$ , where an automorphism of  $\mathcal{A}$  acts coordinatewisely on elements of  $\mathcal{A}^k$ . Note that this action preserves  $B \subseteq \mathcal{A}^k$ , since  $B$  is  $\emptyset$ -definable. For the same reason, automorphisms of  $\text{Aut}(\mathcal{A})$  preserve the relations of  $\mathbb{B}$ . Reassuming,  $\text{Aut}(\mathcal{A})$  acts on the structure  $\mathbb{B}$  by automorphisms. Similarly,  $\text{Aut}(\mathcal{A})$  acts on the structure  $\mathbb{B}^{\leq r}$ , inducing a quotient relational structure  $\mathbb{B}^{\leq r}/\text{Aut}(\mathcal{A})$  over the same signature. The elements of  $\mathbb{B}^{\leq r}/\text{Aut}(\mathcal{A})$  are orbits of  $B^{\leq r}$  under the action of  $\text{Aut}(\mathcal{A})$ ; in other words, elements of  $\mathbb{B}^{\leq r}/\text{Aut}(\mathcal{A})$  are atomic types of  $k$ -tuples of atoms (an atomic type of a tuple of elements  $(a_1, \dots, a_k) \in \mathcal{A}^k$  specifies all equalities among the elements  $a_1, \dots, a_k$ ). Relation symbols are interpreted in  $\mathbb{B}^{\leq r}/\text{Aut}(\mathcal{A})$  existentially, as expected. A crucial but obvious observation is that the quotient structure  $\mathbb{B}^{\leq r}/\text{Aut}(\mathcal{A})$  is finite, by the following lemma.

► **Lemma 19.** *The group  $\text{Aut}(\mathcal{A})$  acts oligomorphically on  $\mathbb{B}$ , i.e., the action splits  $B^n$  into finitely many orbits, for every  $n \geq 1$ .*

We now define the structure  $\mathbb{B}'$  promised in Theorem 18 as  $\mathbb{B}^{\leq r}/\text{Aut}(\mathcal{A})$ , where  $r \geq 3$  is a fixed number at least as large as the maximal arity of the relations in  $\Sigma$ . As required, the structure  $\mathbb{B}'$  is finite. It remains to prove the two items of Theorem 18. Both reductions are shown in the same way. Let  $\mathbb{A}$  be given, where  $\mathbb{A}$  is either finite or  $\emptyset$ -definable. Define  $\mathbb{A}'$  as  $\mathbb{A}^{\leq r}$ . Note that if  $\mathbb{A}$  is  $\emptyset$ -definable, then so is  $\mathbb{A}'$ . Moreover, (the definition of)  $\mathbb{A}'$  is computable from  $\mathbb{A}$  in polynomial time, for a fixed signature  $\Sigma$ . To complete the reductions, it remains to prove the following:

► **Claim 20.** *There is a homomorphism  $\mathbb{A} \rightarrow \mathbb{B}$  if, and only if, there is a homomorphism  $\mathbb{A}' \rightarrow \mathbb{B}'$ .*

The “only if” direction is immediate; from a given homomorphism  $h : \mathbb{A} \rightarrow \mathbb{B}$ , a homomorphism  $h' : \mathbb{A}' \rightarrow \mathbb{B}'$  is obtained by taking the pointwise extension  $h^{\leq r} : \mathbb{A}^{\leq r} \rightarrow \mathbb{B}^{\leq r}$  of  $h$  (also a homomorphism), and then post-composing  $h^{\leq r}$  with the quotient homomorphism from  $\mathbb{B}^{\leq r}$  to  $\mathbb{B}^{\leq r}/\text{Aut}(\mathcal{A})$ .

We now prove the “if” direction. Fix a homomorphism  $f : \mathbb{A}^{\leq r} \rightarrow \mathbb{B}^{\leq r}/\text{Aut}(\mathcal{A})$ . Recall that  $B \subseteq \mathcal{A}^k$ . Consider the set  $D = (A \times \{1, \dots, k\})/\sim$ , where the equivalence relation  $\sim$  is defined as follows. Take  $(a_1, \dots, a_n) \in A^n$ , for  $n \leq r$ . Then  $f(a_1, \dots, a_n) \in B^n/\text{Aut}(\mathcal{A})$  corresponds to an atomic type of  $(n \cdot k)$ -tuples of atoms. In particular for  $n = 2$ , the atomic type concerns tuples  $(x_1^1, \dots, x_1^k, x_2^1, \dots, x_2^k)$  and for each  $1 \leq i, j \leq k$  and  $1 \leq l, m \leq 2$ , specifies a relation  $x_l^i = x_m^j$  or  $x_l^i \neq x_m^j$ . Put  $(a_1, i) \sim (a_2, j)$  in  $A \times \{1, \dots, k\}$  if  $(a_1, i) = (a_2, j)$



or the atomic type specifies the relation  $x_1^i = x_2^j$ . This defines an equivalence relation on  $A \times \{1, \dots, k\}$ , where  $r \geq 3$  is essential for transitivity; it is also important here that  $f$  is a homomorphism and hence preserves projections. Since the set  $D$  is at most countable, there is an injective function  $e : D \rightarrow \mathcal{A}$ . We define a function  $h : A \rightarrow \mathcal{A}^k$ , by composing the abstraction function  $[\_]\sim : A \times \{1, \dots, k\} \rightarrow D$  with the function  $e$ :

$$h(a) = (e([\_]\sim(a, 1)), \dots, e([\_]\sim(a, k))).$$

Note that  $h(a) \in B$  for every  $a \in A$ . It follows by construction that the function  $h : A \rightarrow B$  is a homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$ .  $\blacktriangleleft$

## 5 Concluding remarks

We investigated the homomorphism problem for definable relational structures. Our contribution is a detailed decidability border in the landscape of different variants of the problem.

Most of our proofs work, or can be easily adapted to the variant of the problem where one asks about the existence of an injective homomorphism, or a strong homomorphism, or an embedding. The only exceptions are Theorems 12 and 14 for the case where the target structure  $\mathbb{B}$  is fixed. Our proofs there work for the case of injective homomorphisms, but not for strong homomorphisms or embeddings, and the decidability of these cases remain open.

**Underlying structure  $\mathcal{A}$ .** We briefly describe the assumptions on the structure  $\mathcal{A}$  for which the results presented in this paper still hold.

The definitions and lemmas in Section 2 hold for an arbitrary structure  $\mathcal{A}$ . However, one needs to specify how inputs are represented, specifically, the parameters involved in the input. To represent all definable sets over  $\mathcal{A}$ , we should assume that there is an effective enumeration of its universe. Furthermore, to effectively perform tests on definable sets one needs to assume that the structure is *decidable*: given any first-order formula  $\phi$  over the signature of  $\mathcal{A}$  with  $n$  free variables, and an  $n$  tuple  $\bar{a}$  of elements of  $\mathcal{A}$ , it must be decidable if  $\phi, \bar{a} \models \mathcal{A}$ . For simplicity we assume that the signature of  $\mathcal{A}$  is finite, to avoid questions concerning the encoding of relation symbols.

Theorems 12, 14 and 17 hold for every infinite structure  $\mathcal{A}$ . For Theorems 12 and 17 this is clear, as every structure definable over the pure set is also definable over arbitrary infinite  $\mathcal{A}$ , and existence of a homomorphism does not depend on  $\mathcal{A}$ . For Theorem 14 this is less clear, since the existence of definable homomorphisms depends on  $\mathcal{A}$ . However, an inspection of the proof shows that the result holds for arbitrary  $\mathcal{A}$ .

The  $\emptyset$ -definable homomorphism problem considered in Theorem 8 is decidable (with the same proof) as long as the following conditions hold:

- $\mathcal{A}$  is  $\omega$ -categorical, i.e., it is the only countable model of its first-order theory. An equivalent condition, due to the Ryll-Nardzewski-Engeler-Svenonius theorem [19], is that  $\mathcal{A}$  is countable and  $\text{Aut}(\mathcal{A})$  acts oligomorphically on  $\mathcal{A}$ .
- The number of orbits of  $\mathcal{A}^n$  under the action of  $\text{Aut}(\mathcal{A})$  is computable from a given  $n \in \mathbb{N}$ .

We call such structures *effectively  $\omega$ -categorical*. Any effectively  $\omega$ -categorical structure is (isomorphic to) a decidable structure, so every definable set can be represented. Theorem 8 can be easily generalized so that arbitrary definable structures  $\mathbb{A}, \mathbb{B}$  are given on input, as well as a finite set  $S \subseteq \mathcal{A}$ , and the algorithm determines whether there exists an  $S$ -definable homomorphism from  $\mathbb{A}$  to  $\mathbb{B}$ .



Regarding Theorem 11, in the case when the source structure  $\mathbb{A}$  is assumed to be finite, it is sufficient that  $\mathcal{A}$  is a decidable structure. In the case when the target structure  $\mathbb{B}$  is finite, and arbitrary homomorphisms are considered, the assumptions under which the proof from [21] works are that  $\text{Aut}(\mathcal{A})$  is extremely amenable or, equivalently, that  $\mathcal{A}$  is a *Ramsey structure* [20]. Examples of Ramsey structures include  $(\mathbb{Q}, \leq)$  and the ordered random graph, by [25].

We do not know how to generalize to other atoms the case where only definable homomorphisms to a finite  $\mathbb{B}$  are considered.

Regarding Theorems 4, 16 and 18, the proofs presented in Section 4 work under the following assumptions:

- The structure  $\mathbb{A}$  is definable over a decidable Ramsey structure  $\mathcal{A}$ . For the second item of Theorem 4, we need some additional mild complexity assumptions about  $\mathcal{A}$ , e.g. that its first-order theory is decidable in NEXPTIME (for most reasonable structures it is in PSPACE). It is shown in [11] that if  $\mathcal{A}$  is a Ramsey structure, then extending  $\mathcal{A}$  by finitely many constants still yields a Ramsey structure. Clearly, this preserves decidability of the structure. From this it follows that the assumption made in Section 4 that the relations of  $\mathbb{A}$  and  $\mathbb{B}$  are  $\emptyset$ -definable is not relevant, since if they are  $S$ -definable over  $\mathcal{A}$  for some finite  $S \subseteq \mathcal{A}$ , then they are  $\emptyset$ -definable over  $\mathcal{A}$  extended by elements of  $S$  as constants.
- The structure  $\mathbb{B}$  is definable over a structure  $\mathcal{B}$  which is homogeneous and *finitely bounded*. We say that a structure  $\mathcal{B}$  over a signature  $\Gamma$  is finitely bounded if there is a finite set  $\mathcal{F}$  of finite  $\Gamma$ -structures such that for every finite  $\Gamma$ -structure  $\mathbb{A}$ ,  $\mathbb{A}$  embeds into  $\mathcal{B}$  iff no structure from  $\mathcal{F}$  embeds into  $\mathbb{A}$ . For example, the pure set is finitely bounded, as witnessed by an empty family  $\mathcal{F}$ . This property is crucial for the proof of Claim 20. It is straightforward to generalize this claim to a finitely bounded homogeneous structure (see [11]). Any finitely bounded homogeneous structure is effectively  $\omega$ -categorical, and thus decidable. Moreover, any expansion of a finitely bounded homogeneous structure by a constant is homogeneous and finitely bounded [7].

We do not know whether the finite boundedness condition can be dropped, while assuming that  $\mathcal{B}$  is effectively  $\omega$ -categorical.

**Open problems.** Perhaps the most significant open question that remains is the decidability of the *isomorphism problem*: decide whether two definable structures  $\mathbb{A}, \mathbb{B}$  (say, over the pure set) are isomorphic, or whether there is a definable isomorphism between them. An equivalent formulation of the former question is the *orbit problem*: given a definable structure  $\mathbb{A}$  and two elements  $x, y \in A$ , decide whether there is an automorphism of  $\mathbb{A}$  which maps  $x$  to  $y$ .

This is related to an open problem from [11]: decide whether a given relation  $R$  is first-order definable in a given structure  $\mathbb{A}$ . Indeed, a unary predicate  $R \subseteq A$  is first-order definable in  $\mathbb{A}$  iff it is preserved by all automorphisms of  $\mathbb{A}$ , iff no  $x \in R$  and  $y \in A - R$  lie in the same orbit of  $\text{Aut}(\mathbb{A})$ .

**Acknowledgments.** We are grateful to Albert Atserias, Manuel Bodirsky and Michael Pinsker for useful discussions.

---

## References

- 1 Robert Berger. The undecidability of the domino problem. *Mem. Amer. Math. Soc.*, 66, 1966.

- 2 Manuel Bodirsky. Complexity classification in infinite-domain constraint satisfaction. *Mémoire d'habilitation à diriger des recherches*, Université Diderot – Paris 7, Available at arXiv:1201.0856, 2012.
- 3 Manuel Bodirsky and Jan Kára. The complexity of equality constraint languages. *Theory Comput. Syst.*, 43(2):136–158, 2008.
- 4 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *J. ACM*, 57(2), 2010.
- 5 Manuel Bodirsky, Barnaby Martin, and Antoine Mottet. Constraint satisfaction problems over the integers with successor. In *Procs. ICALP*, volume 9134 of *LNCS*, pages 256–267, 2015.
- 6 Manuel Bodirsky, Barnaby Martin, Michael Pinsker, and András Pongrácz. Constraint satisfaction problems for reducts of homogeneous graphs. In *Procs. of ICALP'16*, 2016. To appear.
- 7 Manuel Bodirsky and Antoine Mottet. Reducts of finitely bounded homogeneous structures, and lifting tractability from finite-domain constraint satisfaction. In *Procs. of LICS'16*, 2016. To appear.
- 8 Manuel Bodirsky and Jaroslav Nešetřil. Constraint satisfaction with countable homogeneous templates. *J. Log. Comput.*, 16(3):359–373, 2006.
- 9 Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. In *Procs. STOC*, pages 655–664, 2011.
- 10 Manuel Bodirsky and Michael Pinsker. Topological Birkhoff. *Transactions of the AMS*, 367:2527–2549, 2015.
- 11 Manuel Bodirsky, Michael Pinsker, and Todor Tsankov. Decidability of definability. *The Journal of Symbolic Logic*, 78(04):1036–1054, 2013.
- 12 Manuel Bodirsky and Michal Wrona. Equivalence constraint satisfaction problems. In *Procs. CSL*, pages 122–136, 2012.
- 13 M. Bojańczyk and S. Toruńczyk. Imperative programming in sets with atoms. In *Procs. FSTTCS 2012*, volume 18 of *LIPICs*, 2012.
- 14 Mikołaj Bojańczyk, Laurent Braud, Bartek Klin, and Sławomir Lasota. Towards nominal computation. In *Procs. POPL 2012*, pages 401–412, 2012.
- 15 Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Log. Meth. Comp. Sci.*, 10, 2014.
- 16 Mikołaj Bojańczyk, Bartek Klin, Sławomir Lasota, and Szymon Toruńczyk. Turing machines with atoms. In *LICS*, pages 183–192, 2013.
- 17 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992.
- 18 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic snc and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, February 1999. doi:10.1137/S0097539794266766.
- 19 Wilfrid Hodges. *Model Theory*. Number 42 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1993.
- 20 A. S. Kechris, V. G. Pestov, and S. Todorcevic. Fraïssé limits, Ramsey theory, and topological dynamics of automorphism groups. *Geometric & Functional Analysis GAFA*, 15(1):106–189, 2005.
- 21 Bartek Klin, Eryk Kopczyński, Joanna Ochremiak, and Szymon Toruńczyk. Locally finite constraint satisfaction problems. In *Procs. of LICS'15*, pages 475–486, 2015.
- 22 Bartek Klin and Michal Szynwelski. SMT solving for functional programming over infinite structures. In *Procs. MSFP'16*, pages 57–75, 2016.
- 23 P. Kolaitis and M. Vardi. The decision problem for the probabilities of higher-order properties. In *Procs. STOC'87*, pages 425–435, 1987.

- 24 Eryk Kopczyński and Szymon Toruńczyk. LOIS: an application of SMT solvers. In *Procs. SMT Workshop*, volume 1716 of *CEUR Proceedings*, pages 51–60, 2016.
- 25 Jaroslav Nešetřil and Vojtech Rödl. Partitions of finite relational and set systems. *Journal of Combinatorial Theory, Series A*, 22(3):289–312, 1977.
- 26 A.M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, 2013.



# On the Sensitivity Conjecture for Disjunctive Normal Forms\*

Karthik C. S.<sup>†1</sup> and Sébastien Tavenas<sup>‡2</sup>

1 Weizmann Institute of Science, Israel  
karthik.srikanta@weizmann.ac.il

2 Microsoft Research, India  
t-sebat@microsoft.com

---

## Abstract

The sensitivity conjecture of Nisan and Szegedy [CC'94] asks whether for any Boolean function  $f$ , the maximum sensitivity  $s(f)$ , is polynomially related to its block sensitivity  $bs(f)$ , and hence to other major complexity measures. Despite major advances in the analysis of Boolean functions over the last decade, the problem remains widely open.

In this paper, we consider a restriction on the class of Boolean functions through a model of computation (DNF), and refer to the functions adhering to this restriction as admitting the Normalized Block property. We prove that for any function  $f$  admitting the Normalized Block property,  $bs(f) \leq 4s(f)^2$ . We note that (almost) all the functions mentioned in literature that achieve a quadratic separation between sensitivity and block sensitivity admit the Normalized Block property.

Recently, Gopalan et al. [ITCS'16] showed that every Boolean function  $f$  is uniquely specified by its values on a Hamming ball of radius at most  $2s(f)$ . We extend this result and also construct examples of Boolean functions which provide the matching lower bounds.

**1998 ACM Subject Classification** F.1.1 Models of Computation

**Keywords and phrases** Boolean function, Sensitivity, Block sensitivity, DNF

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.15

## 1 Introduction

Sensitivity and block sensitivity are complexity measures that are commonly used for Boolean functions. Both these measures were originally introduced for studying the time complexity of CRAW-PRAM's [7, 8, 15]. Block sensitivity is polynomially related to a number of other complexity measures, such as the decision-tree complexity, the certificate complexity, the polynomial degree, and the quantum query complexity [5]. A longstanding open problem is the relation between sensitivity and block sensitivity. From the definitions of sensitivity and block sensitivity, it immediately follows that  $s(f) \leq bs(f)$ , where  $s(f)$  and  $bs(f)$  denote the sensitivity and the block sensitivity of a Boolean function  $f$ . Nisan and Szegedy [16] conjectured that sensitivity is also polynomially related to block sensitivity:

► **Conjecture 1** (Sensitivity Conjecture [16]). *There exist constants  $\delta, c > 0$  such that for every Boolean function  $f$  we have that  $bs(f) \leq c \cdot (s(f))^\delta$ .*

---

\* A full version of the paper is available at <https://arxiv.org/abs/1607.05189>.

† This work was partially supported by Irit Dinur's ERC-StG grant number 239985. Some parts of this work were done while interning at Microsoft Research, India.

‡ This work was supported by ANR project CompA (project number: ANR-13-BS02-0001-01).



© Karthik C. S. and Sébastien Tavenas;

licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 15; pp. 15:1–15:15



Leibniz International Proceedings in Informatics

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

This conjecture is still widely open and the best known upper bound on block sensitivity is exponential in terms of sensitivity [1]. On the other hand, the best known separation (through an example of a Boolean function) between sensitivity and block sensitivity is quadratic [3]; more background and discussion about the sensitivity conjecture can be found in the survey of Hatami et al. [12].

Over the last decade, in the majority of the works concerning the sensitivity conjecture, the focus has been on addressing the conjecture for restricted classes of Boolean functions, where the restriction is imposed by some notion of symmetry [6, 18, 9]. The reason behind pursuing this direction is that nonconstant Boolean functions with a high degree of symmetry must have high complexity according to various measures. Accordingly, all the results in this direction [6, 18, 9] show that the sensitivity of the corresponding functions is large (in terms of the number of variables), and deduce that the sensitivity is close to block sensitivity. While we feel that proving the sensitivity conjecture for a restricted class of Boolean functions is a step in the right direction, we would like to argue that these specific restrictions are limited in their potential to explicitly promote the understanding of the *relationship* between sensitivity and block sensitivity.

In this paper, we prove the sensitivity conjecture for a restricted class of Boolean functions, where the restriction is imposed on a DNF representation of the function. This is one of the first time[s] since Nisan [15] that the sensitivity conjecture is proved for a restriction based on a model of computation (recently, Lin and Zhang [14] proved the sensitivity conjecture for functions admitting circuits with a small number of negation gates, and in a simultaneous work [4], the authors prove the sensitivity conjecture in the case of regular read- $k$  formulas of constant depth with  $k$  constant). Informally, the restriction we impose on the DNF can be described as follows. We assume that the maximal block sensitivity is reached on the all zeroes input and that the function outputs a zero on this input, and notice that for each clause in the DNF, the set of positive literals in the clause corresponds to a sensitive block. Based on the fact that the block sensitivity counts the number of disjoint sensitive blocks, we consider the natural restriction where the set of positive literals of each of the clauses are also disjoint. We say that any function adhering to this restriction admits the *normalized block property*, and we show that for any Boolean function  $f$  admitting the normalized block property,  $\mathbf{bs}(f) < 4\mathbf{s}(f)^2$ .

As the other side of the same coin, this result provides a barrier to building Boolean functions with super-quadratic separation between sensitivity and block sensitivity. Currently, the best known separation is given by an example of Ambainis and Sun [3] who built a function  $f$  with  $\mathbf{bs}(f) = \frac{2}{3}\mathbf{s}(f)^2 - \frac{1}{3}\mathbf{s}(f)$ . Ambainis and Sun additionally showed that their example gives the best possible separation (up to an additive factor) between sensitivity and block sensitivity for all functions that are an OR of functions whose zero-sensitivity equals 1. We build a framework (of restrictions) over DNFs and identify where the result of Ambainis and Sun lies within this framework, and our result that the sensitivity conjecture is true for Boolean functions admitting normalized block property is shown to be an extension of the result of Ambainis and Sun. Additionally, Kenyon-Kutin [13], showed that if the block sensitivity is attained on some input which has blocks of size at most two then,  $\mathbf{bs} \leq e \cdot \mathbf{s}^2$ . More generally,

► **Theorem 2** (Kenyon and Kutin[13]). *For every Boolean function  $f$  on  $n$  variables, and every  $\ell \in \{2, \dots, \mathbf{s}(f)\}$ , we have:*

$$\mathbf{bs}_\ell(f) \leq \frac{e}{(\ell - 1)!} (\mathbf{s}(f))^\ell,$$

where  $\mathbf{bs}_\ell(f)$  is the block sensitivity of  $f$  when each block is restricted to be of size at most  $\ell$ .

Therefore, to construct examples of Boolean function with super-quadratic separation between sensitivity and block sensitivity we now have two barriers. Moreover, we extend the notion of block property to  $t$ -block property, and prove a lower bound on the sensitivity of Boolean functions admitting the  $t$ -block property in terms of  $t$ , and the width and size of the DNF.

Recently, Gopalan et al. [11] investigated the computational complexity of low sensitivity functions and provided interesting upper bounds on their circuit complexity. This was indicated to be a promising alternative approach to the sensitivity conjecture as opposed to getting improved bounds on specific low level measures like block sensitivity or decision tree depth [13, 1, 3]. In particular, they showed that every Boolean function  $f$  is uniquely specified by its values on a Hamming ball of radius at most  $2s(f)$ , and showed various applications of this result. We extend this result by showing that if two Boolean functions  $f$  and  $g$  coincide on a ball of radius  $s(f) + s(g)$  then,  $f = g$ . Furthermore, for every  $p, q > 1$ , we construct examples of Boolean functions  $f$  and  $g$  such that  $s(f) = p$ ,  $s(g) = q$ , and  $f$  and  $g$  coincide on a ball of radius  $s(f) + s(g) - 1$  but  $f \neq g$ , showing that the above result is tight.

Finally, we propose a computational problem motivated by the sensitivity conjecture, and the existing work and results therein. Assuming the sensitivity conjecture to be true, we note that this problem is in **TFNP**, and wonder if resolving the sensitivity conjecture would yield an efficient algorithm to this computational problem.

This paper is organized as follows. In Section 2, we provide the basic definitions of complexity measures, structures, and objects that will be used in the rest of the paper. In Section 3, we define a few restrictions (such as the block property) on DNFs representing Boolean functions and prove the sensitivity conjecture for the class of functions admitting (some of) these structural restrictions. In Section 4, we investigate a structural result of low sensitivity functions. In Section 5, we propose a new computational problem motivated by the sensitivity conjecture. Finally, in Section 6, we conclude with a promising open question on proving the sensitivity conjecture for functions admitting the  $t$ -block property.

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

## 2 Preliminaries

We use the notation  $[n] = \{1, \dots, n\}$ . Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , be a Boolean function. Let  $x \in \{0, 1\}^n$ . For  $i \in [n]$ , we denote by  $x^i$  the input in  $\{0, 1\}^n$  which is obtained by flipping the  $i^{\text{th}}$  bit of  $x$ . Also for any  $B \subseteq [n]$ , we denote by  $x^B$  the input in  $\{0, 1\}^n$  which is obtained by flipping the bits of  $x$  in all coordinates in  $B$ . We will now define two complexity measures on Boolean functions which are of great interest.

► **Definition 3.** The sensitivity of a Boolean function  $f$  at input  $x \in \{0, 1\}^n$ , written  $s(f, x)$ , is the number of coordinates  $i \in [n]$  such that  $f(x) \neq f(x^i)$ . The sensitivity of  $f$ , written  $s(f)$ , is defined as  $s(f) = \max_{x \in \{0, 1\}^n} s(f, x)$ . We define  $s_1(f) = \max_{f(x)=1} s(f, x)$  and  $s_0(f) = \max_{f(x)=0} s(f, x)$ .

► **Definition 4.** The block sensitivity of a Boolean function  $f$  at input  $x \in \{0, 1\}^n$ , for  $k$  disjoint subsets  $B_1, \dots, B_k$  of  $[n]$  (called blocks), written  $bs(f, x, B_1, \dots, B_k)$ , is the number of blocks  $i \in [k]$  such that  $f(x) \neq f(x^{B_i})$ . The block sensitivity of a Boolean function  $f$  at input  $x \in \{0, 1\}^n$ , written as  $bs(f, x)$ , is the maximum of  $bs(f, x, B_1, \dots, B_k)$  over all  $k$  disjoint subsets  $B_1, \dots, B_k$  of  $[n]$  for all  $k \in [n]$ . The block sensitivity of  $f$ , written  $bs(f)$ , is defined as  $bs(f) = \max_{x \in \{0, 1\}^n} bs(f, x)$ . We define  $bs_1(f) = \max_{f(x)=1} bs(f, x)$  and  $bs_0(f) = \max_{f(x)=0} bs(f, x)$ .

We will now introduce a model of representation of Boolean functions.

► **Definition 5.** A DNF (disjunctive normal form) formula  $\Phi$  over Boolean variables  $x_1, \dots, x_n$  is defined to be a logical OR of terms, each of which is a logical AND of literals. A literal is either a variable  $x_i$  or its logical negation  $\bar{x}_i$ . We insist that we can assume that no term contains both a variable and its negation (otherwise we can remove this term). We often identify a DNF formula  $\Phi_f$  with the Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  it computes.

We note here that for every Boolean function  $f$ , there exists at least one (it is not unique) DNF formula  $\Phi_f$  that computes it.

### 3 Block Property

In the following, we will often use the notation  $\vee$  (respectively  $\wedge$ ) for denoting the Boolean operation OR (respectively AND). Let  $f$  be a Boolean function and  $\Phi_f$  be one of its DNF formulas. Let  $X = \{x_1, \dots, x_n\}$  be the set of variables. Let  $d_\vee$  be the fan-in of the  $\vee$ -gate which is usually called the size of the DNF. We label the  $d_\vee$   $\wedge$ -gates as:  $\wedge_1, \dots, \wedge_{d_\vee}$ . Let  $d_{\wedge_i}$  be the fan-in of  $\wedge_i$ . Let  $d_\wedge = \max_i d_{\wedge_i}$  be the width of the DNF. For every  $i \in [d_\vee]$ , let  $A_i$  be the set of variables amongst the literals connected to  $\wedge_i$  appearing **without** a negation and let  $\bar{A}_i$  be the set of variables amongst the literals connected to  $\wedge_i$  appearing **with** a negation. An assignment of the variables is a function  $\sigma : X \rightarrow \{0, 1\}$ . For every  $\wedge_i$ , we define  $S_i$  as follows:

$$S_i = \{\sigma \mid \wedge_i(\sigma) = 1\},$$

where  $\wedge_i(\sigma)$  is the evaluation of  $\wedge_i$  when the assignment to the variables is given by  $\sigma$ .

By negating some variables and/or negating the output of the function, we can always assume that the maximum block sensitivity is the maximum 0-block sensitivity (i.e.,  $\mathbf{bs}_0$ ) and is reached on the all zeroes input. Moreover, given a DNF representation of our function, we can assume that this representation is minimal (i.e., any subformula of the given formula computes a distinct function).

► **Definition 6.** A Boolean function  $f$  represented by a DNF formula  $\Phi_f$  is said to be represented in **compact form** if the following holds:

- (a)  $f(0^n) = 0$ ,
- (b) The maximum 0-block sensitivity is attained on the all zeroes input, i.e.,  $\mathbf{bs}_0(f) = \mathbf{bs}(f, 0)$ ,
- (c) and  $\forall i \in [d_\vee]$ , we have that  $S_i \setminus \bigcup_{j \neq i} S_j \neq \emptyset$ .

Moreover the representation is called **normalized** if the maximal block sensitivity is also attained on the all zeroes input, i.e.,  $\mathbf{bs}(f) = \mathbf{bs}(f, 0)$ .

The condition (c) means that for each  $i$  there exist a  $\sigma$  such that  $\sigma$  makes only  $\wedge_i$  true.

► **Lemma 7.** For every  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , there exists  $f' : \{0, 1\}^n \rightarrow \{0, 1\}$  such that  $\mathbf{s}(f') = \mathbf{s}(f)$ ,  $\mathbf{bs}(f') = \mathbf{bs}(f) = \mathbf{bs}(f', 0^n)$ , and  $f'$  admits a normalized compact form representation.

**Proof.** We claim that for any Boolean function  $f$ , there exists another Boolean function  $f'$  such that  $\mathbf{s}(f) = \mathbf{s}(f')$ ,  $\mathbf{bs}(f) = \mathbf{bs}(f')$ ,  $f'(0^n) = 0$  and such that  $f'$  attains its maximal block sensitivity at the all zeroes input. This is because, if  $f$  attains its maximum block sensitivity at  $a \in \{0, 1\}^n$  then, we define  $f'(x) = f(a) \oplus f(x \oplus a)^1$ , and the claim follows.

---

<sup>1</sup> The operator  $\oplus$  denotes the usual XOR function.



Let us fix a DNF formula for  $f'$ . If there is  $i \in [d_\vee]$  such that  $S_i \subseteq \bigcup_{j \neq i} S_j$ , then we do not change the function by removing  $\wedge_i$ . Thus any such AND gates can be assumed to have been removed.  $\blacktriangleleft$

In fact, we can remark that only the condition  $f(0^n) = 0$  from Definition 6 may need a larger DNF (since, it could need to compute the negation of the original function), other constraints can be achieved without increasing the size of the formula.

We will now describe a structural result about Boolean functions that admit compact form representation. For every  $i \in [d_\vee]$ , we define  $\Gamma_i$  as follows:

$$\Gamma_i = \left\{ j \mid |A_i \cap \bar{A}_j| + |A_j \cap \bar{A}_i| = 1 \right\}.$$

Informally,  $\Gamma_i$  is the set of AND gates which contradict on  $\wedge_i$  on exactly one variable. Let  $\Gamma = \max_i |\Gamma_i|$ . We bound  $\mathbf{s}_1$  using  $\Gamma$  as follows:

► **Lemma 8.** *Any Boolean function  $f$  represented in the compact form admits the following bound on  $\mathbf{s}_1$ :  $d_\wedge - \Gamma \leq \mathbf{s}_1 \leq d_\wedge$ .*

**Proof.** First, we prove that  $\mathbf{s}_1 \leq d_\wedge$ . Let  $a \in \{0, 1\}^n$  be the input for which the maximum  $\mathbf{s}_1$  is attained. By definition of  $\mathbf{s}_1$ , we have that  $f(a) = 1$ . Let  $\wedge_i$  be an AND gate such that  $\wedge_i(a) = 1$ . Suppose  $\mathbf{s}_1 > d_\wedge$  then there exists  $x_j \in X \setminus (A_i \cup \bar{A}_i)$  such that  $f(a^j) = 0$ . But, as  $\wedge_i$  does not depend on  $x_j$ ,  $\wedge_i(a^j) = 1$  and so  $f(a^j)$  still equals 1, which is a contradiction.

We will now prove that  $\mathbf{s}_1 \geq d_\wedge - \Gamma$ . Let  $i_0 = \operatorname{argmax}_i d_{\wedge_i}$ . Let  $b \in S_{i_0} \setminus \bigcup_{j \neq i_0} S_j$  (from Definition 6c such a selection is possible). We have that  $\wedge_{i_0}(b) = 1$  and for all  $j \in [d_\vee] \setminus \{i_0\}$ ,  $\wedge_j(b) = 0$ . It is sufficient to lower bound the cardinality of  $C \subseteq [n]$  such that for all  $i \in C$ , we have that  $f(b^i) = 0$ . Fix some  $x_k \in (A_{i_0} \cup \bar{A}_{i_0})$ . We observe that  $f(b^k) = 1$  implies that there is an AND gate  $\wedge_j$  such that  $(A_{i_0} \cap \bar{A}_j) \cup (\bar{A}_{i_0} \cap A_j) = \{x_k\}$ . There are exactly  $|\Gamma_{i_0}|$  such  $k$ 's. The lower bound follows.  $\blacktriangleleft$

Nisan [15] showed that for all monotone functions the block sensitivity and sensitivity are equal. This was the first time that the sensitivity conjecture was proven for a class of functions captured by a restriction on the model of computation for Boolean functions. In our setting, Nisan's result would be written as follows:

► **Theorem 9** (Nisan [15]). *Let  $f$  be a Boolean function and  $\Phi_f$  be a compact form representation of  $f$ . In  $\Phi_f$  if for every  $i \in d_\vee$ , we had that  $\bar{A}_i = \emptyset$  then,  $\mathbf{bs}(f) = \mathbf{s}(f)$ .*

In this paper, we look at Boolean functions through weaker restrictions on their DNF representation. In this regard, we will now see three kinds of structural impositions on Boolean functions in compact form representation. Later, we will prove the sensitivity conjecture for the class of functions admitting (some of) these structural impositions.

► **Property 10** (Block property). *A Boolean function is said to admit the block property if under a compact form representation  $\forall i, j \in [d_\vee]$  such that  $i \neq j$ , we have that  $A_i \cap A_j = \emptyset$ . Moreover, if there exists such a compact form representation which is also normalized, we will say that the function admits the normalized block property.*

► **Property 11** (Mixing property). *A Boolean function is said to admit the  $\ell$ -mixing property if under a compact form representation  $\forall i, j \in [d_\vee]$  with  $i \neq j$ , such that if  $(A_i \cup \bar{A}_i) \cap (A_j \cup \bar{A}_j) \neq \emptyset$  we have,  $|(A_i \cap \bar{A}_j) \cup (A_j \cap \bar{A}_i)| \geq \ell$ .*

► **Property 12** (Transitive property). *A Boolean function is said to admit the transitive property if under a compact form representation  $\forall i, j, k \in [d_\vee]$ , we have that if  $(A_i \cup \bar{A}_i) \cap (A_j \cup \bar{A}_j) \neq \emptyset$  and if  $(A_j \cup \bar{A}_j) \cap (A_k \cup \bar{A}_k) \neq \emptyset$  then,  $(A_i \cup \bar{A}_i) \cap (A_k \cup \bar{A}_k) \neq \emptyset$ .*

## 15:6 On the Sensitivity Conjecture for Disjunctive Normal Forms

First, we see that if a Boolean function admits the Mixing property then we can improve the bound obtained in Lemma 8.

► **Lemma 13.** *Let  $\ell > 1$ . Any Boolean function admitting the  $\ell$ -mixing property has  $\Gamma = 0$ .*

**Proof.** Fix  $i \in [d_V]$ . Since the function admits  $\ell$ -mixing property, we know that for every  $j \in [d_V]$ , either  $(A_i \cup \bar{A}_i) \cap (A_j \cup \bar{A}_j) = \emptyset$  in which case we have that  $j \notin \Gamma_i$ , or  $|(A_i \cap \bar{A}_j) \cup (A_j \cap \bar{A}_i)| \geq \ell$  in which case we again conclude that  $j \notin \Gamma_i$  because of the following:

$$1 < \ell \leq |(A_i \cap \bar{A}_j) \cup (A_j \cap \bar{A}_i)| = |A_i \cap \bar{A}_j| + |A_j \cap \bar{A}_i|,$$

where the last equality holds because in the definition of DNF formula we insisted that no term contains both a variable and its negation. Therefore, we have that  $\Gamma_i = \emptyset$ . ◀

Consequently, we have that  $\mathbf{s}_1 = d_\wedge$ , for all Boolean functions admitting the  $\ell$ -mixing property with  $\ell > 1$ .

Ambainis and Sun had previously shown in Theorem 2 of [3] that their construction gave the (almost) best possible separation between block sensitivity and sensitivity for a family of Boolean functions. Let us consider the Boolean functions  $f$  which can be written as a variables-disjoint union:

$$f = \bigvee_{i=1}^n g(x_{i,1}, \dots, x_{i,m}). \quad (1)$$

Then (see for example Lemma 1 in [3] or Proposition 31 in [10])  $\mathbf{s}_1(f) = \mathbf{s}_1(g)$ ,  $\mathbf{s}_0(f) = n\mathbf{s}_0(g)$ , and  $\mathbf{bs}_0(f) = n\mathbf{bs}_0(g)$ . So if we can find a lower bound for the sensitivity of  $g$  with respect to  $\mathbf{bs}_0(g)$ , we get the best gap for  $f$  by choosing  $n = \mathbf{s}_1(g)/\mathbf{s}_0(g)$ .

► **Theorem 14 (Ambainis and Sun [3]).** *If  $g$  is a Boolean function such that  $\mathbf{s}_0(g) = 1$  and  $\mathbf{bs}(g) = \mathbf{bs}_0(g)$ , then  $2\mathbf{s}_1(g) \geq 3(\mathbf{bs}(g) - 1)$ .*

In fact, we can notice that these functions belong to our framework (this claim is implicit in their proof of Theorem 14, but we give a proof in the full version):

► **Claim 15.** *Let  $g$  be as in Theorem 14. Let  $f$  be the OR of several copies of  $g$ , where each copy takes its input from a different set of variables, as in Eq. (1). Then, there exists  $f'$  with same block sensitivity and at most same 1-sensitivity which admits the normalized block property, the transitive property, and the 3-mixing property.*

Ambainis and Sun [3] present an explicit Boolean function  $f$  such that  $\mathbf{bs} = \frac{2}{3}\mathbf{s}^2 - \frac{1}{3}\mathbf{s}$ . The function is a variables-disjoint union

$$f = \bigvee_{i=1}^{3n+2} g(x_{i,1}, \dots, x_{i,4n+2}).$$

The function  $g$  outputs one if the  $4n + 2$  corresponding variables satisfy the pattern  $P_{\text{AmbainisSun}}$  or if it is the case after an even-length cyclic rotation of the variables. The pattern starts with  $2n$  0s which are followed by a block of two ones and it finishes by  $n$  copies of the block 0 (the underscore means the variable can be 0 or 1):

0	0	...	...	0	1	1	0	_	0	_	...	...	0	_
---	---	-----	-----	---	---	---	---	---	---	---	-----	-----	---	---

As we only admit the even-length rotations, we can easily see that the normalized block property is ensured. The patterns in  $g$  pairwise intersect, so we also get the transitive property. Finally, if we consider two rotations  $R_1$  and  $R_2$  of the pattern, we can assume that the  $\underline{11}$ -block in  $R_1$  intersects a  $0\underline{\underline{\quad}}$ -block in  $R_2$  (otherwise, we switch  $R_1$  and  $R_2$  and get it). Then the  $\underline{11}$ -block in  $R_2$  will intersect a  $00$ -block in  $R_1$ . The two rotations of the pattern disagree on at least three variables (and in fact exactly three). Hence the 3-mixing property is also verified.

We show in the full version that other functions in literature achieving a quadratic gap (e.g. Rubinfeld [17], Virza [19], Chakraborty [6]) fall in our framework.

We ended up proving a result which supersedes the one mentioned in Theorem 14 both in the lower bound and for a more general family. The above lower bound is **exactly** matched by the Boolean function constructed by Ambainis and Sun [3]. This implies that there cannot exist a Boolean function admitting the normalized block property, the transitive property and the 2-mixing property which has a better separation between block sensitivity and sensitivity than the function constructed by Ambainis and Sun [3].

► **Theorem 16.** *Any Boolean function admitting the normalized block property, the transitive property, the 2-mixing property and which depends on at least two variables has  $3\mathbf{bs} \leq 2\mathbf{s}^2 - \mathbf{s}$ .*

The proof of the above theorem is in the full version. In a previous version of this paper, we did not assume that the number of dependent variables is at least two. However, as Krišjānis Prusis and Andris Ambainis pointed out to us, there was a small error in the proof and indeed the univariate function  $f(x) = x$  does not satisfy this inequality ( $\mathbf{s} = \mathbf{bs} = 1$ ). Moreover, they noticed that, as the 2-mixing property implies  $\mathbf{s}_1 = d_\wedge = \mathbf{C}_1$  (cf. Lemma 13 and the following remark), their result [2] directly implies that any Boolean function admitting the 2-mixing property satisfies  $3\mathbf{bs} \leq 2\mathbf{s}^2 + \mathbf{s}$ .

Our main result is to get rid of the dependence on the transitive property and the mixing property. Imposing only the normalized block property on DNFs is a weak restriction as there is no constraint on  $\overline{A}_i$ . Further, given the DNF in compact form representation admitting the normalized block property is a natural way to represent the function through its (maximal) block sensitivity complexity. We show the following theorem concerning Boolean functions admitting block property:

► **Theorem 17.** *Any Boolean function admitting the block property has  $\mathbf{bs}_0 \leq 4\mathbf{s}^2$ . In particular, if the representation is normalized,  $\mathbf{bs} \leq 4\mathbf{s}^2$ .*

The importance of the result is that the block property seems to be a quite natural restriction for studying the relations between the sensitivity and the block sensitivity. In fact, by assuming that the block sensitivity is maximized, by the blocks  $B_i$ , on the all zeros inputs with  $f(0^n) = 0$  (which is always possible), the block property intuitively asserts the output is one if from the all zeros input, we can get an input in  $f^{-1}(1)$  only by flipping at least one of the blocks  $B_i$ . If it is not the case, it would mean there are other non-disjoint blocks which are present just for diminishing the sensitivity.

Before presenting the proof, we prove three lemmas, after which the above result follows immediately.

► **Lemma 18.** *Any Boolean function  $f$  admitting the block property has  $\mathbf{bs}_0 = d_\vee$ .*

**Proof.** From Definition 6a, we have that  $f(0^n) = 0$  and thus we have that every  $A_i$  is non-empty. Now, it is easy to see that  $\mathbf{bs}_0 \geq \mathbf{bs}(f, 0^n) \geq d_\vee$  – choose each  $A_i$  as a block. Any two blocks are disjoint because of the block property and by flipping any of the blocks, one of the AND gates will evaluate to 1.

From Definition 6b, we know that the maximum 0-block sensitivity is attained on  $0^n$ . Let the sensitive blocks for which it attains maximum 0-block sensitivity be  $B_1, \dots, B_k$ . Thus when some  $B_i$  is flipped to all 1s, at least one of the AND gates evaluates to 1. Since the blocks are disjoint, we can associate a distinct AND gate to each sensitive block. Therefore the number of sensitive blocks is at most the number of AND gates, i.e.,  $\mathbf{bs}_0 = k \leq d_\vee$ . ◀

► **Lemma 19.** *Any Boolean function admitting the block property has  $\mathbf{s} \geq \left\lceil \frac{d_\vee}{2d_\wedge - 1} \right\rceil$ .*

The previous lemma is easily seen as optimal by a multiplicative factor two by considering the OR function.

**Proof.** Let  $E$  be a subset of AND gates such that for any two  $\wedge_i, \wedge_j \in E$ , we have  $A_i \cap \bar{A}_j = \emptyset$  and  $A_j \cap \bar{A}_i = \emptyset$ . Let  $P = \bigcup_{\wedge_i \in E} P_i$ , where  $P_i$  is an arbitrarily chosen subset of  $A_i$  of size  $|A_i| - 1$  (note that  $|A_i| \geq 1$  as otherwise we would have  $f(0^n) = 1$ , contradicting Definition 6a). Consider  $a \in \{0, 1\}^n$ , where  $a_i = 1$  if and only if  $x_i \in P$ . We observe that for all  $\wedge_i \in E$ ,  $\wedge_i(a) = 0$ . Also, for all  $\wedge_i \notin E$ , we have that  $A_i \cap P = \emptyset$  from the block property, and therefore  $\wedge_i(a) = 0$ . In short,  $f(a) = 0$ . Now for any  $\wedge_i \in E$ , let  $x_{q(i)} \in A_i \setminus P_i$ . Since  $\wedge_i(a^{q(i)}) = 1$ , we have that  $\mathbf{s}_0(f, a) \geq |E|$ .

Now, we will prove that there is a set  $E$  such that  $|E| \geq \left\lceil \frac{d_\vee}{2d_\wedge - 1} \right\rceil$ . Let  $G$  be a directed graph on  $d_\vee$  vertices where the  $i^{\text{th}}$  vertex corresponds to  $\wedge_i$ . We have a directed edge from vertex  $i$  to vertex  $j$  if  $\bar{A}_i \cap A_j \neq \emptyset$ . Let  $U(G)$  be  $G$  with orientation on the edges removed. Consider the following procedure for constructing  $E$ :

- (1) Include to  $E$ , the AND gate corresponding to the vertex with the smallest degree in  $U(G)$ .
- (2) Remove the vertex picked in (1) and all its in-neighbors and out-neighbors from  $G$ .
- (3) Repeat (1) if  $G$  is not empty.

From block property, we have that the out-degree of vertex  $i$  in  $G$  is at most  $|\bar{A}_i|$ . Thus the total number of edges in  $G$  is at most  $\sum_{i \in [d_\vee]} |\bar{A}_i| \leq d_\vee(d_\wedge - 1)$ . This implies that the sum of the degree of all vertices in  $U(G)$  is at most  $2d_\vee(d_\wedge - 1)$ . Therefore, there exists a vertex in  $U(G)$  of degree at most  $2d_\wedge - 2$ . By including the corresponding AND gate into  $E$ , the number of vertices in  $G$  reduces by at most  $2d_\wedge - 1$ . In order for  $G$  to be empty, there should be at least  $\left\lceil \frac{d_\vee}{2d_\wedge - 1} \right\rceil$  iterations of the above procedure, and since cardinality of  $E$  grows by 1 after each iteration, we have that  $|E| \geq \left\lceil \frac{d_\vee}{2d_\wedge - 1} \right\rceil$ .

Therefore, we have  $\mathbf{s} \geq \mathbf{s}_0(f, a) \geq |E| \geq \left\lceil \frac{d_\vee}{2d_\wedge - 1} \right\rceil$ . ◀

► **Lemma 20.** *Any Boolean function admitting the block property has  $\mathbf{s} \geq \left\lceil \frac{d_\wedge}{2} \right\rceil$ .*

**Proof.** If  $\mathbf{s}_1 \geq \left\lceil \frac{1+d_\wedge}{2} \right\rceil$ , we are done. Therefore, we can assume  $\mathbf{s}_1 < \left\lceil \frac{1+d_\wedge}{2} \right\rceil$ . Let  $i^* = \operatorname{argmax}_i d_{\wedge_i}$ . Consider  $a \in \{0, 1\}^n$  with  $a_j = 1$  if and only if  $x_j \in A_{i^*}$ . We note that  $\wedge_{i^*}(a) = 1$  and  $|a|$  (Hamming weight of  $a$ ) is nonzero since  $A_{i^*}$  is nonempty from Definition 6a.

Let  $x_j \in A_{i^*}$ . We claim that  $f(a^j) = 0$ . The proof is by contradiction. Suppose,  $f(a^j) = 1$ . It is clear that  $\wedge_{i^*}(a^j) = 0$  as  $x_j \in A_{i^*}$ . Thus, there must exist some  $k \neq i^*$ , such that  $\wedge_k(a^j) = 1$ . From block property, we know that  $A_{i^*} \cap A_k = \emptyset$ , but all variables assigned to 1 in  $a^j$  are in  $A_{i^*}$ . This implies  $A_k = \emptyset$ . Therefore  $\wedge_k(0^n) = 1$ , contradicting Definition 6a.

Now we would like to claim that for any  $x_j \in A_{i^*}$ , we have  $\mathbf{s}_0(f, a^j) \geq 1 + \left\lfloor \frac{d_\wedge}{2} \right\rfloor$ . We first note that  $\mathbf{s}_1(f, a) < \left\lceil \frac{1+d_\wedge}{2} \right\rceil$  and since for any  $x_j \in A_{i^*}$ , we have  $f(a^j) = 0$ , we have

that  $|A_{i^*}| < \lceil \frac{1+d_\Delta}{2} \rceil$ . Let  $D = \{x_p \in \bar{A}_{i^*} \mid f(a^p) = 1\}$ . Since,  $\mathbf{s}_1(f, a) < \lceil \frac{1+d_\Delta}{2} \rceil$ , this implies  $|\bar{A}_{i^*}| - |D| + |A_{i^*}| < \lceil \frac{1+d_\Delta}{2} \rceil$  or equivalently,  $|D| > \lceil \frac{d_\Delta}{2} \rceil - 1$ . Fix  $x_p \in D$  and  $x_j \in A_{i^*}$ . Since  $f(a^p) = 1$ , we know there exists some  $k \neq i^*$ , such that  $\wedge_k(a^p) = 1$ . By block property, we know that  $x_j \notin A_k$ , and this implies  $f(a^{\{j,p\}}) = 1$ . Thus, we have that for any fixed  $x_j \in A_{i^*}$ ,  $\mathbf{s}_0(f, a^j) \geq |D| + 1$  as for every  $x_p \in D$ , we have  $f(a^{\{j,p\}}) = 1$  and also  $f(a) = 1$ . Therefore, for every  $x_j \in A_{i^*}$  we have  $\mathbf{s}_0(f, a^j) \geq |D| + 1 > 1 + \lceil \frac{d_\Delta}{2} \rceil - 1 = \lceil \frac{d_\Delta}{2} \rceil$ .

Therefore, we have that either  $\mathbf{s}_1$  or  $\mathbf{s}_0$  is at least  $\lceil \frac{d_\Delta}{2} \rceil$ . ◀

**Proof of Theorem 17.** From Lemma 19 and Lemma 20, we have that for any Boolean function admitting the block property  $\mathbf{s}^2 > \frac{d_\vee}{4}$ . Combining this with Lemma 18, we have that  $4\mathbf{s}^2 > \mathbf{b}\mathbf{s}_0$ . ◀

We can notice that Lemma 20 is optimal, i.e., we give an example of a Boolean function admitting the block property with  $\mathbf{s}_0 = \mathbf{s}_1 = \lceil d_\wedge/2 \rceil$ . The set of variables is  $X = \{x_1, \dots, x_{2n+1}\}$ . We describe the example by its  $\wedge$ -gates  $\wedge_1, \dots, \wedge_{n+1}$ : for all  $i \in [n]$ ,  $A_i = \{x_{2i}\}$ ,  $\bar{A}_i = \emptyset$ ,  $A_{n+1} = \{x_{2i-1} \mid i \in [n+1]\}$  and  $\bar{A}_i = \{x_{2i} \mid i \in [n]\}$ .

Finally, we conclude with an absolute lower bound on the sensitivity of functions admitting block property.

► **Corollary 21.** *Let  $f$  be a Boolean function which depends on  $n$  variables. If  $f$  admits the block property, then  $2\mathbf{s}(f) \geq n^{1/3}$ .*

**Proof.** The number of variables which appear in the DNF is at most  $d_\vee d_\wedge$ , and so  $d_\vee d_\wedge \geq n$ . By Lemma 19 and Lemma 20,

$$\mathbf{s}^3 \geq \left( \frac{d_\vee}{2d_\wedge} \right) \left( \frac{d_\wedge}{2} \right)^2 \geq \frac{d_\vee d_\wedge}{8} \geq \frac{n}{8}. \quad \blacktriangleleft$$

### 3.1 $t$ -Block Property

In this subsection, we extend the notion of block property to  $t$ -block property as follows.

► **Property 22** ( $t$ -Block property). *A Boolean function is said to admit the  $t$ -block property if under a compact form representation  $\forall x \in X$ , we have  $|\{A_i \mid x \in A_i\}| \leq t$ .*

We have that 1-block property is exactly the same as block property discussed in the previous subsection. Let us notice that the notion of  $t$ -block property is far more general than the one of read- $t$  DNF presented in [4] since, here only the number of times where the variables appear positively is bounded.

First, we show an upper bound on Boolean functions admitting the  $t$ -block property in terms of the size of the DNF. The proof is very similar to the one for Lemma 18 and can be found in the full paper.

► **Lemma 23.** *Any Boolean function  $f$  admitting the  $t$ -block property has  $\mathbf{b}\mathbf{s}_0 \leq d_\vee$ .*

Next, we prove a lower bound on Boolean functions admitting the  $t$ -block property in terms of  $t$ , the width of the DNF, and the size of the DNF.

► **Lemma 24.** *If  $f$  admits the  $t$ -block property, then  $\mathbf{s} \geq \left\lceil \frac{d_\vee}{3td_\wedge - 2t - d_\wedge + 1} \right\rceil$ .*

**Proof.** Let  $E$  be a subset of AND gates such that for any two  $\wedge_i, \wedge_j \in E$ , we have  $(A_i \cup \bar{A}_i) \cap A_j = \emptyset$ . Let  $A = \bigcup_{\wedge_i \in E} A_i$  (note that any  $|A_i| \geq 1$  as otherwise we would have  $f(0^n) = 1$ ,

## 15:10 On the Sensitivity Conjecture for Disjunctive Normal Forms

contradicting Definition 6a). Consider  $\mathcal{A}$  the set of 0-vectors with support in  $A$ . More formally,  $\mathcal{A} = \{a \in \{0, 1\}^n \mid f(a) = 0 \text{ and } \forall i, a_i = 1 \implies x_i \in A\}$ .

First notice that  $0^n \in \mathcal{A}$ , so this set is not empty. Let  $\bar{a}$  be an element of  $\mathcal{A}$  with maximal Hamming weight. For any  $\wedge_i \in E$ , the gate  $\wedge_i$  does not depend on the variables in  $(A \setminus A_i)$  by definition of  $E$  and  $A$ . So,  $f(\bar{a}) = 0$  implies that there exists a variable  $x_{l_i} \in A_i$  such that  $\bar{a}_{l_i} = 0$ . Then, by maximality of Hamming weight of  $\bar{a}$ ,  $f(\bar{a}^{l_i}) = 1$  and so  $\bar{a}$  is 0-sensitive on  $l_i$ . Finally, for all  $i, j \in E$  the indices  $l_i$  and  $l_j$  are distinct since  $A_i \cap A_j = \emptyset$ . Consequently, we have that  $\mathfrak{s}_0(f, \bar{a}) \geq |E|$ .

Now, we will prove that there is a set  $E$  such that  $|E| \geq \left\lceil \frac{d_\vee}{3td_\wedge - 2t - d_\wedge + 1} \right\rceil$ . Let  $G$  be a directed graph on  $d_\vee$  vertices where the  $i^{\text{th}}$  vertex corresponds to  $\wedge_i$ . We have a directed edge from vertex  $i$  to vertex  $j$  if  $\bar{A}_i \cap A_j \neq \emptyset$ . Let  $U(G)$  be  $G$  with orientation on the edges removed. Consider the following procedure for constructing  $E$ :

- (1) Add to  $E$ , the AND gate corresponding to the vertex with the smallest degree in  $U(G)$ .
- (2) Remove the vertex picked in (1) and all its in-neighbors and out-neighbors from  $G$ .
- (3) Remove any vertex from  $G$  associated with a gate  $\wedge_j$  with  $A_i \cap A_j \neq \emptyset$ .
- (4) Repeat from (1) if  $G$  is not empty.

From  $t$ -block property, we have that the out-degree of vertex  $i$  in  $G$  is at most  $t|\bar{A}_i|$ . Thus the total number of edges in  $G$  is at most  $\sum_{i \in [d_\vee]} t|\bar{A}_i| \leq td_\vee(d_\wedge - 1)$ . This implies that

the sum of the degree of all vertices in  $U(G)$  is at most  $2td_\vee(d_\wedge - 1)$ . Therefore, there exists a vertex in  $U(G)$  of degree at most  $2td_\wedge - 2t$ . By including the corresponding AND gate into  $E$ , the number of vertices in  $G$  reduces by at most  $2td_\wedge - 2t + 1$  at step (2). Moreover, at step (3), by the  $t$ -block property, there are at most  $(t-1)|A_i| \leq td_\wedge - d_\wedge$  gates  $\wedge_j$  such that  $A_i \cap A_j \neq \emptyset$  and  $j \neq i$ . Consequently at most  $3td_\wedge - 2t - d_\wedge + 1$  gates are removed at each step. In order for  $G$  to be empty, there should be at least  $\left\lceil \frac{d_\vee}{3td_\wedge - 2t - d_\wedge + 1} \right\rceil$  iterations of the above procedure, and since cardinality of  $E$  grows by 1 after each iteration, we have that  $|E| \geq \left\lceil \frac{d_\vee}{3td_\wedge - 2t - d_\wedge + 1} \right\rceil$ .

Therefore, we have  $\mathfrak{s} \geq \mathfrak{s}_0(f, a) \geq |E| \geq \left\lceil \frac{d_\vee}{3td_\wedge - 2t - d_\wedge + 1} \right\rceil$ . ◀

As a corollary, we obtain the following.

► **Corollary 25.** *Let  $f$  be a Boolean function admitting the  $t$ -block property, with  $t \leq d_\vee d_\wedge^{-1-\varepsilon}$ , for some  $\varepsilon > 0$ . Then,  $\mathfrak{bs}_0(f) \leq t(3\mathfrak{s}(f))^{1+\frac{1}{\varepsilon}}$ .*

**Proof.** Since  $t \leq d_\vee d_\wedge^{-1-\varepsilon}$ , we have that  $d_\wedge \leq \left(\frac{d_\vee}{t}\right)^{1/(1+\varepsilon)}$ . Substituting in Theorem 24, we have that  $\mathfrak{s}(f) \geq \frac{d_\vee}{3t(d_\vee/t)^{1/(1+\varepsilon)}}$ . After rearranging and simplifying, we get  $t^\varepsilon (3\mathfrak{s}(f))^{1+\varepsilon} \geq (d_\vee)^\varepsilon$ . We substitute Lemma 23, and simplify to obtain  $t(3\mathfrak{s}(f))^{1+\frac{1}{\varepsilon}} \geq \mathfrak{bs}_0(f)$ . ◀

### 4 Low Sensitivity Boolean functions

Gopalan et al. [11] show that functions with low sensitivity have concise descriptions, so consequently the number of such functions is small. Indeed, they show that knowing the values on a Hamming ball of radius  $2\mathfrak{s} + 1$  suffices. More precisely,

► **Theorem 26** (Gopalan et al. [11]). *Let  $f$  be a Boolean function of sensitivity  $\mathfrak{s}$ . Then, it is uniquely specified by its values on any ball of radius  $2\mathfrak{s}$ .*

We extend their observation to a more general one:

► **Theorem 27.** *Let  $f$  and  $g$  be two Boolean functions. If  $f$  and  $g$  coincide on a ball of radius  $\mathfrak{s}(f) + \mathfrak{s}(g)$  then,  $f = g$ .*

Before we prove Theorem 27, we note the following handy lemma:

► **Lemma 28.** *Let  $f$  and  $g$  be two Boolean functions. We have  $\mathfrak{s}(f \oplus g) \leq \mathfrak{s}(f) + \mathfrak{s}(g)$  and  $\mathfrak{bs}(f \oplus g) \leq \mathfrak{bs}(f) + \mathfrak{bs}(g)$ <sup>2</sup>.*

**Proof.** For any  $x \in \{0, 1\}^n$  and  $i \in [n]$ , if  $(f \oplus g)(x) \neq (f \oplus g)(x^i)$  then we have that either  $f(x) \neq f(x^i)$  or  $g(x) \neq g(x^i)$ . This implies, for every  $x \in \{0, 1\}^n$ ,  $\mathfrak{s}(f \oplus g, x) \leq \mathfrak{s}(f, x) + \mathfrak{s}(g, x)$ . Similarly, for any  $x \in \{0, 1\}^n$  and  $B \subseteq [n]$ , if  $(f \oplus g)(x) \neq (f \oplus g)(x^B)$  then we have that either  $f(x) \neq f(x^B)$  or  $g(x) \neq g(x^B)$ . This implies, for every  $x \in \{0, 1\}^n$ ,  $\mathfrak{bs}(f \oplus g, x) \leq \mathfrak{bs}(f, x) + \mathfrak{bs}(g, x)$ . ◀

**Proof of Theorem 27.** The proof is by contradiction. Suppose there exists  $a \in \{0, 1\}^n$  such that for every  $r \in \{0, 1\}^n$  of hamming weight at most  $\mathfrak{s}(f) + \mathfrak{s}(g)$ , we have that  $f(a \oplus r) = g(a \oplus r)$ . This implies that for every  $r \in \{0, 1\}^n$  with  $\|r\| \leq \mathfrak{s}(f) + \mathfrak{s}(g)$ , we have  $(f \oplus g)(a \oplus r) = 0$ . Consider  $x \in \{0, 1\}^n$  of the smallest hamming distance from  $a$  such that  $(f \oplus g)(x) = 1$ . If such a  $x$  does not exist then it implies that  $f \oplus g$  is the constant zero function. In that case we have that  $f = g$ , a contradiction. Therefore, let us suppose that  $x$  exists as described above. Let  $d$  be the hamming distance between  $x$  and  $a$ . We know that  $d > \mathfrak{s}(f) + \mathfrak{s}(g)$ . Additionally, we know that there are exactly  $d$  neighbors of  $x$  at hamming distance  $d - 1$  from  $a$ . Since,  $x$  was the input with the smallest distance from  $a$  such that  $(f \oplus g)(x) = 1$ , we know that the  $d$  neighbors of  $x$  at hamming distance  $d - 1$  from  $a$  all evaluate to 0 on  $(f \oplus g)$ . This means that  $\mathfrak{s}(f \oplus g, x) \geq d > \mathfrak{s}(f) + \mathfrak{s}(g)$ , which is a contradiction following Lemma 28. ◀

Next, we explore the tightness of Theorem 27.

► **Proposition 29.** *For every  $p, q \in \mathbb{N}$ , greater than 1, there exists Boolean functions  $f$  and  $g$  such that  $\mathfrak{s}(f) = p$ ,  $\mathfrak{s}(g) = q$ , and  $f$  and  $g$  coincide on a ball of radius  $\mathfrak{s}(f) + \mathfrak{s}(g) - 1$ .*

**Proof.** Without loss of generality we will assume that  $p \leq q$ . Fix  $p$  and  $q$ . We will build two function  $f$  and  $g$  on  $p + q$  variables. Let  $a \in \{0, 1\}^{p+q}$  be a special input defined as follows:  $\forall i \in [p + q]$ ,  $a_i = 1$  if and only if  $i = 1$ , or  $i > 2p$ , or  $i \neq 2p$  is even. Now we define  $f$  and  $g$  as follows:

$$f(x_1, \dots, x_{p+q}) = \begin{cases} 0 & \text{if } \sum_{i=1}^{2p} x_i < p \\ 1 & \text{if } \sum_{i=1}^{2p} x_i > p \\ \sum_{\substack{j=1, \\ j \leq 2p}} j \bmod 2 & \text{if } \sum_{i=1}^{2p} x_i = p \end{cases} \quad g(x) = \begin{cases} 0 & \text{if } x = a \\ f(x) & \text{otherwise.} \end{cases}$$

Now, we will show that  $\mathfrak{s}(f) = p$ . Fix  $x \in \{0, 1\}^{p+q}$ .  $x$  is not sensitive on the last  $q - p$  coordinates. If  $\sum_{i=1}^{2p} x_i < p - 1$  or  $\sum_{i=1}^{2p} x_i > p + 1$  then  $\mathfrak{s}(f, x) = 0$ . If  $\sum_{i=1}^{2p} x_i = p$  then  $\mathfrak{s}(f, x) = p$ . If  $\sum_{i=1}^{2p} x_i = p - 1$  then  $\mathfrak{s}(f, x) \leq p$ . This is because for any subset of  $[2p]$  of size  $p + 1$  there is both an odd number and an even number in the subset (by pigeonhole principle), and thus amongst its  $p + 1$  neighbors of hamming weight  $p$  (in the first  $2p$  coordinates) there

<sup>2</sup>  $\forall x \in \{0, 1\}^n$ ,  $(f \oplus g)(x) = f(x) \oplus g(x)$ .



## 15:12 On the Sensitivity Conjecture for Disjunctive Normal Forms

must be a neighbor which is not sensitive w.r.t.  $x$ . Similarly, we have that if  $\sum_{i=1}^{2p} x_i = p + 1$  then  $\mathfrak{s}(f, x) \leq p$ .

Next, we will show that  $\mathfrak{s}(g) = q$ . Fix  $x \in \{0, 1\}^{p+q}$ . If  $x$  is not in the hamming ball of radius 1 centered at  $a$  then,  $s(g, x) = s(f, x) \leq p \leq q$ . If  $x = a$  then, it is sensitive on all the last  $q - p$  coordinates and has  $p$  sensitive neighbors in the first  $2p$  coordinates. Thus,  $\mathfrak{s}(g, a) = q$ . If  $x$  is a neighbor of  $a$  through one of the last  $q - p$  coordinates (i.e., assuming  $q - p > 0$ ) then  $\mathfrak{s}(g, x) = p + 1 \leq q$ . If  $x$  is a neighbor of  $a$  through one of the first  $2p$  coordinates then, we can assume  $g(x) = 1$ . This means hamming weight of  $x$  in first  $2p$  coordinates is  $p + 1$  and we know that it is not sensitive on the last  $q - p$  coordinates. From the definition of  $a$ , we know that there is at least one neighbor of  $x$  of hamming weight  $p$  in the first  $2p$  coordinates such that its value on  $g$  is the same as  $g(x)$ . Therefore  $\mathfrak{s}(g, x) \leq p \leq q$ .

Finally, we claim that  $f$  and  $g$  coincide on the ball of radius  $p + q - 1$  centered at  $(a \oplus \vec{1})$  (follows from the construction of  $g$ ). This completes the proof as  $f$  and  $g$  are distinct ( $f(a) \neq g(a)$ ) and to distinguish between them by a ball centered at  $(a \oplus \vec{1})$ , we need to consider a ball of radius  $p + q$ . ◀

In the case of monotone Boolean functions, we can improve upon the results in Theorem 26 and Theorem 27 as follows: any monotone Boolean function  $f$  is uniquely specified by its values on the ball of radius  $\mathfrak{s}$  centered at  $0^n$ . This is because, for any input  $x$  of hamming weight greater than  $\mathfrak{s}(f)$ ,  $f(x)$  is equal to 1 if at least one of its neighbors of hamming weight  $|x| - 1$  is evaluated to 1 on  $f$ . In other words,

$$f(x) = \bigvee_{\substack{y=x \oplus e_i \\ |y|=|x|-1}} f(y).$$

Furthermore, this result is tight because Wegener's monotone Boolean function [20]  $f$  of sensitivity  $\frac{1}{2} \log n + \frac{1}{4} \log \log n + \mathcal{O}(1)$  is identical to the constant zero function on the ball of radius  $\mathfrak{s}(f) - 1$  centered at  $0^n$ .

### 5 The Sensitivity Conjecture: A Computational Perspective

We would like to briefly discuss in this section a new perspective on the sensitivity conjecture. Consider a strong version of the sensitivity conjecture which was suggested by Nisan and Szegedy [16]: for every Boolean function  $f$ , we have  $\mathfrak{bs}(f) \leq c \cdot \mathfrak{s}(f)^2$ , for some constant  $c$ . Let us assume that the above conjecture is true. We note here that there is no evidence or reason to refute this strong version of the sensitivity conjecture. Now consider a computational problem called the *sensitivity problem* defined based on this assumption.

► **Definition 30 (Sensitivity Problem).** Given a circuit  $C : \{0, 1\}^n \rightarrow \{0, 1\}$ ,  $x \in \{0, 1\}^n$ , and blocks  $B_1, \dots, B_k$ , the sensitivity problem is to find  $y \in \{0, 1\}^n$  such that  $\mathfrak{s}(C, y) \geq \sqrt{\mathfrak{bs}(C, x, B_1, \dots, B_k)}/c$ .

A solution to the sensitivity problem is guaranteed to exist and a solution can be verified in  $\text{poly}(n)$  time, thus the problem is in **TFNP**. We wonder if the proof of the sensitivity conjecture would give us an efficient algorithm to solve this problem in **P**?

We investigated the proofs of the sensitivity conjecture for restricted classes of Boolean functions that exist in literature. In each of these proofs we indeed find an efficient algorithm to solve the above problem in **P**. For instance, consider the class of Boolean functions admitting the normalized block property. In this case, the computational problem would be that given a DNF  $\Phi$ ,  $x \in \{0, 1\}^n$ , and blocks  $B_1, \dots, B_k$ , find  $y \in \{0, 1\}^n$  such that



$s(\Phi, y) \geq \sqrt{\text{bs}(\phi, x, B_1, \dots, B_k)/2}$  or find two clauses in  $\Phi$  which violate  $\Phi$  admitting the block property. This problem like the sensitivity problem is in **TFNP**. However, the proof of Lemma 19 gives us an efficient algorithm to find an input  $a_1$  with sensitivity  $\lceil \frac{d_v}{2d_\wedge - 1} \rceil$  and the proof of Lemma 20 gives us an efficient algorithm to find an input  $a_2$  with sensitivity  $\lceil \frac{d_\wedge}{2} \rceil$ . Since,  $\text{bs}(\phi, x, B_1, \dots, B_k) \leq \text{bs}(\Phi) = d_v$ , either  $a_1$  or  $a_2$  is a solution to our problem (assuming there is no violation to the block property of  $\Phi$ ). Thus the computational problem in the case of functions admitting the block property is in **P**.

Similarly, for every monotone function  $f$ , and every input  $x$  we have  $s(f, x) = \text{bs}(f, x)$  [15]. Therefore, for the computational version of the sensitivity problem adapted to the monotone restriction, the input  $x$  will be a trivial solution and thus the computational problem would be in **P**. Finally, even for the case of min-term transitive functions, we have an efficient algorithm implicit in the proof of Chakraborty [6] who showed that for any min-term transitive function  $f$ ,  $\text{bs}(f) \leq 2s(f)^2$ .

Returning to ponder on the existence of efficient algorithms for the sensitivity problem, while it is related to the sensitivity conjecture, it is possible that the sensitivity conjecture is true but there is no efficient algorithm for the sensitivity problem. Similarly, it is possible that an efficient algorithm for the sensitivity problem is found without resolving the sensitivity conjecture (in this case the sensitivity problem should be considered to be in **NP** and not in **TFNP**). However, our progress on the sensitivity conjecture under various restricted settings seem to be by finding a vertex of high sensitivity by starting from a given input with high block sensitivity. Therefore, studying various restrictions on models of computations for Boolean functions seems to be the right direction to pursue, in order to make progress on the sensitivity conjecture.

## 6 Conclusion

In this paper, we motivate the study of the sensitivity conjecture through restrictions on a model of computation. In this regard, we introduced a structural restriction on DNFs representing Boolean functions called the normalized block property. We showed that the examples of Boolean functions that are popular in literature for having a quadratic separation between sensitivity and block sensitivity admit this property. More importantly, we showed that the sensitivity conjecture is true for the class of Boolean functions admitting the normalized block property. Furthermore, we extended a result of Gopalan et al. [11] and also provided matching lower bounds for our results. Finally, we motivated a new computational problem about finding an input with (relatively) high sensitivity, with respect to the block sensitivity for a given input.

**Acknowledgements.** We would like to thank Satya Lokam for discussions and encouraging us to work on the sensitivity conjecture. In particular, Karthik would like to thank him for the internship at Microsoft Research. We would like to thank Avishay Tal for pointing out Corollary 21. We would like to thank Roei Tell for motivation and discussions. We would like to thank Andris Ambainis and Krišjānis Prūsis for pointing out an error in a first version of Theorem 16 and for highlighting the relation between this theorem and the result in [2]. Finally, we would like to thank the anonymous reviewers for helping us improve the presentation of the paper.

## References

- 1 Andris Ambainis, Mohammad Bavarian, Yihan Gao, Jieming Mao, Xiaoming Sun, and Song Zuo. Tighter relations between sensitivity and other complexity measures. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 101–113, 2014. doi:10.1007/978-3-662-43948-7\_9.
- 2 Andris Ambainis and Krisjanis Prusis. A tight lower bound on certificate complexity in terms of block sensitivity and sensitivity. In *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 33–44, 2014. doi:10.1007/978-3-662-44465-8\_4.
- 3 Andris Ambainis and Xiaoming Sun. New separation between  $s(f)$  and  $bs(f)$ . *Electronic Colloquium on Computational Complexity (ECCC)*, 18:116, 2011. URL: <http://eccc.hpi-web.de/report/2011/116>.
- 4 Mitali Bafna, Satyanarayana V. Lokam, Sébastien Tavenas, and Ameya Velingker. On the sensitivity conjecture for read-k formulas. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 – Kraków, Poland*, pages 16:1–16:14, 2016. doi:10.4230/LIPICs.MFCS.2016.16.
- 5 Harry Buhman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.
- 6 Sourav Chakraborty. On the sensitivity of cyclically-invariant boolean functions. *Discrete Mathematics & Theoretical Computer Science*, 13(4):51–60, 2011.
- 7 Stephen A. Cook and Cynthia Dwork. Bounds on the time for parallel ram’s to compute simple functions. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 231–233, 1982. doi:10.1145/800070.802196.
- 8 Stephen A. Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.*, 15(1):87–97, 1986. doi:10.1137/0215006.
- 9 Andrew Drucker. Block sensitivity of minterm-transitive functions. *Theor. Comput. Sci.*, 412(41):5796–5801, 2011. doi:10.1016/j.tcs.2011.06.025.
- 10 Justin Gilmer, Michael Saks, and Srikanth Srinivasan. Composition limits and separating examples for some boolean function complexity measures. In *2013 IEEE Conference on Computational Complexity*, pages 185–196. IEEE, 2013.
- 11 Parikshit Gopalan, Noam Nisan, Rocco A. Servedio, Kunal Talwar, and Avi Wigderson. Smooth boolean functions are easy: Efficient algorithms for low-sensitivity functions. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 59–70, 2016. doi:10.1145/2840728.2840738.
- 12 Pooya Hatami, Raghav Kulkarni, and Denis Pankratov. Variations on the sensitivity conjecture. *Theory of Computing, Graduate Surveys*, 4:1–27, 2011. doi:10.4086/toc.gs.2011.004.
- 13 Claire Kenyon and Samuel Kutin. Sensitivity, block sensitivity, and l-block sensitivity of boolean functions. *Inf. Comput.*, 189(1):43–53, 2004. doi:10.1016/j.ic.2002.12.001.
- 14 Chengyu Lin and Shengyu Zhang. Sensitivity conjecture and log-rank conjecture for functions with small alternating numbers. *CoRR*, abs/1602.06627, 2016. URL: <http://arxiv.org/abs/1602.06627>.
- 15 Noam Nisan. CREW prams and decision trees. *SIAM J. Comput.*, 20(6):999–1007, 1991. doi:10.1137/0220062.
- 16 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational Complexity*, 4:301–313, 1994. doi:10.1007/BF01263419.

- 17 David Rubinfeld. Sensitivity vs. block sensitivity of boolean functions. *Combinatorica*, 15(2):297–299, 1995. doi:10.1007/BF01200762.
- 18 Xiaoming Sun. Block sensitivity of weakly symmetric functions. *Theor. Comput. Sci.*, 384(1):87–91, 2007. doi:10.1016/j.tcs.2007.05.020.
- 19 Madars Virza. Sensitivity versus block sensitivity of boolean functions. *Inf. Process. Lett.*, 111(9):433–435, 2011. doi:10.1016/j.ipl.2011.02.001.
- 20 Ingo Wegener. The critical complexity of all (monotone) boolean functions and monotone graph properties. *Information and Control*, 67(1-3):212–222, 1985. doi:10.1016/S0019-9958(85)80036-X.



# The Zero-Error Randomized Query Complexity of the Pointer Function

Jaikumar Radhakrishnan<sup>1</sup> and Swagato Sanyal<sup>2</sup>

- 1 Tata Institute of Fundamental Research, India  
jaikumar@tifr.res.in
- 2 Tata Institute of Fundamental Research, India  
swagato.sanyal@tifr.res.in

---

## Abstract

The pointer function of Göös, Pitassi and Watson and its variants have recently been used to prove separation results among various measures of complexity such as deterministic, randomized and quantum query complexity, exact and approximate polynomial degree, etc. In particular, Ambainis et al. (STOC 2016) obtained the widest possible (quadratic) separations between deterministic and zero-error randomized query complexity, as well as between bounded-error and zero-error randomized query complexity by considering *variants* of this pointer function.

However, as Ambainis et al. pointed out in their work, the precise zero-error complexity of the original pointer function was not known. We show a lower bound of  $\tilde{\Omega}(n^{3/4})$  on the zero-error randomized query complexity of the pointer function on  $\Theta(n \log n)$  bits; since an  $\tilde{O}(n^{3/4})$  upper bound was already shown by Mukhopadhyay and Sanyal (FSTTCS 2015), our lower bound is optimal up to polylog factors. We, in fact, consider a generalization of the original function and obtain lower bounds for it that are optimal up to polylog factors.

**1998 ACM Subject Classification** F.1.1 [Models of Computation] Relations between models, F.1.2 [Modes of Computation] Probabilistic computation

**Keywords and phrases** Deterministic Decision Tree, Randomized Decision Tree, Query Complexity, Models of Computation

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.16

## 1 Introduction

Understanding the relative power of various models of computation is a central goal in complexity theory. In this paper, we focus on one of the simplest models for computing Boolean functions – the query model or the decision tree model. In this model, the algorithm is required to determine the value of a Boolean function by querying individual bits of the input, possibly adaptively. The computational resource we seek to minimize is the number of queries for the worst-case input. That is, the algorithm is charged each time it queries an input bit, but not for its internal computation.

There are several variants of the query model, depending on whether randomization is allowed, and on whether error is acceptable. Let  $D(f)$  denote the deterministic query complexity of  $f$ , that is, the maximum number of queries made by the algorithm for the worst-case input; let  $R(f)$  denote the maximum number of queries made by the best randomized algorithm that errs with probability at most  $1/3$  (say) on the worst-case input. Let  $R_0(f)$  be the zero-error randomized query complexity of  $f$ , that is, the expected number of queries made for the worst-case input by the best randomized algorithm for  $f$  that answers correctly on every input.



© Jaikumar Radhakrishnan and Swagato Sanyal;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 16; pp. 16:1–16:13



Leibniz International Proceedings in Informatics

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

The relationships between these query complexity measures have been extensively studied in the literature. That randomization can lead to significant savings has been known for a long time. Snir [10] showed a  $O(n^{\log_4 3})$  randomized linear query algorithm (a more powerful model than what we discussed) for complete binary NAND tree function for which the deterministic linear query complexity is  $\Omega(n)$ . Later on Saks and Wigderson [9] determined the zero-error randomized query complexity of the complete binary NAND tree function to be  $\Theta(n^{0.7536\dots})$ . They also presented a result of Ravi Boppana which states that the uniform rooted ternary majority tree function has randomized zero-error query complexity  $O(n^{0.893\dots})$  and deterministic query complexity  $n$ . All these examples showed that randomized query complexity can be substantially lower than its deterministic counterpart. On the other hand, Nisan showed that the  $R(f) = \Omega(D(f)^{1/3})$  [8]. Blum and Impagliazzo [3], Tardos [11], Hartmanis and Hemachandra [6] independently showed that  $R_0(f) = \Omega(D(f)^{1/2})$ . Thus, the question of the largest separation between deterministic and randomized complexity remained open. Indeed, Saks and Wigderson conjectured that the complete binary NAND tree function exhibits the widest separation possible between these two measures of complexity.

► **Conjecture 1** ([9]). *For any boolean function  $f$  on  $n$  variables,  $R_0(f) = \Omega(D(f)^{0.753\dots})$ .*

This conjecture was recently refuted independently by Ambainis *et al.* [2] and Mukhopadhyay and Sanyal [7]. Both works based their result on the pointer function introduced by Göös, Pitassi and Watson [5], who used this function to show a separation between deterministic decision tree complexity and unambiguous non-deterministic decision tree complexity. In Section 2, we present the formal definition of the function  $\text{GPW}^{r \times s}$ , which is a Boolean function on  $\tilde{\Theta}(rs)$  bits.

Mukhopadhyay and Sanyal [7] used  $\text{GPW}^{s \times s}$  to obtain the following refutation of Conjecture 1:  $R_0(\text{GPW}^{s \times s}) = \tilde{O}(s^{1.5})$  while  $D(\text{GPW}^{s \times s}) = \Omega(s^2)$ . While this shows that  $\text{GPW}^{s \times s}$  witnesses a wider separation between deterministic and zero-error randomized query complexities than conjectured, the separation shown is not the widest possible for a Boolean function. Independently, Ambainis *et al.* modified  $\text{GPW}^{s \times s}$  in subtle ways, to establish the widest possible (near-quadratic) separation between deterministic and zero-error randomized query complexity, and between zero-error randomized and bounded-error randomized query complexities.

Ambainis *et al.* [2] pointed out, however, that the precise zero-error randomized query complexity (i.e.  $R_0(\text{GPW}^{s \times s})$ ) was not known. One could ask if the optimal separation demonstrated by Ambainis *et al.* is also witnessed by  $\text{GPW}^{s \times s}$  itself. In this work, we prove a near-optimal lower bound on the zero-error randomized query complexity of  $\text{GPW}^{r \times s}$ , which is slightly more general than the  $\text{GPW}^{s \times s}$  considered in earlier works.

► **Theorem 2** (Main theorem).  $R_0(\text{GPW}^{r \times s}) = \tilde{\Omega}(r + \sqrt{rs})$ .

Such a result essentially claims that randomized algorithms cannot efficiently locate certificates for the function. This would be true, for example, if the function could be shown to require large certificates, since the certificate complexity of a function is clearly a lower bound on its zero-error randomized complexity. This straightforward approach does not yield our lower bound, as the certificate complexity of  $\text{GPW}^{r \times s}$  is  $\tilde{O}(r + s)$ . In our proof, we set up a special distribution on inputs, and by analyzing the expansion properties of the pointers, show that a certificate will evade a randomized algorithm that makes only a small number of queries. In fact, the distribution we devise is almost entirely supported on inputs  $X$  for which  $\text{GPW}^{r \times s}(X) = 0$ . This is not an accident: a randomized algorithm can quickly find a certificate for inputs  $X$  if  $\text{GPW}^{r \times s}(X) = 1$  (see Theorem 5 below).

It follows from Theorem 2 that the algorithm of Mukhopadhyay and Sanyal [7] is optimal up to polylog factors.

► **Corollary 3.**  $R_0(\text{GPW}^{s \times s}) = \tilde{\Omega}(s^{1.5})$ .

In addition to nearly determining the zero-error complexity of the original  $\text{GPW}^{s \times s}$  function, our result has two interesting consequences.

- (a) The above mentioned result of Mukhopadhyay and Sanyal [7] showed that  $R_0(\text{GPW}^{s \times s}) = \tilde{O}(D(\text{GPW}^{s \times s})^{0.75})$ . Our main theorem shows that  $\text{GPW}^{s \times s}$  cannot be used to show a significantly better separation between the deterministic and randomized zero-error complexities (ignoring polylog factors). However, the function  $\text{GPW}^{s^2 \times s}$  allows us to derive a better separation<sup>1</sup>:  $R_0(\text{GPW}^{s^2 \times s}) = O(D(\text{GPW}^{s^2 \times s})^{2/3})$ . Our main theorem shows that this is essentially the best separation that can be derived from  $\text{GPW}^{r \times s}$  by varying  $r$  relative to  $s$ , so this method cannot match the near-quadratic separation between these measures, which was shown by Ambainis *et al.* [2] by considering a *variant* of the  $\text{GPW}^{s \times s}$  function.
- (b)  $\text{GPW}^{s \times s}$  exposes a non-trivial polynomial separation between the bounded-error and zero-error randomized query complexities:  $R(\text{GPW}^{s \times s}) = \tilde{O}(R_0(\text{GPW}^{s \times s})^{2/3})$ . This falls short of the near-quadratic separation shown by Ambainis *et al.* [2], but note that before that result no separation between these measures was known.

## 2 The GPW function

The input  $X$  to the *pointer function*,  $\text{GPW}^{r \times s}$ , is arranged in an array with  $r$  rows and  $s$  columns. The cell  $X[i, j]$  of the array contains two pieces of data, a bit  $b_{ij} \in \{0, 1\}$  and a pointer  $\text{ptr}_{ij} \in ([r] \times [s]) \cup \{\perp\}$ .

Let  $\mathcal{A}$  denote the set of all such arrays. The function  $\text{GPW}^{r \times s} : \mathcal{A} \rightarrow \{0, 1\}$  is defined as follows:  $\text{GPW}^{r \times s}(X) = 1$  if and only if the following three conditions are satisfied.

1. There is a unique column  $j^*$  such that for all rows  $i \in [r]$ , we have  $b_{ij^*} = 1$ .
2. In this column  $j^*$ , there is a unique row  $i^*$  such that  $\text{ptr}_{i^*j^*} \neq \perp$ .
3. Now, consider the sequence of locations  $(p_k : k = 0, 1, \dots, s-1)$ , defined as follows: let  $p_0 = (i^*, j^*)$ , and for  $k = 0, 1, \dots, s-2$ , let  $p_{k+1} = \text{ptr}_{p_k}$ . Then,  $p_0, p_1, \dots, p_{s-1}$  lie in distinct columns of  $X$ , and  $b_{p_k} = 0$  for  $k = 1, 2, \dots, s-1$ . In other words, there is a *chain of pointers*, which starts from the unique location in column  $j^*$  with a non-null pointer, visits all other columns in exactly  $s-1$  steps, and finds a 0 in each location it visits (except the first).

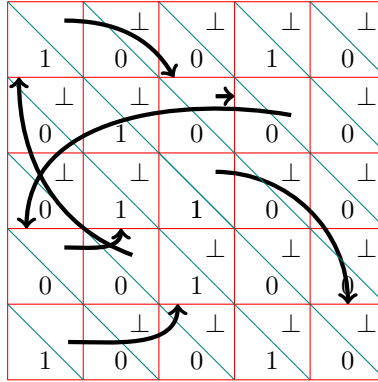
Note that  $\text{GPW}^{r \times s}$  can be thought of as a Boolean function on  $\Theta(rs \log rs)$  bits.

### Upper Bound

The pointer function  $\text{GPW}^{r \times s}$ , as defined above, is parameterized by two parameters,  $r$  and  $s$ . Göös, Pitassi and Watson [5] focus on the special case where  $r = s$ . Mukhopadhyay and Sanyal [7] also state their zero-error randomized algorithm with  $\tilde{O}(s^{1.5})$  queries for this special case; however, it is straightforward to extend their algorithm so that it applies to the function  $\text{GPW}^{r \times s}$  and yields the following upper bound.

► **Theorem 4.**  $R_0(\text{GPW}^{r \times s}) = \tilde{O}(r + \sqrt{rs})$ .

<sup>1</sup> In [1], a similar separation between  $R(\text{GPW}^{s^2 \times s})$  and  $D(\text{GPW}^{s^2 \times s})$  is mentioned.



■ **Figure 1** Input to  $\text{GPW}^{r \times s}$  for  $r = 5, s = 5$ .

Mukhopadhyay and Sanyal also gave a one-sided error randomized query algorithm that makes  $\tilde{O}(s)$  queries on average but never errs on inputs  $X$ , where  $\text{GPW}^{s \times s}(X) = 1$ . Again a straightforward extension yields the following.

► **Theorem 5.** *There is a randomized query algorithm that makes  $\tilde{O}(r + s)$  queries on each input, computes  $\text{GPW}^{r \times s}$  correctly on each input with probability at least  $1/3$ , and in addition never errs on inputs  $X$  where  $\text{GPW}^{r \times s}(X) = 1$ .*

Theorem 2, thus, completely determines the deterministic and all randomized query complexities of a more general function  $\text{GPW}^{r \times s}$ .

## 2.1 The distribution

To show our lower bound, we will set up a distribution on inputs in  $\mathcal{A}$ . Let  $V$  be the locations in the first  $s/2$  columns, i.e.,  $V = [r] \times [s/2]$ ; let  $W$  be the locations in the last  $s/2$  columns, i.e.,  $W = [r] \times ([s] \setminus [s/2])$ . In order to describe the random input  $X$ , we will need the following definitions.

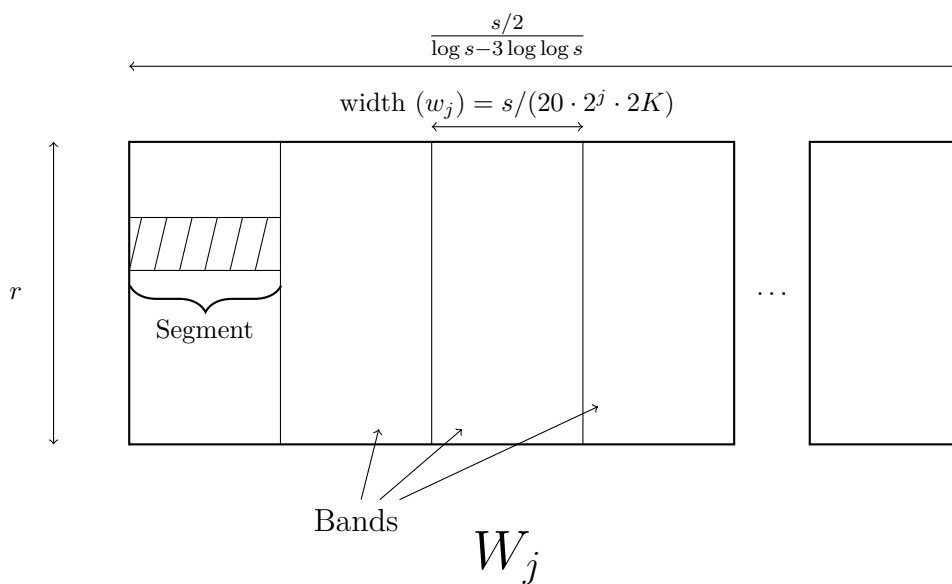
### Pointer chain

For an input in  $\mathcal{A}$ , we say that a sequence of locations  $\mathbf{p} = \langle \ell_0, \ell_1, \ell_2, \dots, \ell_k \rangle$  is a pointer chain, if for  $i = 0, 1, \dots, k-1$ ,  $\text{ptr}_{\ell_i} = \ell_{i+1}$ ; the location  $\ell_0$  is the *head* of the  $\mathbf{p}$  and is denoted by  $\text{head}(\mathbf{p})$ ; similarly,  $\ell_k$  is the *tail* of  $\mathbf{p}$  and is denoted by  $\text{tail}(\mathbf{p})$ . Note that  $\text{ptr}_{\ell_k}$  is not specified as part of the definition of pointer chain  $\mathbf{p}$ ; in particular, it is allowed to be  $\perp$ .

### Random pointer chain

To build our random input  $X$ , we will assign the pointer values of the various cells of  $X$  randomly so that they form appropriate pointer chains. For a set of locations  $S$  we build a *random pointer chain on  $S$*  as follows. First, we uniformly pick a permutation of  $S$ , say  $\langle \ell_0, \ell_1, \dots, \ell_k \rangle$ . Then, we set  $\text{ptr}_{\ell_i} = \ell_{i+1}$  (for  $i = 0, 1, \dots, k-1$ ). We will make such random assignments for sets  $S$  consisting of consecutive locations in some row of  $W$ . We call the special (deterministic) chain that starts at the first (leftmost) location of  $S$ , visits the next, and so on, until the last (rightmost) location, a *path*. Given two pointer chains  $\mathbf{p}_1$  and  $\mathbf{p}_2$  on disjoint sets of locations  $S_1$  and  $S_2$ , we may set  $\text{ptr}_{\text{tail}(\mathbf{p}_1)} = \text{head}(\mathbf{p}_2)$ , and obtain a single pointer chain on  $S_1 \cup S_2$ , whose head is  $\text{head}(\mathbf{p}_1)$  and tail is  $\text{tail}(\mathbf{p}_2)$ . We will refer to this operation as the *concatenation* of  $\mathbf{p}_1$  and  $\mathbf{p}_2$ .





■ **Figure 2** Bands and segments inside block  $W_j$ .

We are now ready to define the random input  $X$ . We will assume that  $r \gg \log s$ , because otherwise  $\sqrt{r}s = \tilde{O}(s)$ , and Theorem 2 follows from the certificate complexity lower bound of  $\tilde{\Omega}(r + s)$  on  $R_0(\text{GPW}^{r \times s})$  (see the sentence following Assumption 6 below). First, consider  $W$ . For all  $\ell \in W$ , we set  $b_\ell = 0$ . To describe the pointers corresponding to  $W$ , we partition the columns of  $W$  into  $K := \log s - 3 \log \log s$  blocks,  $W_1, \dots, W_K$ , where  $W_1$  consists of the first  $s/(2K)$  columns of  $W$ , then  $W_2$  consists of the next  $s/(2K)$  columns, and so on.

$$\left[ \begin{array}{c|cccc} V & W_1 & W_2 & \dots & W_K \end{array} \right]$$

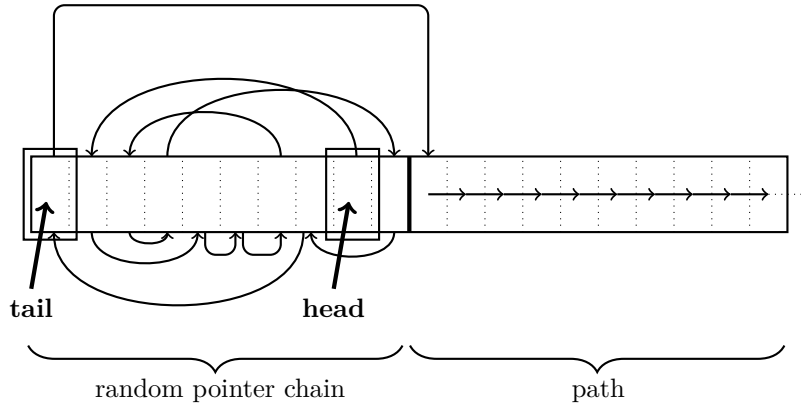
The block  $W_j$ , will be further divided into *bands*; however, the number of bands in different  $W_j$  will be different. There will be  $20 \cdot 2^j$  bands in  $W_j$ , each consisting of  $w_j := s / (20 \cdot 2^j \cdot 2K)$  contiguous chosen columns (note that  $w_j \gg \log s$  by our choice of  $K$ ). See Figure 2.

Each such band will have  $r$  rows; the locations in a single row of a band will be called a *segment*; we will divide each segment into two equal parts, left and right, each with  $w_j/2$  columns.

We are now ready to specify the pointers in each segment of  $W_i$ . In the first half of each segment we place a random (uniformly chosen) pointer chain; in the right half we place a path starting at its leftmost cell and leading to its rightmost cell. See Figure 3. Once all pointer chains in all the segments in a given row are in place, we concatenate them from left to right. All pointers in the last column of  $W$  are set to  $\perp$ . In the resulting input, each row of  $W$  is a single pointer chain with head in the leftmost segment of  $W_1$  and tail in the last column of  $W$ . This completes the description of  $X$  for the locations in  $W$ .

Next, we consider locations in  $V$ . Let  $q := 500 \log s / \sqrt{r}$ . Notice that by our assumption,  $q < 1$ . Independently, for each location  $\ell \in V$ :

- with probability  $q$ , set  $b_\ell = 0$  and  $\text{ptr}_\ell$  to be a random location that is in the *left half* of



■ **Figure 3** A segment consists of a random pointer chain concatenated with a path.

- some segment in  $W$  (that is, among all locations that fall in the left half of some segment, pick one at random and set  $\text{ptr}_\ell$  to that location);
  - with probability  $1 - q$ , set  $b_\ell = 1$  and  $\text{ptr}_\ell = \perp$ .
- This completes the description of the random input  $X$ .

### 3 The lower bound for $\text{GPW}^{r \times s}$

We will consider algorithms that are given query access to the input bits of  $\text{GPW}^{r \times s}$ . A location  $\ell \in [r] \times [s]$  of an input  $X \in \mathcal{A}$  is said to be queried if either  $b_\ell$  is queried, or some bit in the encoding of  $\text{ptr}_\ell$  is queried. By *number of queries*, we will always mean the number of locations queried. A lower bound on the number of locations queried is clearly a lower bound on the number of bits queried.

It can be shown that the certificate complexity of  $\text{GPW}^{r \times s}$  is  $\Omega(r + s)$ ; hence  $R_0(\text{GPW}^{r \times s}) = \Omega(r + s)$ . It remains to show that any zero-error randomized query algorithm for  $\text{GPW}^{r \times s}$  must make  $\Omega(\sqrt{rs}/\text{polylog}(s))$  queries in expectation. We will assume that there is a significantly more efficient algorithm and derive a contradiction.

► **Assumption 6.** *There is a zero-error randomized algorithm that makes at most  $\sqrt{rs}/(\log s)^5$  queries in expectation (taken over the algorithm's coin tosses) on every input  $X$ .*

If  $r < (\log s)^3$  (say), then this assumption immediately leads to a contradiction because  $R_0(\text{GPW}^{r \times s}) = \Omega(s)$ . So, we will assume that  $r \geq (\log s)^3$ .

Consider inputs  $X$  drawn according to the distribution described in the previous section. Since with probability  $1 - o(1)$  every column of  $X$  has at least one zero (see Lemma 10 (a)),  $\text{GPW}^{r \times s}(X) = 0$  with probability  $1 - o(1)$ ; thus, the algorithm returns the answer 0 with probability  $1 - o(1)$ . Taking expectation over inputs  $X$  and the algorithm's coin tosses, the expected number of queries made by the algorithm is at most  $\sqrt{rs}/(\log s)^5$ . Using Markov's inequality, with probability  $1 - o(1)$ , the algorithm stops after making at most  $\sqrt{rs}/(\log s)^4$  queries. By truncating the long runs and fixing the random coin tosses of the algorithm, we obtain a deterministic algorithm. Hence we have the following.

► **Proposition 7.** *If Assumption 6 holds, then there is a deterministic algorithm that (i) queries at most  $\sqrt{rs}/(\log s)^4$  locations, (ii) never returns a wrong answer (it might give no answer on some inputs), and (iii) returns the answer 0 with probability  $1 - o(1)$  for the random input  $X$ .*

Fix such a deterministic query algorithm  $\mathcal{Q}$ . In the next section, we formally establish the following.

► **Lemma 8** (Stitching lemma). *With probability  $1 - o(1)$  over the choices of  $X$ , there is an input  $X' \in \mathcal{A}$  that differs from  $X$  only in locations not probed by  $\mathcal{Q}$  such that  $\text{GPW}^{r \times s}(X') = 1$ .*

Thus, with high probability,  $\mathcal{Q}(X') = \mathcal{Q}(X) = 0$ . This contradicts Proposition 7 (ii). This immediately implies Theorem 2.

## 4 The approach

In this section, we will work with the algorithm  $\mathcal{Q}$  that is guaranteed to exist by Proposition 7. For an input  $X \in \mathcal{A}$  to  $\text{GPW}^{r \times s}$ , let  $G_X = (V', W', E)$  be a bipartite graph, where  $V'$  is the set of columns of  $V$  and  $W'$  is the set of all bands in all blocks of  $W$ . The edge set  $E(G_X)$  is obtained as follows. Recall that pointers from  $V$  lead to segments in  $W$ . Each such segment contains a pointer chain. For a location  $\ell$  in such a chain, let  $\text{pred}(\ell)$  denote the location  $\ell'$  that precedes  $\ell$  in the chain (if  $\ell$  is the head, then  $\text{pred}(\ell)$  is undefined); thus,  $\text{ptr}_{\ell'} = \ell$ . We include the edge  $(j, \beta)$  (connecting column  $j \in V'$  to band  $\beta \in W'$ ) in  $E(G_X)$  if the following holds:

*There is a location  $v$  in column  $j$  and a segment  $p$  in some row of band  $\beta$  such that*

- (c1)  $\text{ptr}_v \in p$ , that is,  $\text{ptr}_v$  is non-null and points to a location in the left half of segment  $p$  (notice that this implies that  $b_v = 0$ );
- (c2)  $\text{pred}(\text{ptr}_v)$  is well defined and is not probed by  $\mathcal{Q}$ ;
- (c3)  $\mathcal{Q}$  makes fewer than  $|p|/4$  probes in segment  $p$ . (Note that this implies that there is a location in the right half of  $p$  that is left unprobed by  $\mathcal{Q}$  and that is not the last location of the segment.)

In the next section, we will show the following.

► **Lemma 9** (Matching lemma). *With probability  $1 - o(1)$  over the choice of  $X$ , for every subset  $R \subseteq V'$  of at most  $s/(\sqrt{r}(\log s)^4)$  columns, there is a matching in  $G_X$  that saturates  $R$ .*

In this section, we will show how Lemma 9 enables us to modify the input  $X$  to obtain an input  $X'$  for which  $\text{GPW}^{r \times s}(X') = 1$ , thereby establishing Lemma 8 (the stitching lemma).

► **Lemma 10.**

- (a) *With probability  $1 - o(1)$ , each column  $j$  of the input  $X$  has a location  $\ell$  such that  $b_\ell = 0$ .*
- (b) *With probability  $1 - o(1)$ , there is a column  $j \in [s/2]$  such that  $\mathcal{Q}$  does not read any location  $\ell$  in column  $j$  with  $b_\ell = 0$ .*

**Proof.**

- (a) All the bits in the columns in  $[s] \setminus [s/2]$  are 0. We show that with high probability, each column in  $V'$  has a 0. The probability that a particular column in  $V'$  does not have any 0 is  $(1 - 500 \log s / \sqrt{r})^r \leq s^{-\Omega(\sqrt{r})}$ . Thus the probability that there is a column  $j \in V'$  which does not have any 0 is at most  $(s/2) \cdot s^{-\Omega(\sqrt{r})} = o(1)$ .
- (b) By Proposition 7,  $\mathcal{Q}$  makes  $t \leq s\sqrt{r}/(\log s)^4$  queries. For  $i = 1, 2, \dots, t$ , let  $R_i$  be the indicator variable for the event that in the  $i$ -th query,  $\mathcal{Q}$  reads a 0 from  $V$ . Then, the expected number of 0's read by  $\mathcal{Q}$  in  $V$  is (we assume that  $\mathcal{Q}$  does not read the same location twice)

$$\sum_{i=1}^t \mathbb{E}[R_i] \leq t \cdot 500 \log s / \sqrt{r} \leq 500s / (\log s)^3.$$

By Markov's inequality, with probability  $1 - o(1)$ , the number of 0's read by  $\mathcal{Q}$  is less than  $s/2$ . It follows, that there is a column in  $V$  in which  $\mathcal{Q}$  has read no 0. ◀

**Proof of Lemma 8.** Assume that the high probability events of Lemmas 9 and 10 hold. This happens with probability  $1 - o(1)$ . We will now describe a sequence of modifications to the input  $X$  at locations not queried by  $\mathcal{Q}$  to transform it into an input  $X'$  such that  $\text{GPW}^{r \times s}(X') = 1$ . Let  $j^* \in V'$  be the column in  $V$  guaranteed by Lemma 10 (b). Define  $A_0 = \{\text{col}_1, \dots, \text{col}_N\} \subseteq V' \setminus \{j^*\}$  to be the set of columns in  $V' \setminus \{j^*\}$  that are not completely read by  $\mathcal{Q}$  (i.e. each column in  $A_0$  has a location unread by  $\mathcal{Q}$ ). Let  $\ell_i$  be a location in the column  $\text{col}_i$  that is unread by  $\mathcal{Q}$ . We first make the following changes to  $X$ , with the aim of starting a pointer chain at column  $j^*$  that passes through  $\text{col}_1, \text{col}_2, \dots, \text{col}_N$ .

- (i) For each unread location  $\ell$  in the column  $j^*$ , set  $b_\ell$  to 1. From the definition of  $j^*$ , the bits of the read locations are already 1.
- (ii) Let  $\ell^*$  be the first unread location of  $j^*$  (i.e. the location with the least row index). Set  $\text{ptr}_{\ell^*}$  to  $\ell_1$ .
- (iii) For each unread location  $\ell \neq \ell^*$  in column  $j^*$ , set  $\text{ptr}_\ell$  to  $\perp$ . From the definition of  $j^*$ , the pointers of the read locations are already  $\perp$ .
- (iv) For  $i = 1, \dots, N - 1$ , set  $b_{\ell_i}$  to 0 and  $\text{ptr}_{\ell_i}$  to  $\ell_{i+1}$ .
- (v) Set  $b_{\ell_N}$  to 0.

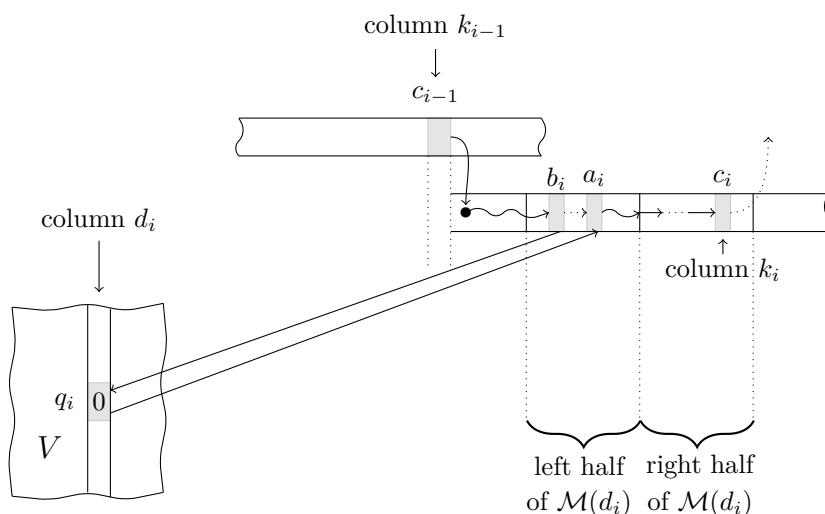
Clearly, the locations modified are not probed by  $\mathcal{Q}$ . Notice that the current input has the pointer chain  $\mathbf{p}_0 = (\ell^*, \ell_1, \dots, \ell_N)$  and the head  $\ell^*$  of the chain lies in the all-ones column  $j^*$ . Furthermore, all locations on the chain except  $\ell^*$  have 0 as their bit. We now show how to further modify our input and extend  $\mathbf{p}$  and visit the remaining columns through locations with 0's. The columns in  $W$  are already neatly arranged in pointer chains. The difficulty is in ensuring that we also visit the set of columns in  $V'$  that are completely read by  $\mathcal{Q}$ , for we are not allowed to make any modifications there. Let  $A_1$  denote these completely read columns in  $V'$ . Since  $\mathcal{Q}$  makes at most  $\sqrt{rs}/(\log s)^4$  queries, we have that  $|A_1| \leq s/(\sqrt{r}(\log s)^4)$ . Lemma 9 implies that there exists a matching  $\mathcal{M}$  in  $G_X$  that saturates  $A_1$ . Order the elements of  $A_1$  as  $d_1, \dots, d_L$  in such a way that for all  $i = 1, \dots, L - 1$ ,  $\mathcal{M}(d_i) < \mathcal{M}(d_{i+1})$  (where we order the bands in  $W$  from left to right), that is, the band that is matched with  $d_i$  lies to the left of the band that is matched to  $d_{i+1}$ .

We will now proceed as follows. For  $i = 1, \dots, L$ , we modify the input (at locations not read by  $\mathcal{Q}$ ) appropriately to induce a pointer chain  $\mathbf{p}_i$ . This pointer chain in addition to visiting a contiguous set of columns in  $W$ , will visit column  $d_i$ . By concatenating these pointer chains in order with the initial pointer chain  $\mathbf{p}_0$  we obtain the promised input  $X'$  for which  $\text{GPW}^{r \times s}(X') = 1$ .

To implement this strategy, recall that there is an edge in  $G_X$  between the column  $d_i$  and the band  $\mathcal{M}(d_i)$ . From the definition of  $G_X$ , it follows that there is a location  $q_i$  in  $d_i$  and a segment  $S_i$  in band  $\mathcal{M}(d_i)$  such that

- (s1)  $\text{ptr}_{q_i}$  leads to the left half of  $S_i$ ;
- (s2)  $\text{pred}(\text{ptr}_{q_i})$  is not probed by  $\mathcal{Q}$ ;
- (s3)  $\mathcal{Q}$  makes fewer than  $|S_i|/4$  queries in segment  $S_i$ .

First, let us describe how  $\mathbf{p}_1$  is constructed. Let  $a_1 = \text{ptr}_{q_1}$  and  $b_1 = \text{pred}(a_1)$  (by (s2)  $b_1$  is not probed by  $\mathcal{Q}$ ); let  $c_1$  be a location in the second half of  $S_1$  that is not probed by  $\mathcal{Q}$  and that is not the last location of  $S_1$  (by (s3) there is such a location). Now, we modify the input  $X$  by setting  $\text{ptr}_{b_1} = q_1$ . Then,  $\mathbf{p}_1$  is the pointer chain that starts at the head of the leftmost segment of  $W_1$  in the same row as  $S_1$  and continues until location  $c_1$ . That is, starting from its head, it follows the pointers of the input until  $b_1$ . Then it follows the



■ **Figure 4** Construction of pointer chain  $\mathbf{p}_i$ .

pointer leading out of  $b_1$  into  $q_1$ , thereby visiting column  $d_1$ . After that, it follows the pointer out of  $q_1$  and comes to  $a_1$ , and keeps following the pointers until  $c_1$ .

In general, suppose  $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_{i-1}$  have been constructed. Suppose  $\text{tail}(\mathbf{p}_{i-1})$  appears in column  $k_{i-1}$ . Then,  $\mathbf{p}_i$  is obtained as follows. Let  $a_i = \text{ptr}_{q_i}$  and  $b_i = \text{pred}(a_i)$ ; let  $c_i$  be a location in the second half of  $S_i$  that is not probed by  $\mathcal{Q}$  and that is not the last location of  $S_i$ . We modify the input by setting  $\text{ptr}_{b_i} = q_i$ . Then  $\mathbf{p}_i$  is the pointer chain with its head in the same row as  $a_i$  and in column  $k_{i-1} + 1$  (note that since  $\text{tail}(\mathbf{p}_{i-1})$  is not in last column of segment  $S_{i-1}$ , column  $k_{i-1} + 1$  is still in the same band as  $S_{i-1}$ ); the pointer chain  $\mathbf{p}_i$  terminates at location  $c_i$ . See Figure 4. Note that  $\mathbf{p}_i$  entirely keeps to one row (the row of  $S_i$ ), except for the diversion from  $b_i$  to  $q_i$ , when it visits column  $d_i$  and returns to  $a_i$ . When  $i = L$ , we let the pointer chain continue until the last column of  $W$ .

In obtaining the pointer chains  $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_L$ , we modified  $X$  at location  $b_1, b_2, \dots, b_L$ . Finally, we concatenate the pointer chains  $\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_L$ ; this requires us to modify  $X$  at locations  $\ell_N = \text{tail}(\mathbf{p}_0), c_1, c_2, \dots, c_{L-1}$ , which were left unprobed by  $\mathcal{Q}$ . The resulting input after these modifications is  $X'$ .

The pointer chain obtained by this concatenation visits each column other than  $j^*$  exactly once, and the bit at every location on it, other than its head, is 0. Hence,  $\text{GPW}^{r \times s}(X') = 1$ . ◀

## 5 Proof of the matching lemma

We will show that every subset  $R \subseteq V'$  of at most  $s/(\sqrt{r}(\log s)^4)$  columns has at least  $|R|$  neighbors in  $W'$ . Then, the claim will follow from Hall's theorem.

Observe that with high probability every column in  $V'$  has  $\Omega(\sqrt{r} \log s)$  pointers leaving it. We expect these pointers to be uniformly distributed among the at most  $\log s$  blocks in  $W$ ; in particular, we should expect that every column in  $V'$  sends  $\Omega(\sqrt{r})$  pointers into each block. We now formally establish this.

▶ **Claim 11.** *Let  $V_j$  be the  $j$ -th column of  $V'$  and  $W_{j'}$  the  $j'$ -th block of  $W$ ; then,*

$$\Pr[\forall j, j' : |\text{ptr}(V_j) \cap W_{j'}| \leq 400\sqrt{r}] = o(1).$$

## 16:10 The Zero-Error Randomized Query Complexity of the Pointer Function

**Proof.** Fix a location in  $\ell \in V_j$ . Let  $\chi_\ell$  be the indicator variable for the event  $\text{ptr}_\ell \in W_{j'}$ . Then, the number of pointers from  $V_j$  into  $W_{j'}$  is precisely  $\sum_{\ell \in V_j} \chi_\ell$ . Since

$$\Pr[\chi_\ell = 1] \geq \frac{500 \log s}{\sqrt{r}} \times \frac{1}{\log s} = \frac{500}{\sqrt{r}},$$

the expected number of pointers from column  $V_j$  into  $W_{j'}$  is at least  $500\sqrt{r}$ . Our claim follows from the Chernoff bound and the union bound (over choices of  $j$  and  $j'$ ). Here, we use the following version of the Chernoff bound (see Dubhashi and Panconesi [4], page 6): for the sum of  $r$  independent 0-1 random variables  $Z_\ell$ , each taking the value 1 with probability at least  $\alpha$ ,

$$\Pr\left[\sum_{\ell} X_\ell \leq (1 - \varepsilon)\alpha r\right] \leq \exp\left(-\frac{\varepsilon^2}{2}\alpha r\right).$$

Since we assume  $r = \Omega((\log s)^3)$ , in our application  $\alpha r \gg \sqrt{r} \geq \log s$ .  $\blacktriangleleft$

Suppose  $j$  is such that  $2^j \leq |R| < 2^{j+1}$ . Then, we will show that  $R$  has the required number of neighbors among the bands of the block  $W_j$ .

**► Claim 12.** *For a set  $R \subseteq V'$  and a block  $W_j$ , consider the set of bands of  $W_j$  into which at least  $2\sqrt{r}$  pointers from  $R$  fall, that is,  $B_j(R) := \{b \in W_j : |\text{ptr}(R) \cap b| \geq 2\sqrt{r}\}$ . Then, for  $j = 1, \dots, K$  and for all  $R$  such that  $2^j \leq |R| < 2^{j+1}$ , we have*

$$\Pr[|B_j(R)| \leq 2|R|] = o(1).$$

**Proof.** We will use the union bound over the choices of  $j$  and  $R$ . Fix the set  $R$ . We may, using Claim 11, condition on the event that there are at least  $400\sqrt{r}|R|$  pointers from  $R$  to  $W_j$ . Fix  $400\sqrt{r}|R|$  of these pointers. Now, the number of pointers that fall outside  $B_j(R)$  is at most  $20 \cdot 2^j \cdot 2\sqrt{r} \leq 100\sqrt{r}|R|$ . That is, if  $|B_j(R)| < 2|R|$ , then there is a set  $T$  of  $2|R|$  bands into which more than  $400\sqrt{r}|R| - 100\sqrt{r}|R| = 300\sqrt{r}|R|$  pointers from  $R$  fall. We will show that it is unlikely for such a set  $T$  to exist. For a fixed  $T$ , the probability of this event is at most

$$\binom{400|R|\sqrt{r}}{300|R|\sqrt{r}} \left(\frac{2|R|}{20 \cdot 2^j}\right)^{300\sqrt{r}|R|} \leq 2^{-100\sqrt{r}|R|}.$$

Using the union bound to account for all choices of  $R$  and the  $\binom{20 \cdot 2^j}{2|R|}$  choices of  $T$ , and using the fact that  $\sqrt{r} \gg \log s$ , we conclude that the probability that  $B_j(R)$  fails to be large enough is at most

$$\sum_{j=0}^{\log s - 3 \log \log s} \sum_{m=2^j}^{2^{j+1}-1} \binom{s/2}{m} \binom{20 \cdot 2^j}{2m} 2^{-100\sqrt{r}m} = o(1). \quad \blacktriangleleft$$

In order to show that with high probability the set  $R$  has the required number of neighbors, we will condition on the high probability event of Claim 12, that is,  $|B_j(R)| > 2|R|$ . Let  $\mathcal{B}$  be the set of such bands  $b$  that receive at least  $2\sqrt{r}$  pointers. For each  $b \in \mathcal{B}$ , let  $P(b)$  be a set of  $2\sqrt{r}$  locations in the columns in  $R$  whose pointers land in  $b$ . If in at least  $|R|$  of the  $2|R|$  such bands  $b$ , there is a pointer from  $P(b)$  satisfying the conditions (c1)–(c3), then we will have obtained the required expansion. Fix a pointer out of  $P(b)$  (which by definition of  $P(b)$  lands in band  $b$ ), and consider the following events.

$\mathcal{E}_1$ : The pointer leads to the same segment as a previous pointer (assume the locations in  $P(b)$  are totally ordered in some way).

$\mathcal{E}_2$ : The pointer leads to the first entry of the pointer chain in its segment (so, that location has no predecessor).

$\mathcal{E}_3$ : At least  $w_j/8$  entries of the segment that the pointer lands in are probed by  $\mathcal{Q}$ .

$\mathcal{E}_4$ : The predecessor of the location where the pointer lands is probed by  $\mathcal{Q}$ .

Consider the pointers that emanate from  $P(b)$  and land in some band  $b \in \mathcal{B}$ . Let  $n_1$  be the number of those pointers for whom  $\mathcal{E}_1$  holds; let  $n_2$  be the number of those pointers for whom  $\mathcal{E}_2$  holds; let  $n_3$  be the number of those pointers for whom  $\mathcal{E}_3$  holds but  $\mathcal{E}_1$  does not hold; let  $n_4$  be the number of those pointers for whom  $\mathcal{E}_4$  holds but  $\mathcal{E}_1, \mathcal{E}_2$  and  $\mathcal{E}_3$  do not hold.

If the claim of our lemma does not hold, then it must be that in at least  $|R|$  of the  $2|R|$  bands of  $\mathcal{B}$ , all pointers that fall there fail to satisfy at least one of the conditions (c1)–(c3); that is, one of  $\mathcal{E}_1, \dots, \mathcal{E}_4$  holds for all  $2\sqrt{r}$  of them. This implies that

$$n_1 + n_2 + n_3 + n_4 \geq 2\sqrt{r}|R|. \quad (1)$$

To prove our claim, we will show that with high probability each  $n_i$  on the left is less than  $\sqrt{r}|R|/2$ . In the following, we fix a set  $R$  and separately estimate the probability that one of the quantities on the left is large. To establish the claim for all  $R$ , we will use the union bound over  $R$ . In the proof, we use the following version of the Chernoff-Hoeffding bound, which can be found in Dubhashi and Panconesi ([4], page 7).

► **Lemma 13** (Chernoff-Hoeffding bound). *Let  $X := \sum_{i \in [n]} X_i$  where  $X_i, i \in [n]$  are independently distributed in  $[0, 1]$ . Let  $t > 2e\mathbb{E}[X]$ . Then*

$$\mathbb{P}[X > t] \leq 2^{-t}.$$

► **Claim 14.**  $\Pr[n_1 \geq \sqrt{r}|R|/2] \leq 2^{-r|R|/2}$ .

**Proof.** The probability that a pointer from  $P(b)$  falls on a segment of a previous pointer is at most  $2\sqrt{r}/r$ . Thus, the expected value of  $n_1$  is at most  $8|R|$ . We may invoke lemma 13 and conclude that

$$\Pr[n_1 \geq \sqrt{r}|R|/2] \leq 2^{-\sqrt{r}|R|/2}. \quad \blacktriangleleft$$

Recall that the number of blocks is  $K = \log s - 3 \log \log s$  and the width of each band in the  $j$ -th block is  $w_j = s/(20 \cdot 2^j \cdot 2K)$ .

► **Claim 15.**  $\Pr[n_2 \geq \sqrt{r}|R|/2] \leq 2^{-\sqrt{r}|R|/2}$ .

**Proof.** A pointer falls on head of random pointer chain in a segment with probability at most  $2/w_j$ . Thus,

$$\mathbb{E}[n_2] \leq \left(\frac{2}{w_j}\right) 4\sqrt{r}|R| \leq \frac{320|R|\sqrt{r}}{(\log s)^2}.$$

Again, our claim follows by a routine application of Lemma 13. ◀

► **Claim 16.**  $\Pr[n_3 \geq \sqrt{r}|R|/2] = 0$ .

**Proof.** If  $n_3 \geq \sqrt{r}|R|/2$ , then the total number of locations read by  $\mathcal{Q}$  is at least

$$n_3 \frac{w_j}{8} \geq \left(\frac{\sqrt{r}|R|}{2}\right) \cdot \frac{w_j}{8} \geq \left(\frac{\sqrt{r}2^j}{2}\right) \left(\frac{s}{8 \cdot 20 \cdot 2^j \log s}\right) \gg \frac{\sqrt{r}s}{320 \log s}.$$

This contradicts our assumption that  $\mathcal{Q}$  makes at most  $\sqrt{r}s/(\log s)^4$  queries. ◀

► **Claim 17.**  $\Pr[n_4 \geq \sqrt{r}|R|/2] \leq 2^{-r|R|/2}$ .

**Proof.** Let us first sketch informally why we do not expect  $n_4$  to be large. Recall that in our random input we place a random pointer chain in the left half of each segment. Once a pointer has landed at a location in this segment, its predecessor is equally likely to be any of the other locations in the segment. So the first probe into that segment has probability about one in  $w_j/2 - 1$  of landing on the predecessor, the second probe has probability about one in  $w_j/2 - 2$  of landing on the predecessor, and so on. Since we assume  $\mathcal{E}_3$   $\mathcal{Q}$  makes at most  $w_j/8$  probes in this segment. So, conditioned on the previous probes being unsuccessful, there are still  $w_j/2 - w_j/8 - 1$  possibilities for the location of the predecessor; so the probability of the probe landing on the predecessor is at most  $1/(w_j/2 - w_j/8 - 1)$ . This implies that in order for  $n_4$  to be at least  $\sqrt{r}|R|/2$  the query algorithm  $\mathcal{Q}$  must make  $\Omega(w_j\sqrt{r}|R|/2)$  queries; but this is more than the number of probes  $\mathcal{Q}$  is permitted.

In order to formalize this intuition, fix (condition on) a choice of pointers from  $V$ . Let us assume that the algorithm makes  $t$  probes. For  $i = 1, 2, \dots, t$ , define indicator random variables  $\chi_i$  as follows:  $\chi_i = 1$  iff the following conditions hold.

- Suppose the  $i$ -th probe is made to a segment  $p$  in band  $b \in \mathcal{B}$ . Let  $\ell$  be the location where the first pointer (among the pointers from  $P(b)$  to  $p$ ) lands. Then, the  $i$ -th probe of  $\mathcal{Q}$  is made to the predecessor of  $\ell$  in the random pointer chain in  $b$ .
- Fewer than  $w_j/8$  of the previous probes were made to this segment.

Observe that if more than one pointer land on  $p$ , then except for the first amongst them (according to the ordering on the locations in  $P(b)$ ), event  $\mathcal{E}_2$  does not hold for the remaining pointers, and hence by definition event  $\mathcal{E}_4$  does not hold either.

Define  $Z = \sum_{i=1}^t \chi_i$ . Note that  $Z$  is an upper bound on  $n_4$ , and we wish to estimate the probability that  $Z \geq \sqrt{r}|R|/2$ . The key observation is that for every choice  $\sigma$  of  $\chi_1, \chi_2, \dots, \chi_{i-1}$ , we have

$$\Pr[\chi_i = 1 \mid \chi_1, \chi_2, \dots, \chi_{i-1} = \sigma] \leq \frac{1}{3w_j/8 - 1} \leq \frac{4}{w_j}. \quad (2)$$

Thus,

$$\mathbb{E}[Z] \leq \left(\frac{4}{w_j}\right) t \leq \left(\frac{4}{w_j}\right) (\log s)^{-4} \sqrt{r} s \leq (\log s)^{-2} \sqrt{r}|R|.$$

The variables  $\chi_i$  are not independent, but it follows from (2) that Lemma 13 is still applicable in this setting. We conclude that

$$\Pr[Z \geq \sqrt{r}|R|/2] \leq 2^{-\sqrt{r}|R|/2}.$$

Since, the above bound holds for each choice of pointers from  $V$ , it holds in general. ◀

Finally, to establish the required expansion for all sets  $R$ , we use the union bound over all  $R$ . The probability that some set  $R$  has fewer than  $|R|$  neighbors is at most

$$4 \sum_{k=1}^{s/(\sqrt{r}(\log s)^4)} \binom{s/2}{k} 2^{-\sqrt{r}k/2} \leq \sum_{k \geq 1} s^k 2^{-\sqrt{r}k/2} \leq \sum_{k \geq 1} s^{-k} = o(1),$$

where we used our assumption that  $r \gg (\log s)^2$ . This completes the proof of the matching lemma.

**Acknowledgment.** We thank Sagnik Mukhopadhyay for useful discussions.



---

**References**

---

- 1 Scott Aaronson. A query complexity breakthrough. Shtetl-Optimized. URL: <http://www.scottaaronson.com/blog/?p=2325>.
- 2 Andris Ambainis, Kaspars Balodis, Aleksandrs Belovs, Troy Lee, Miklos Santha, and Juris Smotrovs. Separations in query complexity based on pointer functions. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 800–813, 2016.
- 3 Manuel Blum and Russell Impagliazzo. Generic oracles and oracle classes (extended abstract). In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 118–126, 1987.
- 4 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.
- 5 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1077–1088, 2015.
- 6 Juris Hartmanis and Lane A. Hemachandra. One-way functions, robustness, and the non-isomorphism of np-complete sets. In *Proceedings of the Second Annual Conference on Structure in Complexity Theory, Cornell University, Ithaca, New York, USA, June 16-19, 1987*, 1987.
- 7 Sagnik Mukhopadhyay and Swagato Sanyal. Towards better separation between deterministic and randomized query complexity. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, pages 206–220, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.206.
- 8 Noam Nisan. CREW prams and decision trees. *SIAM J. Comput.*, 20(6):999–1007, 1991.
- 9 Michael E. Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 29–38, 1986.
- 10 Marc Snir. Lower bounds on probabilistic linear decision trees. *Theor. Comput. Sci.*, 38:69–82, 1985.
- 11 Gábor Tardos. Query complexity, or why is it difficult to separate  $NP^A \cap coNP^A$  from  $P^A$  by random oracles  $A$ ? *Combinatorica*, 9(4):385–392, 1989.



# Robust Multiplication-Based Tests for Reed-Muller Codes

Prahladh Harsha<sup>1</sup> and Srikanth Srinivasan<sup>2</sup>

**1** TIFR, Mumbai, India  
prahladh@tifr.res.in

**2** Dept. of Mathematics, IIT Bombay, Mumbai, India  
srikanth@math.iitb.ac.in

---

## Abstract

We consider the following multiplication-based tests to check if a given function  $f : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$  is the evaluation of a degree- $d$  polynomial over  $\mathbb{F}_q$  for  $q$  prime.

- $\text{Test}_{e,k}$ : Pick  $P_1, \dots, P_k$  independent random degree- $e$  polynomials and accept iff the function  $fP_1 \cdots P_k$  is the evaluation of a degree- $(d + ek)$  polynomial.

We prove the robust soundness of the above tests for large values of  $e$ , answering a question of Dinur and Guruswami (FOCS 2013). Previous soundness analyses of these tests were known only for the case when either  $e = 1$  or  $k = 1$ . Even for the case  $k = 1$  and  $e > 1$ , earlier soundness analyses were not robust.

We also analyze a derandomized version of this test, where (for example) the polynomials  $P_1, \dots, P_k$  can be the *same* random polynomial  $P$ . This generalizes a result of Guruswami *et al.* (STOC 2014).

One of the key ingredients that go into the proof of this robust soundness is an extension of the standard Schwartz-Zippel lemma over general finite fields  $\mathbb{F}_q$ , which may be of independent interest.

**1998 ACM Subject Classification** F.2.1 Numerical Algorithms and Problems

**Keywords and phrases** Polynomials over finite fields, Schwartz-Zippel lemma, Low degree testing, Low degree long code

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.17

## 1 Introduction

We consider the problem of testing if a function  $f : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$  is close to a degree- $d$  multivariate polynomial (over  $\mathbb{F}_q$ , the finite field of  $q$  elements). This problem, in its local testing version, was first studied by Alon, Kaufman, Krivilevich, Litsyn and Ron [1], who proposed and analyzed a natural  $2^{d+1}$ -query test for this problem for the case when  $q = 2$ . Subsequent to this work, improved analyses and generalizations to larger fields were discovered [3, 6]. These tests and their analyses led to several applications, especially in hardness of approximation, which in turn spurred other Reed-Muller testing results (which were not necessarily local tests) [4, 5]. In this work, we give a robust version of one of these latter multiplication based tests due to Dinur and Guruswami [4]. Below we describe this variation of the testing problem, its context, and our results.

### 1.1 Local Reed-Muller tests

Given a field  $\mathbb{F}_q$  of size  $q$ , let  $\mathcal{F}_q(n) := \{f \mid f : \mathbb{F}_q^n \rightarrow \mathbb{F}_q\}$ . The Reed-Muller code  $\mathcal{P}_q(n, d)$ , parametrized by two parameters  $n$  and  $d$ , is the subset of  $\mathcal{F}_q(n)$  that corresponds to those



© Prahladh Harsha and Srikanth Srinivasan;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 17; pp. 17:1–17:14



Leibniz International Proceedings in Informatics

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

functions which are evaluations of polynomials of degree at most  $d$ . If  $n$ ,  $d$  and  $q$  are clear from context,  $r := (q - 1)n - d$ .

The proximity of two functions  $f, g \in \mathcal{F}_q(n)$  is measured by the Hamming distance. Specifically, we let  $\Delta(f, g)$  denote the absolute Hamming distance between  $f$  and  $g$ , i.e.,  $\Delta(f, g) := \#\{x \in \mathbb{F}_q^n \mid f(x) \neq g(x)\}$ . For a family of functions  $\mathcal{G} \subseteq \mathcal{F}_q(n)$ , we let  $\Delta(f, \mathcal{G}) := \min\{\Delta(f, g) \mid g \in \mathcal{G}\}$ . We say that  $f$  is  $\Delta$ -close to  $\mathcal{G}$  if  $\Delta(f, \mathcal{G}) \leq \Delta$  and  $\Delta$ -far otherwise.

The following natural local test to check membership of a function  $f$  in  $\mathcal{P}_2(n, d)$  was proposed by Alon *et al.* [1] for the case when  $q = 2$ .

- AKKLR Test: Input  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ 
  - Pick a random  $d + 1$ -dimensional affine space  $A$ .
  - Accept iff  $f|_A \in \mathcal{P}_2(d + 1, d)$ .

Here,  $f|_A$  refers to the restriction of the function  $f$  to the affine space  $A$ . Bhattacharyya *et al.* [3] showed the following optimal analysis of this test.

► **Theorem 1.1** ([1, 3]). *There exists an absolute constant  $\alpha > 0$  such that the following holds. If  $f \in \mathcal{F}_2(n)$  is  $\Delta$ -far from  $\mathcal{P}_2(n, d)$  for  $\Delta \in \mathbb{N}$ , then*

$$\Pr_A[f|_A \notin \mathcal{P}_2(d + 1, d)] \geq \min\{\Delta/2^r, \alpha\}.$$

Subsequent to this result, Haramaty, Shpilka and Sudan [6] extended this result to all constant sized fields  $\mathbb{F}_q$ . These optimal analyses then led to the discovery of the so-called “short code” (aka the low degree long code) due to Barak *et al.* [2] which has played an important role in several improved hardness of approximation results [4, 5, 9, 10, 7].

## 1.2 Multiplication based tests

We now consider the following type of multiplication-based tests to check membership in  $\mathcal{P}_q(n, d)$ , parametrized by two numbers  $e, k \in \mathbb{N}$ .

- $\text{Test}_{e,k}$ : Input  $f : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$ 
  - Pick  $P_1, \dots, P_k \in_R \mathcal{P}_q(n, e)$ .
  - Accept iff  $fP_1 \cdots P_k \in \mathcal{P}_q(n, d + ek)$ .

This tests computes the point-wise product of  $f$  with  $k$  random degree- $e$  polynomials  $P_1, \dots, P_k$  respectively and checks that the resulting product function  $fP_1 \cdots P_k$  is the evaluation of a degree- $(d + ek)$  polynomial. Unlike the previous test, this test is not necessarily a local test.

The key lemma due to Bhattacharyya *et al.* [3] that led to the optimal analysis in Theorem 1.1 is the following robust analysis of  $\text{Test}_{1,1}$ .

► **Lemma 1.2** ([3]). *Let  $f \in \mathcal{F}_2(n)$  be  $\Delta$ -far from  $\mathcal{P}_2(n, d)$  for  $\Delta = 2^r/100$ . For randomly picked  $\ell \in \mathcal{P}_2(n, 1)$ , we have*

$$\Pr_\ell[\Delta(f \cdot \ell, \mathcal{P}_2(n, d + 1)) < \beta\Delta] = O\left(\frac{1}{2^r}\right),$$

for some absolute constant  $\beta > 0$ .

Observe that the AKKLR test is equivalent to  $\text{Test}_{1,r-1}$  for  $r = n - d$ . This observation coupled with a simple inductive argument using the above lemma implies Theorem 1.1.

Motivated by questions related to hardness of coloring hypergraphs, Dinur and Guruswami studied the  $\text{Test}_{e,1}$  for  $e = r/4$  and proved the following result.

► **Lemma 1.3** ([4]). *Let  $f \in \mathcal{F}_2(n)$  be  $\Delta$ -far from  $\mathcal{P}_2(n, d)$  for  $\Delta = 2^r/100$  and let  $e = (n - d)/4$ . For randomly picked  $P \in \mathcal{P}_2(n, e)$ , we have*

$$\Pr_P[f \cdot P \in \mathcal{P}_2(n, d + e)] \leq \frac{1}{2^{2^{\Omega(e)}}}.$$

Note that the  $\text{Test}_{e,1}$  is not a local test (as is the case with multiplication based tests of the form  $\text{Test}_{e,k}$ ). Furthermore, the above lemma does not give a robust analysis unlike Lemma 1.2. More precisely, the lemma only bounds the probability that the product function  $f \cdot P$  is in  $\mathcal{P}_2(n, d + e)$ , but does not say anything about the probability of  $f \cdot P$  being close to  $\mathcal{P}_2(n, d + e)$  as in Lemma 1.2. Despite this, this lemma has had several applications, especially towards proving improved inapproximability results for hypergraph colouring [4, 5, 9, 10, 7].

### 1.3 Our results

Our work is motivated by the question raised at the end of the previous section: can the analysis of the Dinur-Guruswami Lemma be strengthened to yield a robust version of Lemma 1.3? Such a robust version, besides being interesting of its own right, would yield a soundness analysis of the  $\text{Test}_{e,k}$  for  $k > 1$  (wherein the input function  $f$  is multiplied by  $k$  degree- $e$  polynomials). This is similar to how Lemma 1.2 was instrumental in proving Theorem 1.1.

We begin by first showing this latter result (ie., the soundness analysis of the  $\text{Test}_{e,k}$ ).

► **Theorem 1.4.** *Let  $q, k \in \mathbb{N}$  be constants with  $q$  prime and  $\varepsilon, \delta \in (0, 1)$  be arbitrary constants. Let  $n, d, r, \Delta, e \in \mathbb{N}$  be such that  $r = q(n - 1) - d$ ,  $q^{\varepsilon r} \leq \Delta \leq q^{r/4(q-1)-2}$ , and  $\delta r \leq e \leq r/4k$ . Then, given any  $f \in \mathcal{F}_q(n)$  that is  $\Delta$ -far from  $\mathcal{P}_q(n, d)$  and for  $P_1, \dots, P_k$  chosen independently and uniformly at random from  $\mathcal{P}_q(n, e)$ , we have*

$$\Pr_{P_1, \dots, P_k} [f P_1 P_2 \cdots P_k \in \mathcal{P}_q(n, d + ek)] \leq \frac{1}{q^{q^{\Omega(r)}}}$$

where the  $\Omega(\cdot)$  above hides a constant depending on  $k, q, \delta, \varepsilon$ .

Surprisingly, we show that the above theorem (which we had observed is a simple consequence of a robust version of Lemma 1.3), can in fact, be used to prove the following robust version of Lemma 1.3, answering an open question of Dinur and Guruswami [4].

► **Lemma 1.5.** *Let  $q \in \mathbb{N}$  be a constant with  $q$  prime and  $\varepsilon, \delta \in (0, 1)$  be arbitrary constants. Let  $n, d, r, \Delta, \Delta', e \in \mathbb{N}$  be such that  $r = q(n - 1) - d$ ,  $q^{\varepsilon r} \leq \Delta \leq q^{r/4(q-1)-2}$ , and  $\delta r \leq e \leq r/4k$  where  $k := 1 + \lceil \log_{q/(q-1)}(2\Delta') \rceil$ . Then, given any  $f \in \mathcal{F}_q(n)$  that is  $\Delta$ -far from  $\mathcal{P}_q(n, d)$  and for  $P$  chosen uniformly at random from  $\mathcal{P}_q(n, e)$ , we have*

$$\Pr_P[\Delta(f \cdot P, \mathcal{P}_q(n, d + e)) < \Delta'] \leq \frac{2}{q^{q^{\Omega(r)}}}$$

where the  $\Omega(\cdot)$  above hides a constant depending on  $q, \delta, \varepsilon$ .

Equipped with such multiplication-based tests, we can ask if one can prove the soundness analysis of other related multiplication-based tests. For instance, consider the following test which tests correlation of the function  $f$  with the square of a random degree- $e$  polynomial.

- **Corr-Square<sub>e</sub>:** Input  $f : \mathbb{F}_3^n \rightarrow \mathbb{F}_3$ 
  - Pick  $P \in_R \mathcal{P}_3(n, e)$ .
  - Accept iff  $f \cdot P^2 \in \mathcal{P}_3(n, d + 2e)$ .

This test was used by Guruswami *et al.* [5] to prove the hardness of approximately coloring 3-colorable 3-uniform hypergraphs. However, their analysis was restricted to the squares of random polynomials. Our next result shows that this can be extended to any low-degree polynomial of random polynomials. More precisely, let  $h \in \mathcal{P}_q(n, k)$  be a polynomial of degree exactly  $k$  for some  $k < q$ . Consider the following test.

- **Corr- $h_e$** : Input  $f : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$ 
  - Pick  $P \in_R \mathcal{P}_q(n, e)$ .
  - Accept iff  $f \cdot h(P) \in \mathcal{P}_q(n, d + ek)$ .

We show that an easy corollary of Theorem 1.4 proves the following soundness claim about the test **Corr- $h$** .

► **Corollary 1.6.** *Let  $q, k \in \mathbb{N}$  be constants with  $q$  prime,  $k < q^\dagger$ , and let  $\varepsilon, \delta \in (0, 1)$  be arbitrary constants. Let  $n, d, r, \Delta, e \in \mathbb{N}$  be such that  $r = q(n - 1) - d$ ,  $q^{\varepsilon r} \leq \Delta \leq q^{r/4(q-1)-2}$ , and  $\delta r \leq e \leq r/4k$ . Let  $h \in \mathcal{P}_q(1, k)$  be a univariate polynomial of degree exactly  $k$ . Then, given any  $f \in \mathcal{F}_q(n)$  that is  $\Delta$ -far from  $\mathcal{P}_q(n, d)$  and for  $P$  chosen uniformly at random from  $\mathcal{P}_q(n, e)$ , we have*

$$\Pr_P[f \cdot h(P) \in \mathcal{P}_q(n, d + ek)] \leq \frac{1}{q^{q^{\Omega(r)}/2k}}$$

where the  $\Omega(\cdot)$  above hides a constant depending on  $k, q, \delta, \varepsilon$ .

### A generalization of the Schwartz-Zippel lemma over $\mathbb{F}_q$ .

A special case of Theorem 1.4 is already quite interesting. This case corresponds to when the function  $f$  is a polynomial of degree  $d'$  slightly larger than  $d$ . (It is quite easy to see by the Schwartz-Zippel lemma over  $\mathbb{F}_q$  – which guarantees that a non-zero polynomial of low degree is non-zero at many points – that this  $f$  is far from  $\mathcal{P}_q(n, d)$ .) In this case, we would expect, when we multiply  $f$  with  $k$  random polynomials  $P_1, \dots, P_k \in \mathcal{P}_q(n, e)$ , that the product  $fP_1 \cdots P_k$  is a polynomial of degree  $d' + ek$  with high probability.

We are able to prove a tight version of this statement (Lemma 3.3). For every degree  $d'$ , we find a polynomial  $f$  of degree  $d'$  that maximizes the probability that  $fP_1 \cdots P_k$  has degree  $< d' + s$  for any parameter  $s \leq e$ . This polynomial turns out to be the same polynomial for which the Schwartz-Zippel lemma over  $\mathbb{F}_q$  is tight. This is not a coincidence: it turns out that our lemma, viewed suitably, is a generalization of the Schwartz-Zippel lemma over  $\mathbb{F}_q$  (see Section 3.1 and the full version for more details).

Given the utility of the Schwartz-Zippel lemma in Theoretical Computer Science, we feel that this statement will be of independent interest.

## 1.4 Proof ideas

The basic outline of the proof of Theorem 1.4 is similar to the proof of Lemma 1.3 from the work of Dinur and Guruswami [4] which corresponds to Theorem 1.4 in the case that  $q = 2$  and  $k = 1$ . The argument is essentially an induction on the parameters  $e, r = n - d$ , and  $\Delta$ . We describe this argument in some detail so that we can highlight the variations in our work.

---

<sup>†</sup> The assumption  $k < q$  is necessary here is since otherwise  $h(P)$  could be  $P^q - P$ , which is always 0.

As long as  $r$  is a sufficiently large constant, Lemma 1.2 can be used to show that for any  $f \in \mathcal{F}_2(n)$  that is  $\Delta$ -far from  $\mathcal{P}_2(n, d)$ , there is a variable  $X$  such that for each  $\alpha \in \{0, 1\} = \mathbb{F}_2$ , the restricted function  $f|_{X=\alpha}$  is  $\Delta' = \Omega(\Delta)$ -far from  $\mathcal{P}_2(n - 1, d)$ .\*

Now, to argue by induction, we write

$$f = Xg + h \text{ and } P_1 = XQ_1 + R_1 \tag{1}$$

where  $g, h, Q_1, R_1$  depend on  $n - 1$  variables,  $Q_1$  is a random polynomial of degree  $\leq e - 1$  and  $R_1$  is a random polynomial of degree  $\leq e$ . Using the fact that  $X^2 = X$  over  $\mathbb{F}_2$ , we get  $fP_1 = X((g + h)Q_1 + gR_1) + hR_1$ .

Since  $f|_{X=\alpha}$  is  $\Delta'$ -far from  $\mathcal{P}_2(n - 1, d)$ , we see that both  $h$  and  $g + h$  are  $\Delta'$ -far from  $\mathcal{P}_2(n - 1, d)$ . To apply induction, we note that  $fP_1 \in \mathcal{P}_2(n, d + e)$  iff  $hR_1 \in \mathcal{P}_2(n - 1, d + e)$  – call this event  $\mathcal{E}_1$  – and  $(g + h)Q_1 + hR_1 \in \mathcal{P}_2(n - 1, d + e - 1)$ , which we call  $\mathcal{E}_2$ . We bound the overall probability by  $\Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid R_1]$  (note that  $\mathcal{E}_1$  depends only on  $R_1$ ).

We first observe that  $\Pr[\mathcal{E}_1]$  can be immediately bounded using the induction hypothesis since  $h$  is  $\Delta'$ -far from  $\mathcal{P}_q(n - 1, d + e)$  and  $R_1$  is uniform over  $\mathcal{P}_q(n - 1, e)$ . The second term  $\Pr[\mathcal{E}_2 \mid R_1]$  can also be bounded by the induction hypothesis with an additional argument. We argue that (for any fixed  $R_1$ ) the probability that  $(g + h)Q_1 + gR_1 \in \mathcal{P}_2(n - 1, d + e - 1)$  is bounded by the probability that  $(g + h)Q_1 \in \mathcal{P}_2(n - 1, d + e - 1)$ : this follows from the fact that the number of solutions to any system of linear equations is bounded by the number of solutions of the corresponding homogeneous system (obtained by setting the constant term in each equation to 0). Hence, it suffices to bound the probability that  $(g + h)Q_1 \in \mathcal{P}_2(n - 1, d + e - 1)$ , which can be bounded by the induction hypothesis since  $(g + h)$  is  $\Delta'$ -far from  $\mathcal{P}_2(n - 1, d)$  and  $Q_1$  is uniform over  $\mathcal{P}_2(n - 1, e - 1)$  and we are done.

Though our proofs follow the above template, we need to deviate from the proof above in some important ways which we elaborate below.

The first is the decomposition of  $f$  and  $P_1$  from (1) obtained above, which yields two events  $\mathcal{E}_1$  and  $\mathcal{E}_2$ , the first of which depends only on  $R_1$  and the second on both  $Q_1$  and  $R_1$ . For  $q > 2$ , the standard monomial decomposition of polynomials does not yield such a nice “upper triangular” sequence of events. So we work with a different polynomial basis to achieve this. This choice of basis is closely related to the polynomials for which the Schwartz-Zippel lemma over  $\mathbb{F}_q$  is tight. While such a basis was used in the special case of  $q = 3$  in the work of Guruswami *et al.* [5] (co-authored by the authors of this work), it was done in a somewhat ad-hoc way. Here, we give, what is in our opinion, a more transparent construction that additionally works for all  $q$ . For lack of space, this part of the proof has been omitted from this extended abstract.

Further modifications to the Dinur-Guruswami argument are required to handle  $k > 1$ . We illustrate this with the example of  $q = 2$  and  $k = 2$ . Decomposing as in the Dinur-Guruswami argument above, we obtain  $f = Xg + h$ ,  $P_1 = XQ_1 + R_1$ , and  $P_2 = XQ_2 + R_2$ . Multiplying out, we get

$$fP_1P_2 = X \underbrace{(Q_1Q_2(g + h) + (g + h)(Q_1R_2 + Q_2R_1) + gR_1R_2)}_{:=Q} + hR_1R_2 .$$

Bounding the probability that  $fP_1P_2 \in \mathcal{P}_2(n, d + 2e)$  thus reduces to bounding the probability of event that  $hR_1R_2 \in \mathcal{P}_2(n - 1, d + 2e) - \mathcal{E}_1$  depending only on  $R_1$  and  $R_2$  – and

---

\* Actually, Lemma 1.2 implies the existence of a linear function with this property and not a variable. But after a linear transformation of the underlying space, we may assume that it is a variable.

then the probability that  $Q \in \mathcal{P}_2(n-1, d+2e-1)$  – denoted  $\mathcal{E}_2$  – given any fixed  $R_1$  and  $R_2$ . The former probability can be bounded using the induction hypothesis straightforwardly.

By a reasoning similar to the  $k=1$  case, we can reduce bounding  $\Pr[\mathcal{E}_2 \mid R_1, R_2]$  to the probability that  $Q_1 Q_2(g+h) \in \mathcal{P}_2(n-1, d+2e-1)$ . However, now we face a problem. Note that we have  $g+h = f|_{X=1}$  is  $\Delta'$ -far from  $\mathcal{P}_2(n-1, d)$  and  $Q_1, Q_2 \in \mathcal{P}_2(n-1, e-1)$ . Thus, the induction hypothesis only allows us to upper bound the probability that  $Q_1 Q_2(g+h) \in \mathcal{P}_2(n-1, d+2e-2)$  which is not quite the event that we want to analyze. Indeed, if  $f$  is a polynomial of degree exactly  $d+1$ , then the polynomial  $Q_1 Q_2(g+h) \in \mathcal{P}_2(n, d+2e-1)$  with probability 1. A similar problem occurs even if  $f$  is a polynomial of degree  $d'$  slightly larger than  $d$  or more generally, when  $f$  is *close* to some polynomial of degree  $d'$ .

This naturally forces us to break the analysis into two cases. In the first case, we assume not just that  $f$  is far from  $\mathcal{P}_2(n, d)$  but from  $\mathcal{P}_2(n, d')$  but for some  $d'$  a suitable parameter larger than  $d$ . In this case, we can modify the proof of Dinur and Guruswami to bound the probability that  $f P_1 P_2 \in \mathcal{P}_2(n, d+2e)$  as claimed in Theorem 1.4. In the complementary case when  $f$  is close to some polynomial  $F \in \mathcal{P}_2(n, d')$ , we can essentially assume that  $f$  is a polynomial of degree  $d'$ . In this case, we can use the extension of Schwartz-Zippel lemma referred to above to show that with high probability  $f P_1 P_2$  is in fact a polynomial of degree exactly  $d'+2e$  and is hence not of degree  $d+2e < d'+2e$ .

## 1.5 Organization

We begin with some notation and definitions in Section 2. We prove the extension of the Schwartz-Zippel lemma (Lemma 3.3) in Section 3 and then Theorem 1.4 in Section 4. Finally, we give two applications of Theorem 1.4 in Section 5: one to proving a robust version of the above test (thus resolving a question of Dinur and Guruswami [4]) and the other to proving Corollary 1.6. For lack of space, many proofs have been omitted. The reader is referred to the full version of this paper for details.

## 2 Preliminaries

For a prime power  $q$ , let  $\mathbb{F}_q$  denote the finite field of size  $q$ . We use  $\mathbb{F}_q[X_1, \dots, X_n]$  to denote the standard polynomial ring over variables  $X_1, \dots, X_n$  and  $\mathcal{P}_q(n)$  to denote the ring  $\mathbb{F}_q[X_1, \dots, X_n] / \langle X_1^q - X_1, \dots, X_n^q - X_n \rangle$ .

We can think of the elements of  $\mathcal{P}_q(n)$  as elements of  $\mathbb{F}_q[X_1, \dots, X_n]$  of individual degree at most  $q-1$  in a natural way. Given  $P, Q \in \mathcal{P}_q(n)$ , we use  $P \cdot Q$  or  $PQ$  to denote their product in  $\mathcal{P}_q(n)$ . We use  $P * Q$  to denote their product in  $\mathbb{F}_q[X_1, \dots, X_n]$ .

Given a set  $S \subseteq \mathbb{F}_q^n$  and an  $f \in \mathcal{P}_q(n)$ , we use  $f|_S$  to denote the restricted function on the set  $S$ . Typically,  $S$  will be specified by a polynomial equation. One special case is the case when  $S$  is a hyperplane: i.e., there is a non-zero homogeneous degree-1 polynomial  $\ell(X) \in \mathcal{P}_q(n)$  and an  $\alpha \in \mathbb{F}_q$  such that  $S = \{x \mid \ell(x) = \alpha\}$ . In this case, it is natural to think of  $f|_{\ell(X)=\alpha} = f|_S$  as an element of  $\mathcal{P}_q(n-1)$  by applying a linear transformation that transforms  $\ell(X)$  into one of the variables – say  $X_n$  – and then setting  $X_n = \alpha$ .

For  $d \geq 0$ , we use  $\mathcal{P}_q(n, d)$  to denote the polynomials in  $\mathcal{P}_q(n)$  of degree at most  $d$ .

The following are standard facts about the ring  $\mathcal{P}_q(n)$  and the space of functions mapping  $\mathbb{F}_q^n$  to  $\mathbb{F}_q$ .

### ► Fact 2.1.

1. Consider the ring of functions mapping  $\mathbb{F}_q^n$  to  $\mathbb{F}_q$  with addition and multiplication defined pointwise. This ring is isomorphic to  $\mathcal{P}_q(n)$  under the natural isomorphism that maps each polynomial  $P \in \mathcal{P}_q(n)$  to the function (mapping  $\mathbb{F}_q^n$  to  $\mathbb{F}_q$ ) represented by this polynomial.



2. In particular, each function  $f : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$  can be represented uniquely as a polynomial from  $\mathcal{P}_q(n)$ . As a further special case, any non-zero polynomial from  $\mathcal{P}_q(n)$  represents a non-zero function  $f : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$ .
3. (Schwartz-Zippel lemma over  $\mathbb{F}_q$  [8]) Any non-zero polynomial from  $\mathcal{P}_q(n, d)$  is non-zero on at least  $q^{n-a-1}(q-b)$  points from  $\mathbb{F}_q^n$  where  $d = a(q-1) + b$  and  $0 \leq b < q-1$ .
4. In particular, if  $f, g \in \mathcal{P}_q(n, d)$  differ from each other in at most  $\Delta < q^{n-a-1}(q-b)$  places, then  $f = g$ .
5. (A probabilistic version of the Schwartz-Zippel lemma [6]) It follows from the above that given a non-zero polynomial  $g \in \mathcal{P}_q(n, d)$ , then  $g(x) \neq 0$  at a uniformly random point of  $\mathbb{F}_q^n$  with probability at least  $q^{-d/(q-1)}$ . Similarly, if  $f, g \in \mathcal{P}_q(n, d)$  are distinct, then for uniformly random  $x \in \mathbb{F}_q^n$ , the probability that  $f(x) \neq g(x)$  is at least  $q^{-d/(q-1)}$ .

From now on, we will use without additional comment the fact that functions from  $\mathbb{F}_q^n$  to  $\mathbb{F}_q$  have unique representations as multivariate polynomials where the individual degrees are bounded by  $q-1$ .

Recall that  $m_1 * m_2$  denotes the product of these monomials in the ring  $\mathbb{F}_q[X_1, \dots, X_n]$  while  $m_1 \cdot m_2$  denotes their product in  $\mathcal{P}_q(n) = \mathbb{F}_q[X_1, \dots, X_n] / \langle X_1^q - X_1, \dots, X_n^q - X_n \rangle$ . We say that monomials  $m_1, m_2 \in \mathcal{P}_q(n)$  are *disjoint* if  $m_1 * m_2 = m_1 \cdot m_2$  (where the latter monomial is interpreted naturally as an element of  $\mathbb{F}_q[X_1, \dots, X_n]$ ).

Given distinct monomials  $m_1, m_2 \in \mathbb{F}_q[X_1, \dots, X_n]$ , we say that  $m_1 > m_2$  if either one of the following holds:  $\deg(m_1) > \deg(m_2)$ , or  $\deg(m_1) = \deg(m_2)$  and we have  $m_1 = \prod_i X_i^{e_i}$  and  $m_2 = \prod_i X_i^{e'_i}$  where for the least  $j$  such that  $e_j \neq e'_j$ , we have  $e_j > e'_j$ .

The above is called the *graded lexicographic* order on monomials. The ordering obviously restricts to an ordering on the monomials in  $\mathcal{P}_q(n)$ , which are naturally identified as a subset of the monomials of  $\mathbb{F}_q[X_1, \dots, X_n]$ . The well-known fact about this monomial ordering we will use is the following.

► **Fact 2.2.** For any monomials  $m_1, m_2, m_3$ , we have  $m_1 \leq m_2 \Rightarrow m_1 * m_3 \leq m_2 * m_3$ .

Given an  $f \in \mathcal{P}_q(n)$ , we use  $\text{Supp}(f)$  to denote the set of points  $x \in \mathbb{F}_q^n$  such that  $f(x) \neq 0$ . If  $f \neq 0$ , we use  $\text{LM}(f)$  to denote the largest monomial (in the ordering defined above) with non-zero coefficient in  $f$ .

Let  $m = \prod_{i \in [n]} X_i^{e_i}$  with  $e_i < q$  for each  $i$ . For an integer  $s \geq 0$ , we let

$$U_s(m) := \left\{ \prod_{j \in [n]} X_j^{e'_j} \mid \forall j \ q > e'_j \geq e_j, \sum_j e'_j = d + s \right\}$$

$$D_s(m) := \left\{ \prod_{j \in [n]} X_j^{e'_j} \mid \forall j \ e'_j + e_j < q, \sum_j e'_j = s \right\}.$$

Note that the monomials in  $D_s(m)$  are precisely the monomials of degree  $s$  that are disjoint from  $m$ . Further, the map  $\rho : D_s(m) \rightarrow U_s(m)$  defined by  $\rho(m_1) = m_1 \cdot m$  defines a bijection between  $D_s(m)$  and  $U_s(m)$ , and hence we have

► **Fact 2.3.** For any monomial  $m$  and any  $s \geq 0$ ,  $|U_s(m)| = |D_s(m)|$ .

For non-negative integers  $s \leq e$ , we define  $U_{s,e}(m) := \bigcup_{s \leq t \leq e} U_t(m)$  and  $D_{s,e}(m) := \bigcup_{s \leq t \leq e} D_t(m)$  where  $U_t(m)$  and  $D_t(m)$  are as defined in Section 2. Since  $|U_t(m)| = |D_t(m)|$  for each  $t$  (Fact 2.3), we have  $|U_{s,e}(m)| = |D_{s,e}(m)|$ .

### 3 An extension of the Schwartz-Zippel Lemma over $\mathbb{F}_q$

The results of this section hold over  $\mathbb{F}_q$  where  $q$  is any prime power.

► **Lemma 3.1.** *Let  $d, s \geq 0$  be arbitrary integers with  $d+s \leq n(q-1)$ . Assume  $d = (q-1)u+v$  for  $u, v \geq 0$  with  $v < (q-1)$ . Then the monomial  $m_0 := X_1^{q-1} \cdots X_u^{q-1} X_{u+1}^v$  of degree  $d$  satisfies  $|U_s(m_0)| \leq |U_s(m)|$  for all monomials  $m$  of degree exactly  $d$ .*

**Proof.** Fix any monomial  $m$  of degree  $d$  such that  $|U_s(m)|$  is as small as possible; say  $m = \prod_{j \in [n]} X_j^{e_j}$ . By renaming the variables if necessary, we assume that  $e_1 \geq e_2 \geq \cdots \geq e_n$ .

If  $m \neq m_0$ , then we can find an  $i < n$  such that  $0 < e_{i+1} \leq e_i < q-1$ . Consider the monomial  $m' = X_i^{e_i+1} X_{i+1}^{e_{i+1}-1} \prod_{j \notin \{i, i+1\}} X_j^{e_j}$ . We claim that  $|U_s(m')| \leq |U_s(m)|$ . This will complete the proof of the lemma, since it is easy to check that by repeatedly modifying the monomial in this way at most  $d$  times, we end up with the monomial  $m_0$ . By construction, we will have shown that  $|U_s(m_0)| \leq |U_s(m)|$ .

We are left to show that  $|U_s(m')| \leq |U_s(m)|$  or equivalently (Fact 2.3) that  $|D_s(m')| \leq |D_s(m)|$ . To this end, we show that for any  $(n-2)$ -tuple  $\mathbf{e}' = (e'_1, \dots, e'_{i-1}, e'_{i+2}, \dots, e'_n)$ , that  $|D_s(m', \mathbf{e}')| \leq |D_s(m, \mathbf{e}')|$  where  $D_s(m, \mathbf{e}')$  denotes the set of monomials  $\tilde{m} \in D_s(m)$  such that for each  $j \in [n] \setminus \{i, i+1\}$ , the degree of  $X_j$  in  $\tilde{m}$  is  $e'_j$ . To see this, note that  $D_s(m, \mathbf{e}')$  and  $D_s(m', \mathbf{e}')$  are in bijective correspondence with the sets  $S$  and  $T$  respectively, defined as follows:

$$S = \{(d_1, d_2) \mid 0 \leq d_1 \leq a, 0 \leq d_2 \leq b, d_1 + d_2 = r\}$$

$$T = \{(d_1, d_2) \mid 0 \leq d_1 \leq a-1, 0 \leq d_2 \leq b+1, d_1 + d_2 = r\}$$

where  $a := (q-1) - e_i$ ,  $b := (q-1) - e_{i+1}$ , and  $r = s - \sum_{j \notin \{i, i+1\}} e'_j$ ; note that by assumption,  $(q-1) > e_i \geq e_{i+1}$  and hence  $1 \leq a \leq b$ . Our claim thus reduces to showing  $|T| \leq |S|$ , which is done as follows.

If  $r < 0$  or  $r > a+b$ , then both  $S$  and  $T$  are empty sets and the claim is trivial. So assume that  $0 \leq r \leq a+b$ . In this case, we see that  $|T \setminus S| \leq 1$ : in fact,  $T \setminus S$  can only contain the element  $(r-b-1, b+1)$  and this happens only when the inequalities  $0 \leq r-b-1 \leq a-1$  is satisfied. But this allows us to infer that  $S \setminus T$  contains  $(a, r-a)$  since  $0 \leq r-b-1 \leq r-a$  and  $r-a \leq b$ . Thus,  $|T \setminus S| \leq |S \setminus T|$  and hence  $|T| \leq |S|$ . ◀

We have the following immediate corollary of Lemma 3.1.

► **Corollary 3.2.** *Let  $d, e, s \geq 0$  be arbitrary parameters with  $s \leq e$  and  $d \leq n(q-1)$ . Assume  $d = (q-1)u+v$  for  $u, v \geq 0$  with  $v < (q-1)$ . Then the monomial  $m_0 := X_1^{q-1} \cdots X_u^{q-1} X_{u+1}^v$  satisfies  $|U_{s,e}(m_0)| \leq |U_{s,e}(m)|$  for all monomials  $m$  of degree exactly  $d$ .*

The main technical lemma of this section is the following.

► **Lemma 3.3** (Extension of the Schwartz-Zippel lemma over  $\mathbb{F}_q$ ). *Let  $e, d, s \geq 0$  be integer parameters with  $s \leq e$ . Let  $f \in \mathcal{P}_q(n)$  be non-zero and of degree exactly  $d$  with  $\text{LM}(f) = m_1$ . Then,*

$$\Pr_{P \in_R \mathcal{P}_q(n,e)} [\deg(fP) < d+s] \leq \frac{1}{q^{|U_{s,e}(m_1)|}}.$$

*In particular, using Corollary 3.2, the probability above is upper bounded by  $\frac{1}{q^{|U_{s,e}(m_0)|}}$  where the monomial  $m_0$  is as defined in the statement of Corollary 3.2.*

**Proof.** Let  $P = \sum_{m:\deg(m)\leq e} \alpha_m m$  where the  $\alpha_m$  are chosen independently and uniformly at random from  $\mathbb{F}_q$ . Also, let  $f = \sum_{i=1}^N \beta_i m_i$  where  $\beta_i \neq 0$  for each  $i$  and we have  $m_1 > m_2 > \dots > m_N$  in the graded lexicographic order defined earlier.

Thus, we have

$$fP = \left( \sum_{m:\deg(m)\leq e} \alpha_m m \right) \cdot \left( \sum_{i=1}^N \beta_i m_i \right) = \sum_{\tilde{m}} \left( \sum_{(m,j):mm_j=\tilde{m}} \alpha_m \beta_j \right) \tilde{m}.$$

The polynomial  $fP$  has degree  $< d + s$  iff for each  $\tilde{m}$  of degree at least  $d + s$ , its coefficient in the above expression is 0. Since the  $\beta_i$ 's are fixed, we can view this event as the probability that some set of *homogeneous* linear equations in the  $\alpha_m$  variables are satisfied. By standard linear algebra, this is exactly  $q^{-t}$  where  $t$  is the rank of the linear system. So it suffices to show that there are at least  $|U_{s,e}(m_1)|$  many *independent* linear equations in the system.

Recall that  $|D_{s,e}(m_1)| = |U_{s,e}(m_1)|$ . Now, for each  $m \in D_{s,e}(m_1)$ , consider the monomial  $\tilde{m} = m \cdot m_1 = m * m_1$  (the second equality is true since  $m$  is disjoint from  $m_1$ ). Let  $\tilde{\mathcal{M}}$  denote the set of all such  $\tilde{m}$ . Note that each  $\tilde{m} \in \tilde{\mathcal{M}}$  has degree exactly  $\deg(m) + \deg(m_1) \in [d + s, d + e]$ . Thus, for  $fP$  to have degree  $< d + s$ , the coefficient of each  $\tilde{m}$  must vanish. Further, since  $|\tilde{\mathcal{M}}| = |D_{s,e}(m_1)| = |U_{s,e}(m_1)|$  it suffices to show that the linear equations corresponding to the different  $\tilde{m} \in \tilde{\mathcal{M}}$  are all linearly independent.

To prove this, we argue as follows. Let  $m'$  be a monomial of degree at most  $e$ . We say that  $m'$  *influences*  $\tilde{m} \in \tilde{\mathcal{M}}$  if  $\alpha_{m'}$  appears with non-zero coefficient in the equation corresponding to  $\tilde{m}$ . We now make the following claim.

► **Claim 3.4.** *Let  $\tilde{m} \in \tilde{\mathcal{M}}$  and  $m \in D_{s,e}(m_1)$  be such that  $\tilde{m} = m * m_1$ . Then,  $m$  influences  $\tilde{m}$ . Further, if some monomial  $m'$  influences  $\tilde{m}$ , then  $m' \geq m$ .*

Assuming the above claim, we complete the proof of the lemma as follows. Consider the matrix  $B$  of coefficients obtained by writing the above linear system in the following manner. For each  $\tilde{m} = m * m_1 \in \tilde{\mathcal{M}}$ , we have a row of  $B$  and let the rows be arranged from top to bottom in increasing order of  $m$  (w.r.t. the graded lexicographic order). Similarly, for each  $m'$  of degree at most  $e$ , we have a column and again the columns are arranged from left to right in increasing order of  $m'$ . The  $(\tilde{m}, m')$ th entry contains the coefficient of  $\alpha_{m'}$  in the equation corresponding to the coefficient of  $\tilde{m}$ .

Restricting our attention only to columns corresponding to  $m' \in D_{s,e}(m_1)$ , Claim 3.4 guarantees to us that the submatrix thus obtained is a  $|D_{s,e}(m_1)| \times |D_{s,e}(m_1)|$  matrix that is upper triangular with non-zero entries along the diagonal. Hence, the submatrix is full rank. In particular, the matrix  $B$  (and hence our linear system) has rank at least  $|D_{s,e}(m_1)|$ . This proves the lemma. ◀

**Proof of Claim 3.4.** We start by showing that  $m$  does indeed influence  $\tilde{m}$ . The linear equation corresponding to  $\tilde{m}$  is

$$\sum_{(m',j):m' \cdot m_j = \tilde{m}} \beta_j \alpha_{m'} = 0 \tag{2}$$

where  $m'$  runs over all monomials of degree at most  $e$ .

Clearly, one of the summands in the LHS above is  $\beta_1 \alpha_m$ . Thus, to ensure that  $m$  influences  $\tilde{m}$ , it suffices to ensure that no other summand containing the variable  $\alpha_m$  appears. That is, that  $m \cdot m_j \neq \tilde{m}$  for any  $j > 1$ . (Note that in general unique factorization is *not true* in  $\mathcal{P}_q(n)$ , since  $X^q = X$ .)

## 17:10 Robust Multiplication-Based Tests for Reed-Muller Codes

To see this, note further that  $m \cdot m_j$  is either equal to  $m * m_j$  (if they are disjoint) or has smaller degree than  $m * m_j$ . In either case, we have  $m \cdot m_j \leq m * m_j$ . Thus, we obtain

$$m \cdot m_j \leq m * m_j < m * m_1 = \tilde{m}$$

where the second inequality follows from the fact that  $m_1 > m_j$  and hence (Fact 2.2)  $m' * m_1 > m' * m_j$  for any monomial  $m'$ . This shows that  $\alpha_m$  appears precisely once in the left hand side of (2) and in particular, that it must influence  $\tilde{m}$ .

Now, we show that no  $m' < m$  influences  $\tilde{m}$ . Fix some  $m' < m$ . For any  $j \in [N]$  we have

$$m' \cdot m_j \leq m' * m_j \leq m' * m_1 < m * m_1 = \tilde{m}$$

where the first two inequalities follow from a similar reasoning to above and the third from the fact that  $m' < m$ . Hence, we see that no monomial that is a product of  $m'$  with another monomial from  $f$  can equal  $\tilde{m}$ . In particular, this means that  $m'$  cannot influence  $\tilde{m}$ .

This completes the proof of the claim.  $\blacktriangleleft$

► **Corollary 3.5.** *Let  $n, e, d, P, f$  be as in Lemma 3.3. Further, let  $r$  be such that  $(q-1)n-d = r$  and assume  $r \geq 2e + (q-1)$ . Then,  $\Pr_{P \sim \mathcal{P}_q(n,e)}[\deg(fP) < d+e] \leq q^{-q^{\Omega(e/q)}}$ .*

**Proof.** To prove the corollary, we use Lemma 3.3 with  $s = e$  and prove a lower bound on  $|U_{e,e}(m_0)| = |U_e(m_0)| = |D_e(m_0)|$  where  $m_0$  is the monomial from the statement of Lemma 3.1. Let  $T$  index the  $t = \lfloor \frac{r}{q-1} \rfloor$  variables not present in the monomial  $m_0$ . We can lower bound  $|D_e(m_0)|$  by the number of monomials of degree exactly  $e$  in  $\mathcal{P}_q(n, e)$  supported on variables from  $T$ ; let  $\mathcal{M}$  denote this set of monomials.

Partition  $T$  arbitrarily into two sets  $T_1$  and  $T_2$  such that  $|T_1| = e' = \lfloor e/(q-1) \rfloor$ .

To lower bound  $|\mathcal{M}|$ , note that given any monomial  $m_1$  in  $\mathcal{P}_q(n, e)$  in the variables of  $T_1$ , we can find a monomial  $m_2$  over the variables of  $T_2$  such that their product has degree  $e$ . The reason for this is that  $m_1$  can have degree at most  $e'(q-1) \leq e$  and further, the maximum degree of any monomial in the variables in  $T_2$  is

$$(t - e')(q-1) \geq \left( \frac{r}{q-1} - 1 - \frac{e}{q-1} \right) (q-1) = r - e - (q-1) \geq e$$

where the last inequality follows from our assumed lower bound on  $r$ . Hence, we can always find a monomial  $m_2$  such that  $\deg(m_1 m_2) = e$ . Hence, we can lower bound  $|\mathcal{M}|$  by the number of monomials  $m_1$  over the variables in  $T_1$  which is  $q^{|T_1|} = q^{\Omega(e/q)}$ . We have thus shown that  $|U_{e,e}(m_0)| = q^{\Omega(e/q)}$ . An application of Lemma 3.3 now implies the corollary.  $\blacktriangleleft$

### 3.1 Connection to the Schwartz-Zippel Lemma over $\mathbb{F}_q$

Consider the special case of Lemma 3.3 when  $e = (q-1)n$  and  $s = 0$ . In this case, note that  $\mathcal{P}_q(n, e)$  is just the ring  $\mathcal{P}_q(n)$  and hence the above lemma implies  $\Pr_{P \sim \mathcal{P}_q(n)}[\deg(fP) < d] \leq \frac{1}{q^{|U_{s,e}(m_0)|}}$  where  $m_0$  is the monomial from the statement of Lemma 3.1. Note that as a special case, this implies that  $\Pr_{P \sim \mathcal{P}_q(n)}[fP = 0] \leq \frac{1}{q^{|U_{s,e}(m_0)|}}$ .

Observe that by Fact 2.1,  $fP = 0$  if and only if the polynomial  $fP$  vanishes at each point of  $\mathbb{F}_q^n$ . However, since  $P$  evaluates to an independent random value in  $\mathbb{F}_q$  at each input  $x \in \mathbb{F}_q^n$ , we see that the probability that  $fP$  evaluates to 0 at each point is exactly the probability that  $P(x) = 0$  at each point where  $f(x) \neq 0$ . This happens with probability exactly  $\frac{1}{q^{|\text{Supp}(f)|}}$ .

Putting it all together, we see that  $\frac{1}{q^{|\text{Supp}(f)|}} \leq \frac{1}{q^{|U_{s,e}(m_0)|}}$  and hence,  $|\text{Supp}(f)| \geq |U_{s,e}(m_0)| = |D_{s,e}(m_0)|$ .

For the chosen values of  $e$  and  $s$ , the latter quantity is exactly the total number of monomials – of *any* degree – that are disjoint from  $m_0$ , which is exactly  $(q - v)q^{n-u-1}$ , matching the Schwartz-Zippel lemma over  $\mathbb{F}_q$  (Fact 2.1).

It is also known that the Schwartz-Zippel lemma over  $\mathbb{F}_q$  is tight for a suitably chosen degree  $d$  polynomial  $f$ . Lemma 3.3 is also tight for the same polynomial  $f$ . This fact is not required for the other results of this paper and thus we defer it to the full version.

#### 4 Analyzing Test $_{e,k}$

We prove the main theorem of the paper, namely Theorem 1.4, in this section. The results of this section only hold for *prime* fields. For lack of space, a part of the proof has been omitted.

We argue that the theorem holds by considering two cases. We argue that when  $f$  is  $\Delta$ -far from polynomials of degree  $d + r/4$  – a much stronger assumption than the hypothesis of the theorem – then a modification of the proof of Dinur and Guruswami [4] coupled with a suitable choice of basis for  $\mathcal{P}_q(n, d)$  (see the full version for details) yields the desired conclusion.

If not, then  $f$  is  $\Delta$ -close to some polynomial of degree  $d'$  that is slightly larger than  $d$ . In this case, we can argue that  $f$  is “essentially” a polynomial of degree  $d'$  and for any such polynomial, the product  $fP_1 \dots P_k$  is, w.h.p., a polynomial of degree exactly  $d' + ek$  and hence  $f \notin \mathcal{P}_q(n, d + ek)$ . This requires the results of Section 3.

We will assume throughout that  $r$  is greater than or equal to some fixed constant (possibly depending on  $q, k$ ) since otherwise the statement of the theorem is trivial.

**Case 1:  $f$  is  $\Delta$ -far from  $\mathcal{P}_q(n, d + \frac{r}{4})$ .** For lack of space, this section has been omitted.

See the full version for details.

**Case 2:  $f$  is  $\Delta$ -close to  $\mathcal{P}_q(n, d + \frac{r}{4})$ .** Let  $F \in \mathcal{P}_q(n, d + \frac{r}{4})$  be such that  $f$  is  $\Delta$ -close to  $F$ . Let  $d' = \deg(F)$ . Note that  $d' > d$  since  $f$  is  $\Delta$ -far from  $\mathcal{P}_q(n, d)$  by assumption. Hence, we must have  $d < d' \leq d + \frac{r}{4}$ .

Note that for any  $P_1, \dots, P_k \in \mathcal{P}_q(n, e)$ , we have  $fP_1 \dots P_k$  is  $\Delta$ -close to  $FP_1 \dots P_k$  (since  $f(x) = F(x)$  implies that  $f(x) \cdot \prod_i P_i(x) = F(x) \cdot \prod_i P_i(x)$ ). We have  $FP_1 \dots P_k \in \mathcal{P}_q(n, d' + r/4) \subseteq \mathcal{P}_q(n, d + r/2)$ . Now if  $fP_1 \dots P_k \in \mathcal{P}_q(n, d + ek) \subseteq \mathcal{P}_q(n, d + r/2)$ , then by the Schwartz Zippel lemma over  $\mathbb{F}_q$  (Fact 2.1) applied to polynomials of degree at most  $d + r/2$ , we see that  $fP_1 \dots P_k = FP_1 \dots P_k$ . Hence, we have  $FP_1 \dots P_k \in \mathcal{P}_q(n, d + ek)$  which in particular implies that  $FP_1 \dots P_k$  must have degree strictly less than  $d' + ek$ .

For this event to occur there must be some  $i < k$  such that  $FP_1 \dots P_i$  has degree exactly  $d'_i := d' + ei$  but  $FP_1 \dots P_{i+1}$  has degree strictly less than  $d'_i + e$ .

The above reasoning implies

$$\begin{aligned} \Pr_{P_1, \dots, P_k} [fP_1 \dots P_k \in \mathcal{P}_q(n, d + ek)] &\leq \Pr_{P_1, \dots, P_k} [\deg(FP_1 \dots P_k) < d' + ek] \\ &\leq \sum_{i=1}^k \Pr_{P_1 \dots P_k} [\deg(FP_1 \dots P_{i-1}P_i) < d'_i + e \mid \deg(FP_1 \dots P_{i-1}) = d'_i]. \end{aligned} \quad (3)$$

For each  $i$ , conditioning on any fixed choice of  $P_1, \dots, P_{i-1}$ , the right hand side of (3) can be bounded by  $q^{-q^{\Omega(e/q)}} = q^{-q^{\Omega(r)}}$  using Corollary 3.5 applied with  $d$  replaced by  $d'_i \leq d + r/2 - e = (q - 1)n - (r/2 + e)$ . This implies Theorem 1.4 in this case.

## 5 Two applications

### 5.1 A question of Dinur and Guruswami

In this section, we show how Theorem 1.4 implies Lemma 1.5, thus answering a open question raised by Dinur and Guruswami [4].

**Proof of Lemma 1.5.** The proof of the lemma for robustness  $\Delta'$  can be reduced to Theorem 1.4 for  $k = 1 + \lceil \log_{q/(q-1)}(2\Delta') \rceil$  as follows.

Let  $f$  be  $\Delta'$ -far from  $\mathcal{P}_q(n, d)$  as stated in the lemma. Call  $P$  “lucky” if  $\Delta(f \cdot P, \mathcal{P}_q(n, d + e)) \leq \Delta'$ . We need to bound the probability  $\Pr_{P \in \mathcal{P}_q(n, e)}[P \text{ is lucky}]$ . For a lucky  $P$ , let  $F$  be a degree- $(d + e)$  polynomial that is  $\Delta'$ -close to  $f \cdot P$ . Define  $k := 1 + \lceil \log_{q/(q-1)}(2\Delta') \rceil$ . Now, choose  $P_1, \dots, P_{k-1} \in_R \mathcal{P}_q(n, e)$  and let  $g = fP \cdot \prod_{i < k} P_i$ . Also, let  $G = F \cdot \prod_{i < k} P_i$ ; note that  $G \in \mathcal{P}_q(n, d + ek)$ .

Observe that for any  $x$  such that  $F(x) \neq f(x)P(x)$ , the probability that  $G(x) \neq g(x)$  is at most the probability that all the  $P_i(x)$  are non-zero and this is  $(1 - \frac{1}{q})^{k-1} \leq \frac{1}{2\Delta'}$ . Hence, the probability that any point of difference between  $F$  and  $fP$  survives as a point of difference between  $G$  and  $g$  is at most  $\frac{1}{2}$ . Since no new points of difference are introduced, we see that

$$\begin{aligned} & \Pr_{P, P_1, \dots, P_{k-1}} [fP_1P_2 \cdots P_k \in \mathcal{P}_q(n, d + ek)] \\ & \geq \Pr_P [P \text{ is lucky}] \cdot \Pr_{P, P_1, \dots, P_{k-1}} [f \cdot P \cdot \prod_{i < k} P_i \in \mathcal{P}_q(n, d + ek) \mid P \text{ is lucky}] \\ & = \Pr_P [P \text{ is lucky}] \cdot \Pr_{P, P_1, \dots, P_{k-1}} [g \in \mathcal{P}_q(n, d + ek) \mid P \text{ is lucky}] \\ & \geq \Pr_P [P \text{ is lucky}] \cdot \Pr_{P, P_1, \dots, P_{k-1}} [g = G \mid P \text{ is lucky}] \geq \Pr_P [P \text{ is lucky}] \cdot \frac{1}{2}. \end{aligned}$$

The lemma now follows since Theorem 1.4 implies that  $\Pr_{P, P_1, \dots, P_{k-1}} [fP_1P_2 \cdots P_k \in \mathcal{P}_q(n, d + ek)] \leq q^{-q^{\Omega(r)}}$ .  $\blacktriangleleft$

► **Remark 5.1.** An anonymous reviewer for FSTTCS 2016 pointed out to us that Lemma 1.5 only works if  $\log \Delta' = O(k)$ , which in particular implies that  $\Delta'$  must be a constant (independent of  $n$  and  $d$ ). However, an easy modification of the above idea actually shows a statement of the above form for  $\Delta'$  as large as  $q^{\Omega(r)}$ . We refer the reader to the full version for details.

### 5.2 Analysis of Corr- $h$

Recall the test Corr- $h$  defined in the introduction where  $h \in \mathcal{P}_q(n, k)$  is a polynomial of exact degree  $k$ . In this section, we analyze this test Corr- $h$ , thus proving Corollary 1.6.

For this we need the following properties of polynomials.

**Dual of  $\mathcal{P}_q(n, d)$ :** For any two functions,  $f, g \in \mathcal{F}_q(b)$ , define  $\langle f, g \rangle := \sum_{x \in \mathbb{F}_q^n} f(x) \cdot g(x)$ .

The set of polynomials  $\mathcal{P}_q(n, r - 1)$  is the dual to the set of polynomials  $\mathcal{P}_q(n, d)$  in the following sense.

- For any two polynomials  $P \in \mathcal{P}_q(n, d)$  and  $Q \in \mathcal{P}_q(n, r - 1)$ , we have  $\langle P, Q \rangle = 0$ .
- Furthermore, for any  $P \notin \mathcal{P}_q(n, d)$  and a random  $Q \in_R \mathcal{P}_q(n, r - 1)$ , we have that  $\langle P, Q \rangle$  is an unbiased element of  $\mathbb{F}_q$ .

This implies that the indicator variable for the event “ $f \in \mathcal{P}_q(n, d)$ ” can be equivalently written as  $\mathbb{1}_{f \in \mathcal{P}_q(n, d)} = \mathbf{E}_{Q \in \mathcal{P}_q(n, r-1)} [\omega^{\langle f, Q \rangle}]$ , where  $\omega = e^{2\pi i/q}$ .

**Squaring trick:** We use a standard squaring trick to bound the absolute value of the quantity  $\mathbf{E}_P [\omega^{\langle h(P), f \rangle}]$ . Let us consider the case when  $h(P) = P^2$ . In this case we have

$$\begin{aligned} \left| \mathbf{E}_P [\omega^{\langle P^2, f \rangle}] \right|^4 &= \left| \mathbf{E}_{P, P_1} [\omega^{\langle (P+P_1)^2, f \rangle} \cdot \omega^{\langle -P^2, f \rangle}] \right|^2 = \left| \mathbf{E}_{P, P_1} [\omega^{\langle 2PP_1+P_1^2, f \rangle}] \right|^2 \\ &\leq \mathbf{E}_{P_1} \left[ \left| \mathbf{E}_P [\omega^{\langle 2PP_1+P_1^2, f \rangle}] \right|^2 \right] \\ &= \mathbf{E}_{P_1} \left[ \mathbf{E}_{P, P_2} [\omega^{\langle 2(P+P_2)P_1+P_1^2, f \rangle} \cdot \omega^{\langle -(2PP_1+P_1^2), f \rangle}] \right] \\ &= \mathbf{E}_{P_1} \left[ \mathbf{E}_{P, P_2} [\omega^{\langle 2P_1P_2, f \rangle}] \right] = \mathbf{E}_{P_1, P_2} [\omega^{\langle 2P_1P_2, f \rangle}] \end{aligned}$$

A similar argument shows that when  $h(P)$  is a polynomial of degree exactly  $k$ , we have

$$\left| \mathbf{E}_P [\omega^{\langle h(P), f \rangle}] \right|^{2^k} \leq \mathbf{E}_{P_1, \dots, P_k} [\omega^{\langle k!P_1 \dots P_k, f \rangle}]$$

We are now ready to prove Corollary 1.6.

**Proof of Corollary 1.6.**

$$\begin{aligned} \Pr_{P \in \mathcal{P}_q(n, e)} [f \cdot h(P) \in \mathcal{P}_q(n, d + ek)] &= \left| \mathbf{E}_{P \in \mathcal{P}_q(n, e), Q \in \mathcal{P}_q(n, s-1)} [\omega^{\langle f \cdot h(P), Q \rangle}] \right| \\ &= \left| \mathbf{E}_Q \left[ \mathbf{E}_P [\omega^{\langle h(P), fQ \rangle}] \right] \right|^{2^k/2^k} \\ &\leq \left( \mathbf{E}_Q \left[ \left| \mathbf{E}_P [\omega^{\langle h(P), fQ \rangle}] \right|^{2^k} \right] \right)^{1/2^k} \\ &\leq \left( \mathbf{E}_Q \left[ \mathbf{E}_{P_1, \dots, P_k} [\omega^{\langle k!P_1 \dots P_k, fQ \rangle}] \right] \right)^{1/2^k} \\ &= \left( \mathbf{E}_{P_1, \dots, P_k} \left[ \mathbf{E}_Q [\omega^{\langle P_1 \dots P_k f, Q \rangle}] \right] \right)^{1/2^k} \\ &= \left( \Pr_{P_1, \dots, P_k} \left[ f \cdot \prod_i P_i \in \mathcal{P}_q(n, d + ek) \right] \right)^{1/2^k} \end{aligned}$$

The corollary now follows from Theorem 1.4. ◀

**Acknowledgements.** We thank Madhu Sudan for many encouraging discussions and feedback. We also thank the anonymous reviewers of FSTTCS 2016 for many corrections and pointing out a weakness of Lemma 1.5 (see Remark 5.1).

---

## References

- 1 Noga Alon, Tali Kaufman, Michael Krivelevich, Simon Litsyn, and Dana Ron. Testing Reed-Muller codes. *IEEE Trans. Inform. Theory*, 51(11):4032–4039, 2005. (Preliminary version in *7th RANDOM*, 2003). doi:10.1109/TIT.2005.856958.

- 2 Boaz Barak, Parikshit Gopalan, Johan Håstad, Raghu Meka, Prasad Raghavendra, and David Steurer. Making the long code shorter. *SIAM J. Comput.*, 44(5):1287–1324, 2015. (Preliminary version in *53rd FOCS*, 2012). [arXiv:1111.0405](#), [doi:10.1137/130929394](#).
- 3 Arnab Bhattacharyya, Swastik Kopparty, Grant Schoenebeck, Madhu Sudan, and David Zuckerman. Optimal testing of Reed-Muller codes. In *Proc. 51st IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 488–497, 2010. [arXiv:0910.0641](#), [doi:10.1109/FOCS.2010.54](#).
- 4 Irit Dinur and Venkatesan Guruswami. PCPs via the low-degree long code and hardness for constrained hypergraph coloring. *Israel Journal of Mathematics*, 209:611–649, 2015. (Preliminary version in *54th FOCS*, 2013). [doi:10.1007/s11856-015-1231-3](#).
- 5 Venkat Guruswami, Prahladh Harsha, Johan Håstad, Srikanth Srinivasan, and Girish Varma. Super-polylogarithmic hypergraph coloring hardness via low-degree long codes. In *Proc. 46th ACM Symp. on Theory of Computing (STOC)*, pages 614–623, 2014. [arXiv:1311.7407](#), [doi:10.1145/2591796.2591882](#).
- 6 Elad Haramaty, Amir Shpilka, and Madhu Sudan. Optimal testing of multivariate polynomials over small prime fields. *SIAM J. Comput.*, 42(2):536–562, 2013. (Preliminary version in *52nd FOCS*, 2011). [doi:10.1137/120879257](#).
- 7 Sangxia Huang.  $2^{(\log N)^{1/10-o(1)}}$  hardness for hypergraph coloring. (manuscript), 2015. [arXiv:1504.03923](#).
- 8 Tadao Kasami, Shu Lin, and W. Wesley Peterson. Polynomial codes. *IEEE Trans. Inform. Theory*, 14(6):807–814, 1968. [doi:10.1109/TIT.1968.1054226](#).
- 9 Subhash Khot and Rishi Saket. Hardness of coloring 2-colorable 12-uniform hypergraphs with  $2^{(\log n)^{\Omega(1)}}$  colors. In *Proc. 55th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 206–215, 2014. [doi:10.1109/FOCS.2014.30](#).
- 10 Girish Varma. Reducing uniformity in Khot-Saket hypergraph coloring hardness reductions. *Chicago J. Theor. Comput. Sci.*, 2015(3):1–8, 2015. [arXiv:1408.0262](#), [doi:10.4086/cjtcs.2015.003](#).



# Querying Regular Languages over Sliding Windows

Moses Ganardi<sup>1</sup>, Danny HucKe<sup>2</sup>, and Markus Lohrey<sup>\*3</sup>

1 University of Siegen, Germany

ganardi@eti.uni-siegen.de

2 University of Siegen, Germany

hucKe@eti.uni-siegen.de

3 University of Siegen, Germany

lohrey@eti.uni-siegen.de

---

## Abstract

We study the space complexity of querying regular languages over data streams in the sliding window model. The algorithm has to answer at any point of time whether the content of the sliding window belongs to a fixed regular language. A trichotomy is shown: For every regular language the optimal space requirement is either in  $\Theta(n)$ ,  $\Theta(\log n)$ , or constant, where  $n$  is the size of the sliding window.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, F.4.3 Formal Languages

**Keywords and phrases** streaming algorithms, regular languages, space complexity

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.18

## 1 Introduction

*Streaming algorithms*, i.e. algorithms that process a non-terminating stream  $a_1a_2a_3\cdots$  of data values and which have at time  $t$  only direct access to the current symbol  $a_t$ , received a lot of attention in recent years, see [1] for a general reference. Two variants of streaming algorithms can be found in the literature:

- In the *standard model* the algorithm computes at time  $t$  a value  $f(a_1 \cdots a_t)$  that depends on the whole history.
- In the *sliding window model* the algorithm computes at time  $t$  a value  $f(a_{t-n+1} \cdots a_t)$  that depends on the  $n$  last symbols (we should assume  $t \geq n$  here). The value  $n$  is also called the *window size*.

For many applications, the sliding window model is more appropriate. Quite often data items in a stream are outdated after a certain time, and the sliding window model is a simple way to model this. The typical application is the analysis of a time series as it may arise in medical monitoring, web tracking, or financial monitoring. In all these applications, the most recent data items are more important than older ones.

A general goal in the area of sliding window algorithms is to avoid the explicit storage of the whole window, and, instead, to work in considerably smaller space, e.g. polylogarithmic space. In the seminal paper of Datar et al. [9], where the sliding window model was introduced, the authors prove that the number of 1's in a 0/1-sliding window of size  $n$  can be maintained in space  $\frac{1}{\varepsilon} \cdot \log^2 n$  if one allows a multiplicative error of  $1 \pm \varepsilon$ . A matching lower bound is provided as well in [9]. Other algorithmic problems that were addressed in

---

\* M. Lohrey was supported by the DFG-project LO748/11-1.



the extensive literature on sliding window streams include the computation of statistical data (e.g. computation of the variance and  $k$ -median [4], and quantiles [3]), optimal sampling from sliding windows [7], database querying (e.g. processing of join queries over sliding windows [12]) and graph problems (e.g. checking for connectivity and computation of matchings, spanners, and minimum spanning trees [8]). The reader can find further references in the surveys [1, Chapter 8] and [6]. Another natural problem, whose investigation has so far been surprisingly neglected for the sliding window model, is the membership problem for a language or, equivalently, the computation of Boolean queries on the sliding window. In its general form, one fixes a language  $L$  over the alphabet of the data stream, and asks for an algorithm that can check at any time whether the content of the sliding window belongs to  $L$ . In this paper, we are mainly interested in the case, where  $L$  is a regular language.

Note that in the standard streaming model, it is trivial to solve the membership problem for a regular language  $L$  in constant space. For a data stream  $a_1a_2a_3\cdots$  the algorithm simply runs a deterministic finite automaton for  $L$  and only stores the current state (which needs constant space since we assume the automaton to be fixed and not part of the input). This obvious fact might explain why the membership problem for regular languages in the streaming model has not received any attention so far. In contrast, there exist papers that deal with membership problems for (restricted classes of) context-free languages in the standard streaming model, see the paragraph on related work below.

Note that in the sliding window model the above algorithm (simulation of a DFA on the data stream) does not work. The problem is the removal of the left-most symbol from the sliding window in each step. A naïve approach is to store the whole window in  $O(n)$  bits and simulate the DFA on this word. In fact, there exist very simple languages  $L$  for which this is the best possible solution in order to be able to decide at any point of time whether the current content of the sliding window belongs to  $L$ . An example is the language  $a\{a,b\}^*$  of all words that start with  $a$ . The point is that by repeated checking whether the sliding window content belongs to  $a\{a,b\}^*$ , one can recover the exact content of the sliding window, which implies that every sliding window algorithm for querying  $a\{a,b\}^*$  has to use  $n$  bits of storage (where  $n$  is the window size). The main result of this paper is a trichotomy: The optimal space needed for querying a regular language  $L$  in the sliding window model falls into three classes with respect to its growth rate: constant space,  $\Theta(\log n)$ , and  $\Theta(n)$ , where  $n$  is the window size. We characterize the regular languages by its optimal growth rate algebraically in terms of the syntactic homomorphism and the left Cayley graph of the syntactic monoid of a regular language. The precise characterizations are a bit technical and will be presented in Section 4.

The sliding window model we have talked about so far is also known as the *fixed-size model*, since the sliding window has a fixed size  $n$ . In the literature there exists a second model as well which is known as the *variable-size model*, see e.g. [3]. In this model, the arrival of new data items and the expiration of old items can happen independently, which means that the sliding window can grow and shrink. We also determine the space complexity of querying a regular language for the variable-size model. Again, we prove the same trichotomy as above (constant space,  $\Theta(\log n)$ , and  $\Theta(n)$ ), but the corresponding three classes of regular languages differ slightly from the situation in the fixed-size model.

**Related work.** In [5] the authors consider the problem of membership checking for various subclasses of context-free languages in the standard streaming model (where the whole history is checked for membership). For deterministic linear languages, a randomized streaming algorithm is presented which works in space  $O(\log n)$  and has an inverse polynomial one-sided

error. On the other hand, a visibly pushdown language  $L$  exists, for which every randomized streaming algorithm with an error probability  $< 1/2$  must use space  $\Omega(n)$  [5].

One may consider our streaming algorithms as algorithms for testing membership of a dynamic word in a language  $L$ , where the update operations are restricted. In the variable-size model, these updates are the removal of the first symbol from the word, and appending a given symbol  $a$  to the word. Membership testing algorithms for regular languages that allow the replacement of the symbol at a specified position were studied in [11]. The focus of [11] is on the cell probe complexity of updates and membership queries.

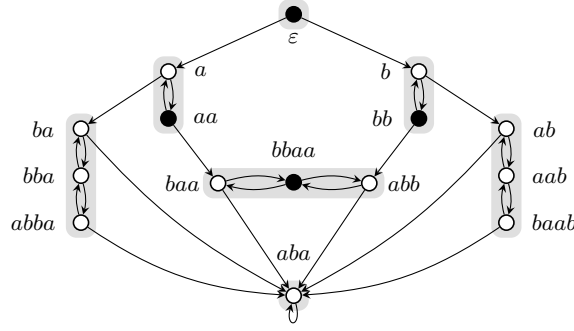
## 2 Preliminaries

Let  $\Sigma$  be a finite alphabet. For a word  $w = a_1 \cdots a_k \in \Sigma^*$  of length  $|w| = k$  we define  $w[i] = a_i$  and  $w[i : j] = a_i \cdots a_j$  if  $i \leq j$  and  $w[i : j] = \varepsilon$  if  $i > j$ . A word  $v \in \Sigma^*$  is a *suffix* of the word  $w$  if there exists a word  $u \in \Sigma^*$  such that  $w = uv$ .

We assume that the reader is familiar with the basic notions of formal languages, in particular regular languages. Our query algorithms for regular languages make use of the description of regular languages by finite monoids; see e.g. the textbook [14] for more details. A *monoid* is a set  $M$  together with an associative operation  $\cdot : M \times M \rightarrow M$  and an element  $1 \in M$  satisfying  $1 \cdot x = x \cdot 1 = x$  for all  $x \in M$ . A function  $h : M \rightarrow N$  between two monoids  $M, N$  is a *homomorphism* if  $h(1) = 1$  and  $h(x \cdot y) = h(x) \cdot h(y)$  for all  $x, y \in M$ . A language  $L \subseteq \Sigma^*$  is *recognized* by a monoid  $M$  if there exists a homomorphism  $h : \Sigma^* \rightarrow M$  from the free monoid  $\Sigma^*$  into a monoid  $M$  and a set  $F \subseteq M$  such that  $w \in L$  if and only if  $h(w) \in F$  for all  $w \in \Sigma^*$ . It is well known that the class of regular languages is exactly the class of languages recognized by finite monoids. For every language  $L \subseteq \Sigma^*$  the *syntactic congruence*  $\equiv_L$  on  $\Sigma^*$  is defined by  $u \equiv_L v$  if and only if for all  $x, y \in \Sigma^*$ :  $xuy \in L$  iff  $xvy \in L$ . The set of congruence classes  $\Sigma^*/\equiv_L$  forms a monoid, which is called the *syntactic monoid* of  $L$  and is denoted by  $M(L)$ . It is the smallest monoid which recognizes  $L$ . The function  $h : \Sigma^* \rightarrow M(L)$  which maps a word  $u$  to its congruence class  $[u]_{\equiv_L}$  is a homomorphism, called the *syntactic homomorphism* of  $L$ .

In this paper all graphs are finite, directed and vertex-colored. For a graph  $\Gamma$  we denote by  $V(\Gamma)$  and  $E(\Gamma)$  the set of vertices and edges of  $\Gamma$ , respectively. Graphs may have loops, i.e.  $E(\Gamma)$  is an arbitrary subset of  $V(\Gamma) \times V(\Gamma)$ . For graphs  $\Gamma$  and  $\Delta$ , a *homomorphism* from  $\Gamma$  to  $\Delta$  is a function  $\varphi : V(\Gamma) \rightarrow V(\Delta)$  such that for all  $v \in V(\Gamma)$  the vertices  $v$  and  $\varphi(v)$  have the same color and  $(u, v) \in E(\Gamma)$  implies  $(\varphi(u), \varphi(v)) \in E(\Delta)$ . We call a graph  $\Gamma$  *homomorphic* to  $\Delta$  if there exists a homomorphism from  $\Gamma$  to  $\Delta$ . For a subset  $S \subseteq V(\Gamma)$  we denote by  $\text{reach}_\Gamma(S)$  the subgraph of  $\Gamma$  which is induced by all nodes that are reachable from  $S$ . A graph  $\Gamma$  is *strongly connected* if for all  $u \in V(\Gamma)$  we have  $\text{reach}_\Gamma(\{u\}) = \Gamma$ . A *strongly connected component*, briefly SCC, of  $\Gamma$  is an inclusion maximal subset  $S \subseteq V(\Gamma)$  such that the subgraph induced by  $S$  is strongly connected. The set of SCCs of a graph is partially ordered by  $S_1 \preceq S_2$  iff a vertex in  $S_2$  is reachable from a vertex in  $S_1$ . An SCC of  $\Gamma$  is *trivial* if it consists of a single node  $v$  and  $(v, v) \notin E(\Gamma)$ , otherwise the SCC is called *non-trivial*. A graph is a *directed cycle* if it is strongly connected and every vertex has outdegree (and indegree) 1. Our characterizations of regular languages will refer to homomorphisms from certain graphs (that we define below) to directed cycles. Note that every monochromatic graph is homomorphic to a directed cycle of size one.

For a monoid  $M$  and a subset  $A$  of  $M$  we denote by  $\Gamma(M, A)$  the (*unlabelled*) *left Cayley graph* over the vertex set  $M$  with the edge set  $\{(x, y) \mid y = a \cdot x \text{ for some } a \in A\}$ . If the subset  $A \subseteq M$  generates  $M$ , i.e. every element of  $M$  is a finite product over  $A$ , then  $y \in M$



■ **Figure 1** The left Cayley graph  $\Gamma(M, A, F)$  from Example 1 for the language  $(aa \mid bb)^*$ . Vertices from  $F$  are black and SCCs are shaded.

is reachable from  $x \in M$  in  $\Gamma(M, A)$  if and only if  $x \leq_{\mathcal{L}} y$  in  $M$ , which is defined by

$$x \leq_{\mathcal{L}} y \iff \exists \ell \in M: x = \ell \cdot y. \tag{1}$$

The  $\mathcal{L}$ -equivalence is defined by  $x \equiv_{\mathcal{L}} y$  if and only if  $x \leq_{\mathcal{L}} y \leq_{\mathcal{L}} x$ . For a subset  $F \subseteq M$  we denote with  $\Gamma(M, A, F)$  the graph  $\Gamma(M, A)$ , where in addition all vertices from  $F$  (resp.,  $M \setminus F$ ) are colored with 1 (resp., 0).

► **Example 1.** Consider the regular language  $L = (aa \mid bb)^*$ . Let  $h : \{a, b\}^* \rightarrow M$  be the syntactic homomorphism of  $L$  into its syntactic monoid  $M$  with 15 elements. Figure 1 shows the left Cayley graph  $\Gamma(M, A, F)$ , where  $A = \{h(a), h(b)\}$  and  $F = h(L)$ . Note that every SCC is homomorphic to a directed cycle.

### 3 Sliding window models

In the literature, one distinguishes two sliding window models: The *fixed-size model* and the *variable-size model*, see also [3] for a discussion of these models.

#### 3.1 The fixed-size model

A *data stream* over  $\Sigma$  is an infinite sequence  $a_1 a_2 a_3 \dots$  of symbols  $a_i \in \Sigma$ . The idea is that a data stream represents the sequence of data that is produced by some process. At time  $t$ , the observer of this process can only see symbol  $a_t$ .

Fix an  $n \in \mathbb{N}$ , which is called the *window size*. Moreover, fix a data stream  $a_1 a_2 a_3 \dots$ . At time  $t \geq 0$  the *sliding window* contains the word  $a_{t-n+1} a_{t-n+2} \dots a_t$  consisting of the  $n$  last symbols, where  $a_i = a$  for a distinguished symbol  $a \in \Sigma$  when  $i \leq 0$ . Thus, in the beginning the sliding window is filled with  $a$ 's. Let us denote with  $W_n(t)$  the content of the sliding window at time  $t$ .

In the *fixed-size sliding-window model* we want to answer queries about the window content  $W_n(t)$ , where the window size  $n$  is fixed. For this, the algorithm has at time  $t$  access to the  $n$ -th symbol  $a_t$  and a previously computed data structure, that w.l.o.g. can be assumed to be a bit string  $S_n(t) \in \{0, 1\}^*$ . The goal is to compute the query, based on  $a_t$  and  $S_n(t)$ . The simplest solution is to store (a binary coding of)  $W_n(t)$  in  $S_n(t)$ , but in many cases we can find a better solution, where  $S_n(t)$  is considerably smaller than  $W_n(t)$ . Moreover, we would like to have such a query algorithm for every window size  $n$ . Note that this is a *non-uniform model*: For every  $n$  we may have a different query algorithm. This will

not be crucial for our upper bounds, since our algorithms will work for all window sizes  $n$  (which is a parameter in the algorithms). But working with a non-uniform model makes our lower bounds stronger.

In this paper, we are only interested in Boolean queries, i.e. queries that output a single bit. Let us fix a language  $L \subseteq \Sigma^*$ . We say that  $L$  is *streamable in space  $s(n)$  in the fixed-size model* if for every window size  $n$  there exists an algorithm such that for every input stream  $a_1 a_2 a_3 \dots$  the following holds:

- The algorithm maintains a bit string  $S_n(t)$  of length at most  $s(n)$ , where  $S_n(0)$  is an arbitrary bit string of length at most  $s(n)$  (it corresponds to an initial state).
- At every time  $t \geq 0$ , the algorithm has only access to  $S_n(t)$  and  $a_{t+1}$ . Based on these data, the algorithm computes  $S_n(t+1)$  and decides correctly whether  $W_n(t) \in L$ .

### 3.2 The variable-size model

In the fixed-size model, at every time a new data item arrives and the oldest data item is removed from the window. In contrast, in the *variable-size sliding-window model* the arrival of new data items and the expiration of old items is decoupled and can happen independently. This means that the window can grow and shrink. One can think of an adversary that executes an infinite sequence of operations  $\text{op}_1, \text{op}_2, \text{op}_3 \dots$ , where every  $\text{op}_i$  is either a **pop**-operation or a **push**( $a$ )-operation for a symbol  $a \in \Sigma$ . A **pop**-operation deletes the first symbol from the window; this corresponds to the situation where the first item in the window expires and falls out of the window (if the window is already empty it stays empty after a **pop**). A **push**( $a$ )-operation appends the symbol  $a$  at the right end of the sliding window; this corresponds to the arrival of an  $a$  in the data stream. In this way we can define for an infinite sequence  $\text{op}_1, \text{op}_2, \text{op}_3 \dots$  of operations  $\text{op}_i \in \{\text{pop}\} \cup \{\text{push}(a) \mid a \in \Sigma\}$  the window content  $W(t)$  at time  $t \in \mathbb{N}$ , where  $W(0) = \varepsilon$ . We say that the language  $L$  is *streamable in space  $s(n)$  in the variable-size model* if there exists an algorithm such that for every infinite sequence  $\text{op}_1, \text{op}_2, \text{op}_3 \dots$  of operations the following holds:

- At every time  $t \geq 0$ , the algorithm stores a bit string  $S(t)$  of length at most  $s(|W(t)|)$ , where  $S(0) = \varepsilon$ .
- At time  $t \geq 0$ , the algorithm has only access to  $S(t)$  and the operation  $\text{op}_{t+1}$ . Based on these data, the algorithm computes  $S(t+1)$  and decides correctly whether  $W(t) \in L$ .

Note the uniformity of this definition. There is a single algorithm that has to work for every window size. Also note that if  $L$  is streamable in space  $s(n)$  in the variable-size model, then  $L$  is also streamable in space  $s(n)$  in the fixed-size model.

The variable-size model captures various other streaming models that appeared in the literature. For instance, the standard model that was mentioned in the introduction corresponds to the case where no **pop**-operations are allowed. Another realistic model is the *time-stamp based model*, where the data items arrive at arbitrary time points (which are real numbers) and the sliding window contains all data values with an arrival time from the interval  $[t - \tau, t]$ , where  $t$  is the current time and  $\tau$  is a fixed duration. Also the time-stamp based model can be simulated by the variable-size model, see [3] for details.

## 4 Streaming algorithms for regular languages

In this section, we will prove our main results. Let  $L \subseteq \Sigma^*$  be a regular language. We will query the content of the sliding window for membership in  $L$ . Let  $M = M(L)$  be the syntactic monoid of  $L$  and  $h : \Sigma^* \rightarrow M$  be the syntactic homomorphism. Let  $F = h(L)$ , hence  $L = h^{-1}(F)$ . We simply write  $\Gamma$  for the two-colored left Cayley graph  $\Gamma(M, A, F)$

■ **Table 1** The trichotomy results for querying regular languages in the sliding window model.

	constant space	logarithmic space	linear space
fixed-size model	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
variable-size model	$\{\emptyset, \Sigma^*\}$	$(\mathcal{C}_1 \cup \mathcal{C}_2) \setminus \{\emptyset, \Sigma^*\}$	$\mathcal{C}_3$

where  $A = h(\Sigma)$ . Recall that  $\Gamma$  is a finite directed graph, possibly with loops. For the rest of this section we fix  $\Sigma$ ,  $L$ ,  $M$ ,  $h$ ,  $A$ , and  $\Gamma$ . It is important for our results that  $L$  is fixed, and not part of the input. This implies that the monoid  $M$  and the graph  $\Gamma$  can be hard-wired into our algorithms.

- We partition the set of all regular languages over  $\Sigma$  into three classes  $\mathcal{C}_1$ ,  $\mathcal{C}_2$ ,  $\mathcal{C}_3$ , where
- $L \in \mathcal{C}_1$  if and only if for every non-trivial SCC  $S$  of  $\Gamma$  the subgraph  $\text{reach}_\Gamma(S)$  is homomorphic to a directed cycle,
  - $L \in \mathcal{C}_2$  if and only if  $L \notin \mathcal{C}_1$  and every SCC of  $\Gamma$  is homomorphic to a directed cycle,
  - $L \in \mathcal{C}_3$  if and only if  $L \notin (\mathcal{C}_1 \cup \mathcal{C}_2)$ .

For instance, the language  $(aa \mid bb)^*$  from Example 1 belongs to  $\mathcal{C}_2$ . Other examples for languages in  $\mathcal{C}_1 \cup \mathcal{C}_2$  are *open* languages, i.e. languages of the form  $\Sigma^*L$  where  $L$  is a regular language over  $\Sigma$ . Examples for languages in  $\mathcal{C}_1$  are languages of the form  $\Sigma^*w$  for  $w \in \Sigma^*$ .

For the fixed-size model we will show that (i) languages in  $\mathcal{C}_1$  are streamable in constant space, (ii) languages in  $\mathcal{C}_2$  are streamable in space  $O(\log n)$  but not streamable in space  $o(\log n)$ , and (iii) languages in  $\mathcal{C}_3$  are not streamable in space  $o(n)$ . For the variable-size model, languages in  $\mathcal{C}_3$  are still not streamable in space  $o(n)$ , but here only the languages  $\emptyset$  and  $\Sigma^*$  are streamable in constant space. The remaining languages  $(\mathcal{C}_1 \cup \mathcal{C}_2) \setminus \{\emptyset, \Sigma^*\}$  are streamable in space  $O(\log n)$  but not streamable in space  $o(\log n)$  in the variable-size model. Table 1 summarizes both trichotomies.

► **Example 2.** Let  $L_1 = \{a, b\}^*a$  be the set of all words that end with an  $a$ . Obviously,  $L_1$  is streamable in constant space in the fixed-size model: The algorithm has to store nothing. At each time  $t$  one can determine from the current symbol  $a_t$  whether the window content belongs to  $L_1$ , which is the case for  $a_t = a$ . Similarly, for every finite word  $w \in \{a, b\}^*$  the language  $\{a, b\}^*w$  is streamable in constant space in the fixed-size model: The algorithm has to store the last  $|w| - 1$  symbols from the stream. Note that this argument fails for the variable-size model: In fact,  $L_1$  is not streamable in constant space in the variable-size model; this follows from Theorem 7 in Section 4.2.

► **Example 3.** Let  $L_2 = \{a, b\}^*a\{a, b\}^*$  be the set of all words that contain an  $a$ . This language is streamable in space  $O(\log n)$  in the variable-size model. The algorithm stores (i) the current window size  $n$  (using  $O(\log n)$  bits), and (ii) the position  $p$  of the right-most  $a$  in the window (using  $O(\log n)$  bits). We set  $p$  to 0 if the window contains no  $a$ . This information can be easily updated: For a **pop**-operation, the algorithm sets  $n := \max\{0, n - 1\}$  and  $p := \max\{0, p - 1\}$ . For a **push**( $a$ )-operation, the algorithm sets  $n := n + 1$  and  $p := n$ . Finally, for a **push**( $b$ )-operation only  $n$  is incremented.

On the other hand,  $L_2$  is not streamable in space  $o(\log n)$  in the fixed-size model: If  $L_2$  would be streamable in space  $o(\log n)$  then one could represent every number  $1 \leq i \leq n$  by a bit string of length  $o(\log n)$ , namely by the  $o(\log n)$ -size data structure  $d(i)$  obtained after moving the word  $b^{i-1}ab^{n-i}$  into the sliding window, where  $n$  is the window size. To recover  $i$  from  $d(i)$  one continues the stream with  $b$ 's and thereby simulates the query algorithm for  $L_2$

starting with the data structure  $d(i)$ . The smallest number of  $b$ 's after which a membership query for  $L_2$  is answered negatively is  $i$ .

► **Example 4.** Let  $L_3 = a\{a, b\}^*$  be the set of all words that start with an  $a$ . An argument similar to Example 3 shows that  $L_3$  is not streamable in space  $o(n)$  in the fixed-size model. More precisely, if  $L_3$  would be streamable in space  $o(n)$  in the fixed-size model, then one could represent every word  $w \in \{a, b\}^n$  by a bit string of length  $o(n)$ , namely by the  $o(n)$ -size data structure  $d(w)$  obtained after moving the word  $w$  into the sliding window, where  $n$  is the window size. To recover  $w$  from  $d(w)$  one simulates the query algorithm starting with the data structure  $d(w)$ . The query result after seeing  $i - 1$  further symbols from the stream yields the  $i$ -th symbol of  $w$ : A positive (resp., negative) query answer yields an  $a$  (resp.,  $b$ ).

## 4.1 Upper bounds

In this section we will prove two upper bounds on the space for querying regular languages in the sliding window model. First, we show that every language in  $\mathcal{C}_1 \cup \mathcal{C}_2$  is streamable in logarithmic space in both streaming models.

► **Theorem 5.** *If every SCC of  $\Gamma$  is homomorphic to a directed cycle, then  $L$  is streamable in space  $O(\log n)$  in the variable-size model and hence also in the fixed-size model.*

**Proof.** Since the fixed-size model can be simulated by the variable-size model, it suffices to present an algorithm for the variable-size model.

Let  $w \in \Sigma^*$  be a word of length  $n$  and  $\text{Suf}(w)$  be the set of suffixes of  $w$ , which includes the empty word and  $w$  itself. Define the preorder  $\preceq$  on  $\text{Suf}(w)$  by  $u \preceq v$  iff  $h(u) \leq_{\mathcal{L}} h(v)$ , where  $\leq_{\mathcal{L}}$  is defined in (1). This is in fact a total preorder: If  $v \in \text{Suf}(w)$  is a suffix of  $u \in \text{Suf}(w)$  then  $u \preceq v$ . But note that we may have  $u \preceq v \preceq u$  for two different suffixes of  $w$ . The word  $w$  is a smallest element w.r.t.  $\preceq$ . The induced equivalence relation  $\equiv$  is defined by  $u \equiv v$  iff  $u \preceq v \preceq u$ . Clearly,  $u \equiv v$  iff  $h(u) \equiv_{\mathcal{L}} h(v)$ . As usual, denote with  $\text{Suf}(w)/\equiv$  the set of equivalence classes of  $\equiv$ . Note that  $|\text{Suf}(w)/\equiv|$  is bounded by a constant which only depends on the monoid  $M$  and not on the window size  $n$ . One can identify the elements of  $\text{Suf}(w)/\equiv$  with intervals on the set of positions of  $w$ . Hence we can represent  $(\text{Suf}(w), \preceq)$  by storing a constant number of interval endpoints using  $O(\log n)$  bits. Our streaming algorithm (for window size  $n$ ) stores the following data:

- the total preorder  $(\text{Suf}(w), \preceq)$ , using  $O(\log n)$  bits,
- the function  $f : \text{Suf}(w)/\equiv \rightarrow M$  defined by  $f(C) = h(v)$  where  $v$  is the shortest suffix in the equivalence class  $C$ , using  $O(1)$  bits.

We describe these data conveniently by a sequence

$$p_0, m_1, p_1, m_2, p_2, \dots, m_{k-1}, p_{k-1}, m_k, p_k \quad (2)$$

such that the following holds:

- $1 \leq k \leq |M|$ ,
- $0 = p_0 < p_1 < \dots < p_{k-1} < p_k = n + 1$ ,
- $m_1, \dots, m_k \in M$  and  $m_k$  is the unit element of  $M$ .

The meaning of this sequence is the following: The equivalence classes of  $\equiv$  are the sets  $C_i = \{w[p : n] \mid p_{i-1} < p \leq p_i\}$  for  $1 \leq i \leq k$  (the class  $C_k$  contains the empty suffix for  $p = p_k = n + 1$ ). The monoid element  $m_i$  is  $h(w[p_i : n])$  for  $1 \leq i \leq k$  (hence,  $m_k = 1$  is the unit element). Thus,  $m_i = h(v)$  where  $v$  is the shortest suffix in its equivalence class  $C_i$ .

On the sequence (2) we can now perform the desired queries: In order to test whether  $w \in L$ , one has to check whether  $h(w) \in F$ . For this we consider the monoid element  $m_1$ .



Note that  $m_1 \equiv_{\mathcal{L}} h(w)$ . Hence, the vertices  $h(w)$  and  $m_1 = h(w[p_1 : n])$  belong to the same SCC  $S$  of  $\Gamma$ . Note that  $h(w) = h(w[1 : p_1 - 1])m_1$ . We cannot store this word  $w[1 : p_1 - 1]$ , in fact we do not even store its image under  $h$ . But, by assumption, the SCC  $S$  of  $\Gamma$  that contains  $h(w)$  and  $m_1$  has a homomorphism  $\varphi$  onto a directed cycle  $\Theta$ . Thus, we can compute  $\varphi(h(w))$  by traversing the cycle from  $\varphi(m_1)$  for  $p_1 - 1$  steps (the homomorphic image of  $\Gamma$  under  $\varphi$  is hard-wired into the algorithm). The color of  $\varphi(h(w))$  in  $\Theta$  then indicates whether  $h(w) \in F$  (i.e.  $w \in L$ ) or  $h(w) \notin F$  (i.e.  $w \notin L$ ).

For a **pop**-operation on  $w$  and  $p_1 > 1$ , the algorithm updates the sequence (2) to

$$p_0, m_1, p_1 - 1, m_2, p_2 - 1, \dots, m_{k-1}, p_{k-1} - 1, m_k, p_k - 1.$$

Otherwise, if  $p_1 = 1$  then the algorithm updates the sequence (2) to

$$p_1 - 1, m_2, p_2 - 1, \dots, m_{k-1}, p_{k-1} - 1, m_k, p_k - 1.$$

Finally, let us consider a **push(a)**-operation on  $w$ . Note that  $\equiv_{\mathcal{L}}$  is a right congruence, i.e.  $x \equiv_{\mathcal{L}} y$  implies  $xz \equiv_{\mathcal{L}} yz$  for all  $x, y, z \in M$ . This means that our interval-representation of  $(\text{Suf}(wa), \preceq)$  can be obtained from the interval-representation of  $(\text{Suf}(w), \preceq)$  by possibly merging successive intervals. In order to detect, which intervals have to be merged, note that for all  $u, v \in \text{Suf}(w)$  we have

$$ua \equiv va \iff h(u)h(a) \equiv_{\mathcal{L}} h(v)h(a) \iff f([u]_{\equiv})h(a) \equiv_{\mathcal{L}} f([v]_{\equiv})h(a),$$

because  $h(u) \equiv_{\mathcal{L}} f([u]_{\equiv})$  and  $h(v) \equiv_{\mathcal{L}} f([v]_{\equiv})$ , and the fact that  $\equiv_{\mathcal{L}}$  is a right congruence. Using this, we can detect whether two successive intervals that represent the classes  $[u]_{\equiv}$  and  $[v]_{\equiv}$  have to be merged into a single interval. Formally, we process the sequence (2) as follows: We walk over the sequence from left to right. For every  $1 \leq i \leq k - 1$  we check whether  $m_i h(a) \equiv_{\mathcal{L}} m_{i+1} h(a)$ . If this is true, then we remove  $m_i, p_i$  from the sequence, otherwise we replace  $m_i, p_i$  by  $m_i h(a), p_i$ . Then we continue with  $i + 1$  (if  $i < k - 1$ ). Finally, we check whether  $h(a) \equiv_{\mathcal{L}} 1$ . If this holds, then we replace  $m_k, p_k = 1, n + 1$  by  $1, n + 2$ , otherwise we replace  $1, n + 1$  by  $h(a), n + 1, 1, n + 2$ .  $\blacktriangleleft$

Next we show that languages in  $\mathcal{C}_1$  are streamable in constant space in the fixed-size model.

**► Theorem 6.** *Let  $\text{reach}_{\Gamma}(S)$  be homomorphic to a directed cycle for every non-trivial SCC  $S$  of  $\Gamma$ . Then  $L$  is streamable in space  $O(1)$  in the fixed-size model.*

**Proof.** Observe that every path in  $\Gamma$  of length at least  $c := |V(\Gamma)|$  (a constant) contains a vertex in a non-trivial SCC  $S$  and therefore ends in  $\text{reach}_{\Gamma}(S)$ . Fix a window size  $n$ . If  $n < c$ , we store the window content explicitly and can test whether  $w \in L$ , e.g. using an automaton for  $L$ . Now assume  $n \geq c$ . For a window content  $w \in \Sigma^*$  we explicitly store the suffix  $v$  of length  $c$ . Clearly this suffix can be updated when a new symbol arrives in the window. Also  $v$  suffices to test whether  $w \in L$ . We compute  $h(v)$  and a non-trivial SCC  $S$  such that  $h(v)$  is contained in  $\text{reach}_{\Gamma}(S)$ . Let  $\varphi : \text{reach}_{\Gamma}(S) \rightarrow \Theta$  be the homomorphism into a directed cycle  $\Theta$ . Then we compute  $\varphi(h(w))$  by traversing  $\Theta$  starting from the vertex  $\varphi(h(v))$  for  $n - c$  steps. The color of  $\varphi(h(w))$  determines whether  $w \in L$ .  $\blacktriangleleft$

As in most previous work on the sliding window model, our focus is on the space requirements of query algorithms. But it is also interesting to note that in Theorem 5 and 6 we can achieve constant time for all update and query operations on the RAM model with register length  $O(\log n)$ . Let us show this for Theorem 5 first. Recall that the sequence



(2) that we manipulate in the proof of Theorem 5 has constant length. For a `pop`- or `push(a)`-operation, the manipulation of (2) only needs constant time. To see this, note that the numbers  $p_i$  in (2) are only incremented or decremented and that all operations in the monoid  $M$  need constant time since  $M$  is fixed. Finally, for a membership query we traverse the cycle  $\Theta$  starting from  $\varphi(m_1)$  for  $p_1 - 1$  steps. To do this in constant time, we store also the numbers  $(p_i - 1) \bmod \ell(\Theta)$ , where  $\ell(\Theta)$  is the length of the cycle. We have to maintain these remainders for all (constantly many) cycle lengths  $\ell(\Theta_1), \dots, \ell(\Theta_c)$ , where  $\Theta_1, \dots, \Theta_c$  are the cycles to which the SCCs of  $\Gamma$  are homomorphic. For Theorem 6 it suffices to traverse  $\Theta$  for  $(n - c) \bmod \ell(\Theta)$  steps.

## 4.2 Lower bounds

In this section, we prove matching lower bounds for the upper bounds from the previous section. Let us first show that constant space in the variable-size model makes it impossible to query any non-trivial language. Roughly speaking, the reason is that in order to query a non-trivial language in the variable-size model one has to know when the sliding window is empty. But for this, one has to maintain the size of the window, for which  $\log n$  bits are needed.

► **Theorem 7.** *If  $L \subseteq \Sigma^*$  and  $\emptyset \subsetneq L \subsetneq \Sigma^*$ , then  $L$  is not streamable in space  $O(1)$  in the variable-size model.*

**Proof.** Towards a contradiction assume that  $L$  is streamable in the variable-size model in space  $m$ , where  $m$  is a constant, which means that the algorithm has at most  $2^m$  pairwise distinct data structures. We can assume that  $\varepsilon \in L$ , otherwise consider the complement  $\Sigma^* \setminus L$  which is also streamable in space  $m$ . Let further  $w \in \Sigma^*$  be a word such that  $w \notin L$ . Consider the  $2^m + 1$  words  $w^0, w^1, w^2, \dots, w^{2^m}$ . There are two numbers  $0 \leq i < j \leq 2^m$  such that the stream prefixes  $w^i$  and  $w^j$  lead to the same data structure. After  $(j - 1) \cdot |w|$  further `pop`-operations the window contains  $\varepsilon \in L$  and the word  $w \notin L$ , respectively, which is a contradiction. ◀

For the remaining lower bounds, we need the following simple graph theoretic lemma:

► **Lemma 8.** *Let  $\Gamma$  be a finite directed vertex-colored graph (possibly with loops) and let  $s$  be a vertex from which all vertices of  $\Gamma$  are reachable. Assume that all vertices have outdegree  $\geq 1$  and  $s$  has indegree  $\geq 1$ . If  $\Gamma$  is not homomorphic to a directed cycle, then there exist paths  $\pi_0, \pi_1$  of the same length from  $s$  to vertices  $s_0, s_1$  which have distinct colors.*

**Proof.** Let  $V_n$  be the set of vertices which are reachable from  $s$  via a path of length  $n$  for  $n \geq 0$ . The union  $\bigcup_{n \geq 0} V_n$  is the set of vertices reachable from  $s$ , which by assumption is  $V(\Gamma)$ . Towards a contradiction assume that every set  $V_n$  is monochromatic. Let  $\approx$  be the transitive-reflexive closure of the binary relation  $R$  on  $V(\Gamma)$  defined by  $R = \bigcup_{n \geq 0} V_n \times V_n$ . Then, every equivalence class of  $\approx$  is monochromatic. Hence, we can construct the quotient graph  $\Gamma/\approx = (\{[v]_\approx \mid v \in V(\Gamma)\}, \{([u]_\approx, [v]_\approx) \mid (u, v) \in E(\Gamma)\})$ . Moreover, the equivalence class  $[u]_\approx$  has the same color as all its elements. Clearly,  $\Gamma$  is homomorphic to  $\Gamma/\approx$ .

We claim that every vertex in  $\Gamma/\approx$  has out-degree 1: Since every vertex in  $\Gamma$  has outdegree  $\geq 1$ , the same holds for  $\Gamma/\approx$ . Moreover, if a vertex  $v$  is contained in some set  $V_n$ , then all successors of  $v$  are contained in  $V_{n+1}$ . This implies that  $R$  respects the successor relation, i.e. whenever  $(u, v) \in R$  and  $(u, u'), (v, v') \in E(\Gamma)$ , then also  $(u', v') \in R$ . Hence, also  $\approx$  respects the successor relation. This proves that every vertex in  $\Gamma/\approx$  has out-degree 1. Finally, each node has an incoming edge since  $s$  has an incoming edge and all other nodes are reachable from  $s$ . It follows that  $\Gamma/\approx$  must in fact be a directed cycle. ◀



■ **Figure 2** The origin of the words used in the proofs of Theorem 9 (left) and Theorem 10 (right).

Now we show that languages in  $\mathcal{C}_3$  are not streamable in space  $o(n)$  in the fixed-size model.

► **Theorem 9.** *If some SCC  $S$  of  $\Gamma$  is not homomorphic to a directed cycle, then  $L$  is not streamable in space  $o(n)$  in the fixed-size model and hence not streamable in space  $o(n)$  in the variable-size model.*

**Proof.** We apply Lemma 8 with an arbitrary node  $s \in S$  to the subgraph of  $\Gamma$  induced by  $S$ . Therefore, there exist paths  $\pi_0$  and  $\pi_1$  of the same length  $k$  from  $s$  to nodes  $s_0, s_1 \in S$ , which are colored differently, say  $s_0 \notin F$  and  $s_1 \in F$ . Let  $u_0, u_1 \in \Sigma^k$  be words representing the paths  $\pi_0, \pi_1$  and let  $u \in \Sigma^*$  such that  $h(u) = s$ , which exists since  $h$  is surjective. Since  $S$  is strongly connected, there also exist paths  $\pi'_0$  and  $\pi'_1$  from  $s_0$  and  $s_1$ , respectively, back to  $s$ . This results in words  $v_0, v_1 \in \Sigma^+$  such that  $h(v_0 u_0 u) = h(u) = h(v_1 u_1 u)$ ,  $h(u_0 u) \notin F$ ,  $h(u_1 u) \in F$ . The situation is shown in Figure 2 on the left. Choose numbers  $p, q > 0$  such that  $z_0 = (v_0 u_0)^p$  and  $z_1 = (v_1 u_1)^q$  have the same length. We get  $h(z_0 u) = h(z_1 u) = h(u)$ . Let  $x_0, x_1$  such that  $z_0 = x_0 u_0$  and  $z_1 = x_1 u_1$  and hence  $|x_0| = |x_1|$ .

Let  $z = z_0$  (we could also set  $z = z_1$ ). We have  $h(u) = h(z^m u)$  for every  $m$ . Note that  $z \neq \varepsilon$ . By replacing  $x_0$  and  $x_1$  by  $z^m x_0$  and  $z^m x_1$ , respectively, for  $m$  large enough we can therefore assume that  $|x_0| = |x_1| \geq |u|$ . Let  $z = z' z''$  with  $|z''| + |u| = |x_0|$ .

Assume now that  $L$  is streamable in space  $o(n)$  in the fixed-size model. We will deduce a contradiction. Consider an arbitrary bit string  $\alpha = a_1 \cdots a_n \in \{0, 1\}^n$  of length  $n$ . We encode this bit string by the word  $w(\alpha) = z_{a_1} z_{a_2} \cdots z_{a_n} z'$  of length  $n' = \Theta(n)$ . Let  $n'$  be the window size. For  $n$  large enough, there must exist bit strings  $\alpha = a_1 \cdots a_n$  and  $\beta = b_1 \cdots b_n$  of length  $n$  such that  $\alpha \neq \beta$  but after moving  $w(\alpha)$  and  $w(\beta)$  into the sliding window, the same internal data structure arises. Let  $1 \leq i \leq n$  be a position such that w.l.o.g.  $a_i = 0$  and  $b_i = 1$ . We now move the word  $z'' z^{i-1} u$  of length  $(i-1)|z| + |z''| + |u| = (i-1)|z| + |x_0| = (i-1)|z| + |x_1|$  into the sliding window. The window contents are then:

$$\begin{aligned} u_0 z_{a_{i+1}} \cdots z_{a_n} z' z'' z^{i-1} u &= u_0 z_{a_{i+1}} \cdots z_{a_n} z^i u \\ u_1 z_{b_{i+1}} \cdots z_{b_n} z' z'' z^{i-1} u &= u_1 z_{b_{i+1}} \cdots z_{b_n} z^i u \end{aligned}$$

Of course, the stream prefixes  $w(\alpha) z'' z^{i-1} u$  and  $w(\beta) z'' z^{i-1} u$  must still lead to the same data structure. But we have  $h(u_0 z_{a_{i+1}} \cdots z_{a_n} z^i u) = h(u_0 u) \notin F$  and  $h(u_1 z_{b_{i+1}} \cdots z_{b_n} z^i u) = h(u_1 u) \in F$ , which is a contradiction. ◀

For a word  $w \in \Sigma^*$  of length  $k$  we define the signature  $\delta(w) = b_1 \cdots b_k \in \{0, 1\}^*$  such that  $b_i = 1$  if  $h(w[i : k]) \in F$  and  $b_i = 0$  otherwise. To complete our trichotomy, we finally show that languages in  $\mathcal{C}_2$  are not streamable in space  $o(\log n)$  in the fixed-size model.

► **Theorem 10.** *If some SCC  $S$  of  $\Gamma$  is non-trivial and  $\text{reach}_\Gamma(S)$  is not homomorphic to a directed cycle, then  $L$  is not streamable in space  $o(\log n)$  in the fixed-size model and hence not streamable in space  $o(\log n)$  in the variable-size model.*

**Proof.** Let  $S$  be the strongly connected component of  $\Gamma$  which is not trivial and where  $\text{reach}_\Gamma(S)$  is not homomorphic to a directed cycle. Pick an arbitrary node  $s \in S$ . It must have indegree  $\geq 1$  since  $S$  is non-trivial. Since  $h$  is surjective there exists  $u \in \Sigma^*$  with  $h(u) = s$ . We apply Lemma 8 to  $s$  and the subgraph  $\text{reach}_\Gamma(S)$ . This yields words  $x, y \in \Sigma^+$  of equal length, say  $k \geq 1$ , which correspond to the paths  $\pi_0, \pi_1$  in the lemma, such that  $h(xu) \in F$  and  $h(yu) \notin F$ . Further, since  $S$  is strongly connected and non-trivial, there exists a non-trivial path  $\pi$  from  $s$  back to  $s$ , which yields a word  $w \in \Sigma^+$  with  $h(wu) = h(u)$ . The situation is shown in Figure 2 on the right. Let  $\ell = |w|$  and write  $k$  uniquely as  $k = c \cdot \ell + (\ell - p + 1) = (c + 1) \cdot \ell - p + 1$  for  $c \geq 0$  and  $1 \leq p \leq \ell$ . Consider the word  $w[p : \ell]w^c$ . In  $\Gamma$  this word yields the path consisting of  $c$  repetitions of the circle  $\pi$  followed by  $\ell - p + 1$  more steps of  $\pi$  such that the whole path has length  $k$ . If  $h(w[p : \ell]w^c u) \in F$  then we can replace  $x$  by  $w[p : \ell]w^c$ , otherwise we can replace  $y$  by  $w[p : \ell]w^c$ . Without loss of generality we can assume that  $x = w[p : \ell]w^c$ . From  $h(xu) \in F$  and  $h(yu) \notin F$  it follows that the signatures  $\delta(xu)$  and  $\delta(yu)$  differ in the first position. We can assume that for each position  $i > 1$  we have  $\delta(xu)[i] = \delta(yu)[i]$ , otherwise we update the words  $x$  and  $y$  to the suffixes  $x[i : k]$  and  $y[i : k]$ , respectively, where  $i$  is the maximal position such that  $\delta(xu)[i] \neq \delta(yu)[i]$ .

Now assume that  $L$  is streamable in space  $s(n) \in o(\log n)$  in the fixed-size model. We will deduce a contradiction. For  $n$  large enough we consider the  $n$  words  $z_i = ww^{n-i}yw^i$  ( $1 \leq i \leq n$ ) of equal length  $n' = \ell \cdot n + |u| + |y| = \ell \cdot n + |u| + k \in \Theta(n)$ . Large enough here means that  $2^{s(n')} < n$ ; such an  $n$  exists since  $s(n) \in o(\log n)$  and  $n' \in \Theta(n)$ . We now fix the window size to  $n'$  and move the words  $z_i$  ( $1 \leq i \leq n$ ) into the window. Since  $2^{s(n')} < n$ , there exist  $i < j$  such that after moving  $z_i$  and  $z_j$  in the window, the same internal data structure arises. Hence the two stream prefixes  $z_i w^{n-i}u$  and  $z_j w^{n-i}u$  also lead to the same internal data structure. Moreover, after the stream prefix  $z_i w^{n-i}u = ww^{n-i}yw^i u$  the content of the sliding window is  $yw^i u$  (the suffix of  $ww^{n-i}yw^i u$  of length  $n' = \ell \cdot n + |u| + k$ ), which does not belong to  $L$  since  $h(yw^i u) = h(yu) \notin F$ . So, it remains to show that the suffix of length  $n'$  of the stream prefix  $z_j w^{n-i}u = ww^{n-j}yw^{n+j-i}u$  belongs to  $L$ . We distinguish two cases (recall that  $k = c \cdot \ell + (\ell - p + 1)$ ): If  $j - i \geq c + 1$ , then the suffix of  $ww^{n-j}yw^{n+j-i}u$  of length  $n'$  is  $w[p : \ell]w^{n+c}u = xw^i u$  which belongs to  $L$  since  $h(xw^i u) = h(xu) \in F$ . If  $j - i \leq c$ , then the suffix of  $ww^{n-j}yw^{n+j-i}u$  of length  $n' = \ell \cdot n + |u| + k$  is  $y[1 + (j - i)\ell : k]w^{n+j-i}u$ . We have  $h(y[1 + (j - i)\ell : k]w^{n+j-i}u) = h(y[1 + (j - i)\ell : k]u)$ . Now recall that the signatures  $\delta(xu)$  and  $\delta(yu)$  only differ in the first position. Since  $1 + (j - i)\ell \geq 2$  it follows that  $h(y[1 + (j - i)\ell : k]u) \in F$  if and only if  $h(x[1 + (j - i)\ell : k]u) \in F$ . Since  $x = w[p : \ell]w^c$  and  $j - i \leq c$  we have  $x[1 + (j - i)\ell : k] = w[p : \ell]w^{c-j+i}$ . Thus, we have  $h(x[1 + (j - i)\ell : k]u) = h(w[p : \ell]w^{c-j+i}u) = h(w[p : \ell]w^c u) = h(xu) \in F$ , which finally show that  $y[1 + (j - i)\ell : k]w^{n+j-i}u$  belongs to  $L$ .

To sum up, we found two stream prefixes  $z_i w^{n-i}u$  and  $z_j w^{n-i}u$ , which lead to the same internal data structure, but after seeing  $z_i w^{n-i}u$  the window content does not belong to  $L$ , whereas after seeing  $z_j w^{n-i}u$  the window content belongs to  $L$ . This is a contradiction.  $\blacktriangleleft$

## 5 Streaming algorithms for non-regular languages

It would be interesting to know whether our classification can be extended to larger language classes. As a first step, one might consider deterministic context-free languages or the subclass of visibly pushdown languages [2]. All visibly pushdown languages that we have considered so far fall into our trichotomy.

► **Example 11.** Let  $L_4 = \{a^k b^k \mid k \geq 0\}$ . This language is streamable in space  $O(\log n)$  in the variable-size model. The algorithm stores (i) the current window length  $n$  and the unique numbers  $k, m$  such that  $a^k b^m$  is the longest suffix of the window that belongs to  $a^* b^*$ . Note that  $1 \leq k + m \leq n$ . This information can be maintained: For a **pop**-operation,  $k$  and  $m$  are not changed unless  $n = k + m$ . In this case,  $k$  is decremented if  $k > 0$ . If  $k = 0$  then  $m$  is decremented. For a **push**( $b$ )-operation,  $m$  is incremented. Finally, for a **push**( $a$ )-operation, the algorithm sets  $k := 1$  and  $m := 0$  if  $m > 0$ . If  $m = 0$ , then  $k$  is incremented.

Assume that  $L_4$  is streamable in space  $o(\log n)$  in the fixed-size model. Similar to Example 3 we would be able to represent every number  $1 \leq i \leq n$  by a bit string of length  $o(\log n)$ , namely by the data structure obtained by inserting the word  $a^{n+i} b^{n-i}$  into a sliding window of size  $2n$ .

► **Example 12.** Let  $L_5$  be the Dyck-language over a single pair  $(, )$  of brackets. We claim that  $L_5$  is not streamable in space  $o(n)$  in the fixed-size model. In order to get a contradiction, assume that  $L_5$  is streamable in space  $o(n)$  in the fixed-size model. As in Example 4 we deduce that every bit string of length  $n$  can be represented with  $o(n)$  many bits. For this, we encode a bit string  $\alpha = a_1 a_2 \cdots a_n$  ( $a_i \in \{0, 1\}$ ) by the word  $u(\alpha) = u_1 u_2 \cdots u_n$ , where  $u_i = ()$  if  $a_i = 0$  and  $u_i = (())$  if  $a_i = 1$ . Note that  $|u(\alpha)| = 4n$ . We then represent  $\alpha$  by the  $o(n)$ -size data structure  $d(\alpha)$  obtained by moving  $u(\alpha)$  in the sliding window, where the window size is  $4n$ . To recover  $a_i$  from  $d(\alpha)$  one continues the data stream with  $2i - 1$  many repetitions of  $()$ . Then, the window content belongs to  $L_5$  if and only if  $a_i = 0$ .

The following example shows that there exists a non-context-free language whose optimal space requirement is  $\Theta(\sqrt{n})$  in the fixed-size model.

► **Example 13.** Let  $L_6 = \{w^k \mid n \geq 0, w \in \{a, b\}^*, |w| = k\}$ . We claim that in the fixed-size model,  $L_6$  is streamable in space  $O(\sqrt{n})$  but not in space  $o(\sqrt{n})$ . If the window size  $n$  is not a square, then the query algorithm can always answer with no. So, assume that the window size is  $n = m^2$ . The algorithm then stores for the window content  $w$  (i) the length- $m$  suffix  $s$  of  $w$  and (ii) the largest position  $p$  such that  $m + 1 \leq p \leq n$  and  $w[p] \neq w[p - m]$ , where we set  $p = m$  if such a position does not exist. Note that  $w \in L_6$  if and only if  $p = m$ . This information  $s, p$  can be maintained. For  $s$  this is clear. To maintain  $p$ , the algorithm checks whether the next symbol in the stream is the first symbol of  $s$ . If this is the case, the algorithm sets  $p := \max\{p - 1, m\}$ , otherwise it sets  $p := n$ .

The argument that  $L_6$  is not streamable in space  $o(\sqrt{n})$  in the fixed-size model is similar to the argument in Example 4. One shows that from the data structure that is obtained by moving  $w^m$  (with  $w \in \{a, b\}^m$ ) into the sliding window, one can recover the word  $w$ .

In the variable-size model,  $L_6$  is not even streamable in space  $o(n)$ : It is a basic result in communication complexity that equality checking of two words  $x$  and  $y$  of length  $n$  needs  $\Omega(n)$  bits of communication. Assume that  $L_6$  is streamable in space  $o(n)$  in the variable-size model. Then Alice, who initially has access to  $x$ , and Bob, who has access to  $y$ , could check  $x = y$  by exchanging  $o(n)$  bits, where  $n = |x| = |y|$ : Alice pushes the word  $x$  into the window and then sends the  $o(n)$ -size data structure to Bob. Bob then pushes the word  $y^{n-1}$  into the window and afterwards check whether the window content belongs to  $L_6$ , which is the case if and only if  $x = y$ .

In the long version of this paper, we will present for every  $k \geq 2$  an example for a non-deterministic context-free language that in the fixed-size model is streamable in space  $O(n^{1/k})$  but not streamable in space  $o(n^{1/k})$ .

## 6 Future work

Our results on querying regular languages in the sliding window model open several avenues for further research. First of all, one might also consider randomized query algorithms for the sliding window model. For the standard streaming model randomized query algorithms were studied in [5] for subclasses of context-free languages.

We also plan to investigate whether our trichotomy can be extended to larger language classes. As remarked above, it fails for non-deterministic context-free languages. But it is open whether there exists a deterministic context-free language or even a visibly pushdown language  $L$  such that in the fixed-size (resp., variable-size) model  $L$  is streamable in space  $o(n)$  but not streamable in space  $O(\log n)$ .

It would be interesting to know the space complexity of querying regular languages in the sliding window model, when the regular language is part of the input, and, for instance, given by a deterministic finite automaton (DFA). The syntactic monoid of  $L(A)$ , where  $A$  is an  $m$ -state DFA, can have size  $m^m$  [13]. This yields the space bound  $O(\log(n) \cdot m \cdot \log(m) \cdot m^m)$  in the proof of Theorem 5, where  $n$  is the window size. But maybe a better algorithm exists.

Finally, one might also study weighted automata in the sliding window model. A weighted automaton computes for an input word a value from a semiring, which is the Boolean semiring for classical finite automata; see [10] for details. The goal would be to maintain the semiring value to which the sliding window content maps.

**Acknowledgements.** We thank Philipp Reh for spotting a mistake in an earlier version of the paper.

---

## References

- 1 Charu C. Aggarwal. *Data Streams – Models and Algorithms*. Springer, 2007.
- 2 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of STOC 2004*, pages 202–211. ACM Press, 2004.
- 3 Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of PODS 2004*, pages 286–296. ACM, 2004.
- 4 Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O’Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of PODS 2003*, pages 234–243. ACM, 2003.
- 5 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theor. Comput. Sci.*, 494:13–23, 2013.
- 6 Vladimir Braverman. Sliding window algorithms. In *Encyclopedia of Algorithms*, pages 2006–2011. Springer, 2016.
- 7 Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *J. Comput. Syst. Sci.*, 78(1):260–272, 2012.
- 8 Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic graphs in the sliding-window model. In *Proceedings of ESA 2013*, volume 8125 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013.
- 9 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.
- 10 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 2009.
- 11 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997.

## 18:14 Querying Regular Languages over Sliding Windows

- 12 Lukasz Golab and M. Tamer Özsu. Processing sliding window multi-joins in continuous queries over data streams. In *Proceedings of VLDB 2003*, pages 500–511. Morgan Kaufmann, 2003.
- 13 Markus Holzer and Barbara König. On deterministic finite automata and syntactic monoid size. *Theor. Comput. Sci.*, 327(3):319–347, 2004.
- 14 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, Basel, Berlin, 1994.

# Decidability and Complexity of Tree Share Formulas

Xuan Bach Le<sup>1</sup>, Aquinas Hobor<sup>2</sup>, and Anthony W. Lin<sup>3</sup>

1 School of Computing, National University of Singapore, Singapore  
lxbach@comp.nus.edu.sg

2 Yale-NUS College and School of Computing, National University of Singapore, Singapore  
hobor@comp.nus.edu.sg

3 Yale-NUS College, Singapore  
anthony.w.lin@yale-nus.edu.sg

---

## Abstract

Fractional share models are used to reason about how multiple actors share ownership of resources. We examine the decidability and complexity of reasoning over the “tree share” model of Dockins *et al.* using first-order logic, or fragments thereof. We pinpoint a connection between the basic operations on trees union  $\sqcup$ , intersection  $\sqcap$ , and complement  $\bar{\square}$  and countable atomless Boolean algebras, allowing us to obtain decidability with the precise complexity of both first-order and existential theories over the tree share model with the aforementioned operations. We establish a connection between the multiplication operation  $\bowtie$  on trees and the theory of word equations, allowing us to derive the decidability of its existential theory and the undecidability of its full first-order theory. We prove that the full first-order theory over the model with both the Boolean operations ( $\sqcup$ ,  $\sqcap$ ,  $\bar{\square}$ ) and the restricted multiplication operation ( $\bowtie$  with constants on the right hand side) is decidable via an embedding to tree-automatic structures.

**1998 ACM Subject Classification** F.1.1 Models of Computation, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic, F.4.3 Formal Languages

**Keywords and phrases** Fractional Share Models, Resource Accounting, Countable Atomless Boolean Algebras, Word Equations, Tree Automatic Structures

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.19

## 1 Introduction

**The state of the art:** Fractional shares enable reasoning about shared ownership of resources between multiple parties, *e.g.* ownership of memory cells by different threads in a concurrent program [7]. Threads are then allowed to take actions depending on the amount of ownership they have, *e.g.* with full ownership allowing both reading and writing, partial ownership allowing only reading, and empty ownership allowing nothing. Although rational numbers are the most obvious model for fractional shares, they are unfortunately not a good model for realistic program verification because they do not satisfy the so-called “disjointness” axiom [3], *i.e.*  $\forall x, y. x + x = y \Rightarrow x = y = 0$ . Dockins *et al.* proposed a better model for fractional shares based on binary trees with Boolean leaves [10]. A *tree share*  $\tau \in \mathbb{T}$  is inductively defined as follows:  $\tau \triangleq \circ \mid \bullet \mid \begin{array}{c} \wedge \\ \tau \quad \tau \end{array}$ , where  $\circ$  denotes an “empty” leaf while  $\bullet$  a “full” leaf.

The tree  $\circ$  is thus the empty share, and  $\bullet$  the full share. There are two “half” shares:  $\begin{array}{c} \wedge \\ \circ \quad \bullet \end{array}$  and  $\begin{array}{c} \wedge \\ \bullet \quad \circ \end{array}$ , and four “quarter” shares, beginning with  $\begin{array}{c} \wedge \\ \bullet \quad \circ \quad \circ \end{array}$ . It is a feature that the two



© X. Bach Le, Aquinas Hobor, and Anthony W. Lin;  
licensed under Creative Commons License CC-BY

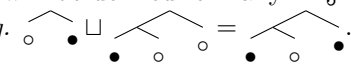
36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 19; pp. 19:1–19:14

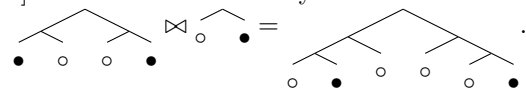


Leibniz International Proceedings in Informatics

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

half shares in  $\mathbb{T}$  are distinct, as compared to the two half shares in  $\mathbb{Q}$ , 0.5 and 0.5, which are of course equal. The ability to represent distinct partial shares of “equal measure” is closely related to why the disjointness axiom holds. The basic operations for combining trees are union  $\sqcup$ , intersection  $\sqcap$ , and complement  $\bar{\square}$ ; these will be defined formally in §2 but to a first approximation they are all defined leafwise, *e.g.* 

A number of program logics incorporate tree shares to model fractional ownership [11, 12, 29, 3], but it has been unclear how to reason about them automatically, which has posed a significant barrier to their use in verification tools. One reason for this barrier is the lack of foundational results regarding decidability and complexity of theories over tree shares. The only published result of this kind proves the decidability of entailment between systems of equations over tree shares, a less-expressive format than general first-order formulae [19].

In addition to union, intersection, and complement, Dockins *et al.* defined a “multiplication” operator on tree shares, written  $\tau_1 \bowtie \tau_2$  [10]. The basic idea is that you take each  $\bullet$  leaf in  $\tau_1$  and replace it with a full copy of  $\tau_2$ , *e.g.* 

Dockins *et al.* showed that  $\bowtie$  could be used to split any nonempty tree  $\tau$  into two nonempty trees that joined together to equal the original since  $\forall \tau. \tau = (\tau \bowtie \widehat{\bullet}) \sqcup (\tau \bowtie \widehat{\circ})$ . More generally, the  $\bowtie$  operator can be used as a kind of “scoping” or “gluing” operator to combine different uses of tree shares together. Although  $\bowtie$  has been used in metatheory [3], it has never been used in an automated tool because its decidability properties were unclear.

**Contributions:** In this paper, we provide the first systematic study of decidability and complexity of theories over the tree share model.

First (§3), we show that the tree share model  $\mathcal{M} \triangleq (\sqcap, \sqcup, \bar{\square}, \circ, \bullet)$  is a Countable Atomless Boolean Algebra (CABA), which are known to be unique up to isomorphism [28]. The first-order theory over CABAs is known to be decidable and, in fact, complete for the class  $\text{STA}(*, 2^{cn}, n)$  of problems solvable by an alternating Turing machine with  $n$  alternations in exponential time [17], the same complexity class as the first-order theory over  $(\mathbb{R}, +, 0, 1)$  [4]. In addition, the full existential theory over CABAs is known to be NP-complete [24]. Our connection shows that these decidability and complexity results transfer to  $\mathcal{M}$ .


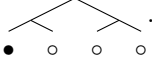
We then (§4) proceed to decision problems over the tree shares with the multiplication operator  $\bowtie$ . Our main result here is that the tree share model  $\mathcal{S} \triangleq (\mathbb{T}, \bowtie)$  that only allows  $\bowtie$  (*i.e.* but not  $\sqcup$ ,  $\sqcap$ , and  $\bar{\square}$ ) is – in a technical sense – “equivalent” to the logical structure of words with the concatenation operator. Makanin [20] showed that reasoning about a single equation over this structure (a.k.a. *word equations*) is decidable. More complex problems are known to be reducible to this basic case in polynomial-time, *e.g.* the existential theory over the structure [8]. Accordingly, we deduce that the existential theory over  $\mathcal{S}$  is decidable in polynomial space but NP-hard, whereas the first-order theory over  $\mathcal{S}$  is undecidable.

Finally (§5), we consider restrictions on  $\bowtie$  that admit a decidable theory. We define the family of one-argument functions indexed by tree constants that applies bowtie on the right-hand side  $\bowtie_{\square}$ , *i.e.*  $\bowtie_{\tau}(\tau') \triangleq \tau' \bowtie \tau$ . We prove that the combined theory of  $\mathcal{T} \triangleq (\mathbb{T}, \sqcap, \sqcup, \bar{\square}, \bowtie_{\square})$  has an embedding into *tree-automatic structures*. Since the first-order theory of tree-automatic structures is decidable [6], we obtain the decidability of the first-order theory of this extension of the tree share model with  $\bowtie_{\square}$ . This suggests the potential application of powerful heuristics for automata (*e.g.* antichain and simulation [1]) for providing a practical decision procedure for the tree share model.



**2 Formal preliminaries: the Tree Share model  $\mathbb{T}$  of Dockins *et al.* [10]**

Here we summarize additional details of tree shares and their associated theory from [10].



**Canonical forms.** In the first paragraph of §1 we presented the first quarter share as  instead of *e.g.* . This is deliberate: the second choice is not a valid share because the tree is not in *canonical form*. A tree is in canonical form when it is in its most compact representation under the inductively-defined equivalence relation  $\cong$ :

$$\begin{array}{ccccccc} \overline{o \cong o} & \overline{\bullet \cong \bullet} & \overline{o \cong \begin{array}{c} \diagup \diagdown \\ o \quad o \end{array}} & \overline{\bullet \cong \begin{array}{c} \diagup \diagdown \\ \bullet \quad \bullet \end{array}} & \frac{\tau_1 \cong \tau'_1 \quad \tau_2 \cong \tau'_2}{\begin{array}{c} \diagup \diagdown \\ \tau_1 \quad \tau_2 \end{array} \cong \begin{array}{c} \diagup \diagdown \\ \tau'_1 \quad \tau'_2 \end{array}} \end{array}$$







As we will see, operations on tree shares sometimes need to fold/unfold trees to/from canonical form, a practice we will indicate using the symbol  $\cong$ . Canonicity is needed to guarantee some of the algebraic properties of tree shares; managing it requires a little care in the proofs but does not pose any fundamental difficulties to the overall theory.

**Boolean algebra operations.** The connectives  $\sqcup$  and  $\sqcap$  first unfold both trees to the same shape; then calculate leafwise using the rules  $o \sqcup \tau = \tau \sqcup o = \tau$ ,  $\bullet \sqcup \tau = \tau \sqcup \bullet = \bullet$ ,  $o \sqcap \tau = \tau \sqcap o = o$ , and  $\bullet \sqcap \tau = \tau \sqcap \bullet = \tau$ ; and finally refold back into canonical form, *e.g.*:

$$\begin{array}{ccccccc} \begin{array}{c} \diagup \diagdown \\ o \quad o \end{array} \sqcup \begin{array}{c} \diagup \diagdown \\ o \quad \bullet \end{array} \cong \begin{array}{c} \diagup \diagdown \\ o \quad o \end{array} \sqcup \begin{array}{c} \diagup \diagdown \\ o \quad \bullet \end{array} = \begin{array}{c} \diagup \diagdown \\ \bullet \quad \bullet \end{array} \cong \begin{array}{c} \diagup \diagdown \\ \bullet \quad \bullet \end{array} \\ \begin{array}{c} \diagup \diagdown \\ o \quad o \end{array} \sqcap \begin{array}{c} \diagup \diagdown \\ o \quad \bullet \end{array} \cong \begin{array}{c} \diagup \diagdown \\ o \quad o \end{array} \sqcap \begin{array}{c} \diagup \diagdown \\ o \quad \bullet \end{array} = \begin{array}{c} \diagup \diagdown \\ o \quad o \end{array} \cong o \end{array}$$

Complementation is simpler, since flipping leaves between  $o$  and  $\bullet$  does not affect whether a tree is in canonical form, *e.g.*:  = . Using these definitions we get all of

the usual properties for Boolean algebras, *e.g.*  $\overline{\tau_1 \sqcap \tau_2} = \overline{\tau_1} \sqcup \overline{\tau_2}$ . Moreover, we can define a partial ordering between trees using intersection in the usual way, *i.e.*  $\tau_1 \sqsubseteq \tau_2 \triangleq \tau_1 \sqcap \tau_2 = \tau_1$ . We can enjoy a strict partial order as well:  $\tau_1 \sqsubset \tau_2 \triangleq \tau_1 \sqsubseteq \tau_2 \wedge \tau_1 \neq \tau_2$ .

**Properties of tree multiplication  $\bowtie$ .** Since it is nonstandard, the “tree multiplication” operator  $\bowtie$  deserves some additional attention. The good news first:  $\bowtie$  is associative, has an identity  $\bullet$ , and is injective for non- $o$  elements, *i.e.*  $\mathcal{S}^+ \triangleq (\mathbb{T} \setminus \{o\}, \bowtie)$  forms a cancellative monoid. Somewhat unsurprisingly, multiplication by the “additive identity”  $o$  reduces to  $o$ . Unfortunately,  $\bowtie$  is not commutative (  $\bowtie$   =   $\neq$   =   $\bowtie$  ),

although we do enjoy a distributive property over  $\sqcup$  and  $\sqcap$  on the right hand side. Accordingly:

► **Lemma 1** (Properties of  $\bowtie$ ).

- Associativity :  $\tau_1 \bowtie (\tau_2 \bowtie \tau_3) = (\tau_1 \bowtie \tau_2) \bowtie \tau_3$  (1)
- Identity element :  $\tau \bowtie \bullet = \bullet \bowtie \tau = \tau$  (2)
- Zero element :  $\tau \bowtie o = o \bowtie \tau = o$  (3)
- Left cancellation :  $\tau \neq o \Rightarrow \tau \bowtie \tau_1 = \tau \bowtie \tau_2 \Rightarrow \tau_1 = \tau_2$  (4)
- Right cancellation :  $\tau \neq o \Rightarrow \tau_1 \bowtie \tau = \tau_2 \bowtie \tau \Rightarrow \tau_1 = \tau_2$  (5)
- (6)

**Typical use of  $\mathbb{T}$  in program verification.** A standard way to use fractional shares in program verification is by modifying the standard maps-to predicate of separation logic to take a share as an additional argument. The predicate  $x \overset{\pi}{\mapsto} y$  then means that the heap has a cell at address  $x$ , which is owned with nonempty fraction  $\pi \neq \circ$  and whose value is  $y$ . We use  $\pi$  here because we often use share variables rather than concrete trees  $\tau$ .

To combine divided fractional ownership stakes back together it is traditional to use the “join” relation, written  $\tau_1 \oplus \tau_2 = \tau_3$ . The join relation is defined in turn using the primitive Boolean algebra operators:  $\tau_1 \oplus \tau_2 = \tau_3 \triangleq \tau_1 \sqcup \tau_2 = \tau_3 \wedge \tau_1 \sqcap \tau_2 = \circ$ . In other words, the join relation is a kind of disjoint union; it is partial because *e.g.*  $\bullet \oplus \bullet$  is undefined. Critically for verification  $\oplus$  **does** satisfy the disjointness axiom:  $\forall x, y. x \oplus x = y \Rightarrow x = y = \circ$ . Using  $\oplus$  we can state the following relationship between the spatial conjunction  $*$  and the underlying Boolean operators as  $x \overset{\pi_1}{\mapsto} y * x \overset{\pi_2}{\mapsto} z \dashv\vdash y = z \wedge x \overset{\pi_1 \oplus \pi_2}{\mapsto} y$  ( using  $\dashv\vdash$  for b entailment).

It is common that we want to “split” a share  $\pi$  into sub-shares  $\pi_1, \pi_2$  so that the permission can be transferred. This can be done within  $\mathcal{M} \triangleq (\sqcap, \sqcup, \bar{\square}, \circ, \bullet)$  using the following rule:

$$\frac{\pi \neq \circ}{x \overset{\pi}{\mapsto} v \dashv\vdash \exists \pi_1, \pi_2. (x \overset{\pi_1}{\mapsto} v * x \overset{\pi_2}{\mapsto} v) \wedge \pi_1 \oplus \pi_2 = \pi \wedge \pi_1 \neq \circ \wedge \pi_2 \neq \circ} \text{ SPLITJOIN}$$

This rule has some drawbacks. The most obvious is the lengthy size of the entailment’s consequent, even though we only split  $\pi$  into two pieces. Second, existential quantifiers are expensive in program verification since they tend to increase the size of the proof obligations and here we introduce two of them. Third, we have no control over what the shares  $\pi_1$  and  $\pi_2$  are – that is,  $\pi_1$  and  $\pi_2$  are not uniquely determined. Moreover, they are indistinguishable, which makes it difficult to assign different permitted actions for them.

On the other hand, each of these issues can be solved nicely using  $\bowtie$  due to its right distributivity over  $(\sqcap, \sqcup)$ , and thus over  $\oplus$ , yielding the following rules:

$$\frac{\tau_1 \oplus \dots \oplus \tau_n = \tau \quad \pi \neq \circ \quad \bigwedge_{i=1}^n \tau_i \neq \circ}{x \overset{\pi \bowtie \tau}{\mapsto} v \dashv\vdash x \overset{\pi \bowtie \tau_1}{\mapsto} v * \dots * x \overset{\pi \bowtie \tau_n}{\mapsto} v} \text{ SPLITJOIN } \bowtie \quad (7)$$

### 3 Tree Shares are a model for Countable Atomless Boolean Algebras

In this section, we pinpoint the fact that  $\mathcal{M} = (\sqcap, \sqcup, \bar{\square}, \circ, \bullet)$  is a model for Countable Atomless Boolean Algebra (CABA). Let  $\mathcal{B} = (\cap, \cup, \bar{\square}, \mathbf{0}, \mathbf{1})$  be a Boolean Algebra (BA), we define a partial order  $\subseteq$  on  $\mathcal{B}$  ( $\sqsubseteq$  for  $\mathcal{M}$ , resp.):  $a_1 \subseteq a_2 \triangleq a_1 \cap \bar{a}_2 = \mathbf{0}$  and  $a_1 \subset a_2 \triangleq a_1 \subseteq a_2 \wedge a_2 \not\subseteq a_1$ .  $\mathcal{B}$  is *atomless* if  $\forall a. \mathbf{0} \subset a \Rightarrow \exists a'. \mathbf{0} \subset a' \subset a$ .  $\mathcal{B}$  is *countable* if its domain is countable. Dockins *et al.* [10] proved that  $\mathcal{M}$  is a model for BA where  $\mathbf{0} \triangleq \circ$ ,  $\mathbf{1} \triangleq \bullet$ ,  $\sqcup \triangleq \cup$ ,  $\sqcap \triangleq \cap$ . The atomless property can be derived from the Infinite Splitability property of tree shares [19]: let  $a \sqsupset \circ$  and  $a_1, a_2 \neq \circ$  such that  $a_1 \oplus a_2 = a$ . This implies  $a_1 \sqcup a_2 = a \wedge a_1 \sqcap a_2 = \circ$ . By Stone’s representation theorem each BA is isomorphic to a BA of powerset, thus  $a_1 \sqcup a_2 = a$  implies  $a_1 \subseteq a$  and  $a_2 \subseteq a$ . Suppose that  $a_1 = a$  then  $a_2 \sqcap a = \mathbf{0}$  which is a contradiction because  $\mathbf{0} \sqsupset a_2 \sqsupset a$ . As a result,  $a_1 \neq a$  and thus  $a_1 \subset a$ . The proof that  $\mathbb{T}$  is countable is achieved by enumerating  $\mathbb{T}$  in the ascending order of tree height  $|\tau|$  using the following total strict order  $\prec$ :

$$\frac{}{\circ \prec \bullet} \quad \frac{|\tau_1| < |\tau_2|}{\tau_1 \prec \tau_2} \quad \frac{|\tau_1 \frown \tau'_1| = |\tau_2 \frown \tau'_2| \quad \tau_1 \prec \tau_2}{\tau_1 \frown \tau'_1 \prec \tau_2 \frown \tau'_2} \quad \frac{|\tau \frown \tau_1| = |\tau \frown \tau_2| \quad \tau_1 \prec \tau_2}{\tau \frown \tau_1 \prec \tau \frown \tau_2}$$

It is known that there is a unique model for CABA up to isomorphism [28], so we can reason about the complexity of  $\mathcal{M}$  in terms of CABA. Let  $\text{STA}(f(n), g(n), h(n))$  be the class of sets accepted by alternating Turing machines which use at most  $f(n)$  space,  $g(n)$  time and  $h(n)$  alternations between universal and existential states on a given input of length  $n$ . Any field in the description can be replaced with symbol  $*$  to indicate no bound is required. Kozen [17] proved that the elementary theory of infinite BAs is  $\leq_{\log}$ -complete for the Berman complexity class  $\bigcup_{c < \omega} \text{STA}(*, 2^{cn}, n)$ , which lies between the class of deterministic exponential space and non-deterministic exponential space.

We now investigate the complexity of an important sub-theory of  $\mathcal{M}$ , namely the existential theory. Basically, this sub-theory includes all valid sentences whose prenex normal form contains only existential quantifiers. Its counterpart is the universal theory in which all the quantifiers are universal. A result by Marriott *et al.* [24] showed that the existential theory and universal theory for infinite BAs are in NP-complete and co-NP-complete respectively.

#### 4 Decidability of general multiplication $\bowtie$ over Tree Shares

In this section, we will prove the following results about  $\mathcal{S} = (\mathbb{T}, \bowtie)$ :

► **Theorem 2** (Complexity of  $\mathcal{S}$ ).

1. *The existential theory of  $\mathcal{S}$  is decidable in PSPACE.*
2. *The existential theory of  $\mathcal{S}$  is NP-hard.*
3. *The general first-order theory over  $\mathcal{S}$  is undecidable.*

The proof of Theorem 2 largely rests on the identical conclusions for the key subtheory  $\mathcal{S}^+ \triangleq (\mathbb{T}^+, \bowtie)$ , where  $\mathbb{T}^+ \triangleq \mathbb{T} \setminus \{\circ\}$  are the “positive trees” obtained by removing the “zero element”  $\circ$  from  $\mathbb{T}$ :

► **Lemma 3** (Complexity of  $\mathcal{S}^+$ ).

1. *The existential theory of  $\mathcal{S}^+$  is decidable in PSPACE.*
2. *The existential theory of  $\mathcal{S}^+$  is NP-hard.*
3. *The general first-order theory over  $\mathcal{S}^+$  is undecidable.*

We will prove Lemma 3 shortly, but first let us use it to polish off Theorem 2:

**Proof of Theorem 2.** We take each part in turn as follows:

1. Represent the set of variables  $V = \{x_1, \dots, x_n\}$  in a given formula  $F$  of  $\mathcal{S}$  as a  $n$ -length bitvector. We can enumerate through all possibilities  $P_1, \dots, P_{2^n}$  for this vector using linear space and binary addition. For each possibility  $P_j$ , variable  $x_i$ 's bit is 0 to indicate that  $x_i$  must be  $\circ$  and 1 when  $x_i$  must be non- $\circ$ . For each  $x_k$  that is marked as  $\circ$ , we substitute  $\circ$  for  $x_k$  in  $F$  to reach  $F_j$  and simplify using the rules

$$\begin{array}{ccc} \frac{\pi_1 \bowtie \circ = \pi_2}{\pi_2 = \circ} & \frac{\circ \bowtie \pi_1 = \pi_2}{\pi_2 = \circ} & \frac{\pi_1 \bowtie \pi_2 = \circ}{\pi_1 = \circ \vee \pi_2 = \circ} \\ \frac{\pi_1 \bowtie \circ \neq \pi_2}{\pi_2 \neq \circ} & \frac{\circ \bowtie \pi_1 \neq \pi_2}{\pi_2 \neq \circ} & \frac{\pi_1 \bowtie \pi_2 \neq \circ}{\pi_1 \neq \circ \wedge \pi_2 \neq \circ} \end{array}$$

We can then just check to make sure that the resulting “fresh” (in)equalities are consistent with the current value of the bitvector  $P_j$ . If not, we have reached a contradiction and can proceed to the next bitvector  $P_{j+1}$ . If so, then after removing the trivial equalities (*e.g.*  $\circ = \circ$ ) from  $F_j$  we are left with an equivalent formula  $F_j^+$  which is in  $\mathcal{S}^+$ , so

- by Lemma 3.1 we can check if  $F_j$  is satisfiable in PSPACE. If so, we know that  $F_j$  is satisfiable, and thus that  $F$  is satisfiable. If not, we proceed to the next bitvector  $P_{j+1}$ ; if all  $F_j$  are unsatisfiable then  $F$  is unsatisfiable.
2. By Lemma 3.2 it is sufficient to reduce a formula  $F^+$  in  $\mathcal{S}^+$  to  $\mathcal{S}$ . Let  $V$  be the set of variables in  $F^+$  and define  $F \triangleq F^+ \wedge \left( \bigwedge_{x \in V} x \neq \circ \right)$ ; note that we construct  $F$  in linear time from  $|F^+|$ .  $F$  is satisfiable in  $\mathcal{S}$  if and only if  $F^+$  is satisfiable in  $\mathcal{S}^+$ , so we are done.
  3. Any extension of an undecidable theory is also undecidable; by Lemma 3.3 we are done.  $\blacktriangleleft$

#### 4.1 Word equations

To prove Lemma 3 we will show that  $\mathcal{S}^+$  is isomorphic to the theory of word equations. Let us recall this theory. Let  $A = \{a_1, a_2, \dots\}$  be a finite set of letters and  $\cdot$  be a concatenation operator that combines letters into words. Let  $A^*$  be the Kleene closure of  $A$  using  $\cdot$ . We define a model for the alphabet  $A$ , written  $\mathcal{W}_A$  as the pair  $(A^*, \cdot)$ . Now let  $V = \{v_1, v_2, \dots\}$  be a finite set of variables, and  $w \in \mathbb{W} \triangleq (A \cup V)^*$  a finitely generated word that includes both letters and variables. We extend a *word context*  $\rho : V \rightarrow A^*$  to the domain  $A \cup V$  by mapping constants to themselves, and further to the domain  $\mathbb{W}$  by replacing each letter within a word with its value in  $\rho$ . A *word equation*  $E_{\mathbb{W}}$  is a pair of words  $(w_1, w_2) \in \mathbb{W} \times \mathbb{W}$ . We say that  $\rho$  is a solution of  $E_{\mathbb{W}}$  if  $\rho(w_1) = \rho(w_2)$ .

The satisfiability of word equation asks whether a word equation  $E_{\mathbb{W}}$  has a solution  $\rho$ , denoted  $\mathbf{SAT}_{\mathbb{W}}(E_{\mathbb{W}})$ . Makanin proposed a complete treatment to this problem in a series of papers [20, 21, 22] but his method was highly intractable (quadruple-exponential non-deterministic time [16]). Substantial research since has improved this bound, *e.g.* [2, 13]. The best known complexity bound for this problem is PSPACE and NP-hard [25, 26] and it is hypothesized to be NP-complete. Importantly for our present result, the existential theory over word equations is known to be reducible to  $\mathbf{SAT}_{\mathbb{W}}$  in polynomial time [8]. Finally, the first order theory over  $\mathbb{W}$  is known to be undecidable [23, 18].

**Infinite alphabets.** To define our isomorphism from  $\mathbb{T}^+$  to  $A^*$  it will be convenient if the alphabet  $A$  can be countably infinite. Accordingly, we must reduce word equations over an infinite alphabet to the standard finite case. Let  $\sigma : \mathbb{W} \rightarrow \mathcal{P}(A)$  be the function that extracts the set of letters from a word  $w$ , *e.g.*  $\sigma(v_1 a_1 a_3 v_2) = \{a_1, a_3\}$  and extend  $\sigma$  to  $\mathbb{W} \times \mathbb{W}$  by  $\sigma(w_1, w_2) = \sigma(w_1) \cup \sigma(w_2)$ . Let  $\phi : \mathbb{W} \times \mathcal{P}(A) \rightarrow \mathbb{W}$  be the projection function that takes a word  $w$  and a set of letters  $B \subseteq A$  and removes all letters in  $w$  that are not in  $B$ , *e.g.*  $\phi(v_1 a_1 a_3 v_2, \{a_1, a_2\}) = v_1 a_1 v_2$ . It is not hard to prove that  $\phi$  with fixed  $B$  is an homomorphism over  $\mathcal{W}_A$ . Now we are ready to state and prove the extension to infinite alphabets:


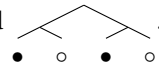
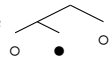


► **Lemma 4** (Infinite alphabet word equations). *Let  $A$  be infinite and  $E_{\mathbb{W}} = (w_1, w_2)$  a word equation over  $A$ .  $E_{\mathbb{W}}$  is satisfiable in  $\mathcal{W}_A$  iff  $E_{\mathbb{W}}$  is satisfiable in  $\mathcal{W}_{\sigma(E_{\mathbb{W}})}$ .*

**Proof.**  $\Leftarrow$  is trivial. Let  $\rho : V \rightarrow A^*$  be a solution of  $E_{\mathbb{W}}$  over  $A$  and  $\rho' = \lambda v. \phi(\rho(v), \sigma(E_{\mathbb{W}}))$ . Notice  $\rho'$  preserves all the letters in  $E_{\mathbb{W}}$  and  $\rho(w_1) = \rho(w_2)$  implies  $\rho'(w_1) = \rho'(w_2)$ . Thus  $\rho'$  is a new solution of  $E_{\mathbb{W}}$  that only contains letters from  $\sigma(E_{\mathbb{W}})$ .  $\blacktriangleleft$

#### 4.2 Finding an infinite alphabet inside $\mathbb{T}^+$

Since  $\bowtie$  is a kind of multiplication operation, and the fundamental building blocks of  $(\mathbb{N}, \times)$  are prime numbers, it is natural to wonder whether there is an analogue on trees. There is:

► **Definition 5** (Prime trees).  $\tau \in \mathbb{T}^+ \setminus \{\bullet\}$  is *prime* if  $\forall \tau_1, \tau_2. \tau = \tau_1 \bowtie \tau_2 \Rightarrow (\tau_1 = \bullet \vee \tau_2 = \bullet)$ . Furthermore, let  $\text{Prime}(\tau)$  indicate  $\tau$  is prime and  $\mathbb{T}_p$  be the set of all prime trees.

Examples of tree primes are  and . On the other hand, the tree  is not prime since it can be factored as   $\bowtie$  . Prime trees have many nice properties:

► **Lemma 6** (Properties of prime trees).

1. There are countably infinitely many prime trees.
2. Let  $\tau_1, \tau'_1, \tau_2, \tau'_2 \in \mathbb{T}_p$ ,  $\tau_1 \bowtie \tau_2 = \tau'_1 \bowtie \tau'_2$  iff  $\tau_1 = \tau'_1$  and  $\tau_2 = \tau'_2$ .
3. Given two prime tree sequences  $S_1 = \tau_1^{k_1}, \dots, \tau_1^{k_1}$  and  $S_2 = \tau_2^{k_2}, \dots, \tau_2^{k_2}$ ,  $S_1 = S_2$  iff their  $\bowtie$  products are equal:  $\bowtie_{i=1}^{k_1} \tau_1^i = \bowtie_{i=1}^{k_2} \tau_2^i \Leftrightarrow (k_1 = k_2 \wedge \bigwedge_{i=1}^{k_1} \tau_1^i = \tau_2^i)$ .

To prove Lemma 6 we must define the notation  $|\tau|$  to be the height of  $\tau$  (with  $|\circ| = |\bullet| = 0$  and counting up from there). Given this notation it is simple to define the set of all trees up to height  $n$ , written  $\mathbb{T}^n$ . We will also need the following technical lemma which allows us to split an application of bowtie  $\tau_2 \bowtie \tau_3$  to children of  $\tau_2$ :

► **Lemma 7** (Split for  $\bowtie$ ). Let  $\tau_1, \tau_2, \tau_3, \tau_1^l, \tau_1^r \in \mathbb{T}^+$  and  $\tau_1 = \tau_2 \bowtie \tau_3 \wedge \tau_1 = \widehat{\tau_1^l \tau_1^r}$  then either (1)  $\tau_2 = \bullet \wedge \tau_1 = \tau_3$  or (2)  $\exists \tau_2^l, \tau_2^r. \tau_2 = \widehat{\tau_2^l \tau_2^r} \wedge \tau_1^l = \tau_2^l \bowtie \tau_3 \wedge \tau_1^r = \tau_2^r \bowtie \tau_3$ .

**Proof.** The case  $\tau_2 = \bullet$  is trivial. Otherwise, there exists  $\tau_2^l, \tau_2^r \in \mathbb{T}$  such that  $\tau_2 = \widehat{\tau_2^l \tau_2^r}$ . By definition of  $\bowtie$ ,  $\tau_1 = \tau_2 \bowtie \tau_3$  is computed by replacing each leaf  $\bullet$  in  $\tau_2$  with  $\tau_3$ , which is equivalent to replace each leaf  $\bullet$  in  $\tau_2^l$  and  $\tau_2^r$  with  $\tau_3$ . Thus,  $\tau_1^l = \tau_2^l \bowtie \tau_3$  and  $\tau_1^r = \tau_2^r \bowtie \tau_3$ . ◀

**Proof of Lemma 6.**

1. We construct an infinite sequence  $S$  of prime trees: let  $p_1 \triangleq \widehat{\bullet \circ}$ ,  $p_j \triangleq \widehat{p_{j-1} \bullet}$ , i.e.

$$S \triangleq \left\{ \widehat{\bullet \circ}, \widehat{\bullet \circ \bullet}, \widehat{\bullet \circ \bullet \circ}, \dots \right\}$$

It is immediate that  $p_1$  is prime. To prove that  $p_i$  is prime for  $i > 1$ , we proceed as follows. Suppose  $p_i = \tau_1 \bowtie \tau_2$  and neither  $\tau_1$  nor  $\tau_2$  is  $\bullet$ . The right subtree of each  $p_i$  is just  $\bullet$  and by the definition of  $\bowtie$  must contain a copy of  $\tau_2$ , i.e.  $\tau_2 = \bullet$ , so we have a contradiction and  $p_i$  is prime.

2. We prove by induction on the height of  $\tau_1, \tau'_1$ . The base case  $\mathbb{T}^0$  is easy to verify. Assume it holds for  $\mathbb{T}^k$  and  $\tau_1, \tau'_1 \in \mathbb{T}^{k+1}$ . Let  $\tau_1 = \widehat{\tau_1^l \tau_1^r}, \tau'_1 = \widehat{\tau_1'^l \tau_1'^r}$  then by Lemma 7, we derive  $\tau_1^l \bowtie \tau_2 = \tau_1'^l \bowtie \tau_2, \tau_1^r \bowtie \tau_2 = \tau_1'^r \bowtie \tau_2$ . By our induction hypothesis,  $\tau_1^l = \tau_1'^l, \tau_1^r = \tau_1'^r, \tau_2 = \tau_2$ . Consequently,  $\tau_1 = \tau'_1$ .
3. This is a simple generalization of property 2. ◀

Of course the real fun with prime numbers is the the unique factorization theorem. Since  $\bowtie$  is not commutative we get a stronger version of the traditional theorem:

► **Lemma 8** (Unique representation of  $\mathcal{S}^+$ ). For each  $\tau \in \mathbb{T}^+ \setminus \{\bullet\}$ , there exists a unique sequence  $\tau_1, \dots, \tau_n \in \mathbb{T}_p$  such that  $\tau = \bowtie_{i=1}^n \tau_i$ . Each  $\tau_i$  is called a *prime factor* of  $\tau$ .

**Proof.** We prove by induction on the height of  $\tau$ . The base case  $\mathbb{T}^1$  is trivial. Assume it holds for  $\mathbb{T}^k$  and let  $\tau \in \mathbb{T}^{k+1}$ . If  $\tau$  is prime then we are done. Otherwise, let  $\tau^1, \tau^2 \in \mathbb{T}^k \setminus \{\bullet\}$  and  $\tau = \tau^1 \bowtie \tau^2$ . By our induction hypothesis, there are 2 sequences  $\tau_1^1, \dots, \tau_{k_1}^1 \in \mathbb{T}_p$  and  $\tau_1^2, \dots, \tau_{k_2}^2 \in \mathbb{T}_p$  such that  $\tau^1 = \bowtie_{i=1}^{k_1} \tau_i^1$  and  $\tau^2 = \bowtie_{i=1}^{k_2} \tau_i^2$  and thus  $\tau = (\bowtie_{i=1}^{k_1} \tau_i^1) \bowtie (\bowtie_{i=1}^{k_2} \tau_i^2)$ . The uniqueness is a consequence of property 3 from Lemma 6.  $\blacktriangleleft$

► **Corollary 9** (Basis of  $\mathcal{S}^+$ ).  $\mathbb{T}_p \cup \{\bullet\}$  is a basis of  $\mathcal{S}^+$ , i.e. the closure of  $\mathbb{T}_p$  over  $\bowtie$  together with  $\bullet$  is  $\mathbb{T}^+$ . Furthermore, it is the smallest basis: if  $B$  is a basis of  $\mathcal{S}^+$  then  $\mathbb{T}_p \cup \{\bullet\} \subseteq B$ .

Accordingly, we will use  $\mathbb{T}_p$  as our “infinite alphabet” in our isomorphism.

### 4.3 Connecting Tree Shares to Word Equations

We are ready to make the central connection needed for Lemma 3:

► **Lemma 10.**  $(\mathbb{T}^+, \bowtie)$  is isomorphic to  $(\mathbb{T}_p^*, \cdot)$

**Proof.** Let  $f : \mathbb{T}^+ \rightarrow \mathbb{T}_p^*$  be defined as follows. First, map the identity element  $\bullet$  to the empty word  $\epsilon$  and then for each prime tree  $\tau_p \in \mathbb{T}^+$  map  $\tau_p$  to itself. Finally, for each composite  $\tau \in \mathbb{T}^+$  map  $\tau$  to exactly the concatenation of its (unique) prime factors.

We now wish to prove that for any  $\tau_1$  and  $\tau_2$ ,  $f(\tau_1 \bowtie \tau_2) = f(\tau_1) \cdot f(\tau_2)$ . Let us consider the easy cases first. If  $\tau_1 = \bullet$  then  $f(\tau_1 \bowtie \tau_2) = f(\tau_2) = \epsilon \cdot f(\tau_2) = f(\tau_1) \cdot f(\tau_2)$ . The situation is symmetric when  $\tau_2 = \bullet$ . Now let us consider the case when neither  $\tau_1$  nor  $\tau_2$  is  $\bullet$ . Let  $p_1, \dots, p_i$  be the unique prime factors of  $\tau_1$  and  $p'_1, \dots, p'_j$  be the unique prime factors of  $\tau_2$ . By Lemma 8,  $p_1, \dots, p_i, p'_1, \dots, p'_j$  are exactly the unique prime factors of  $\tau_1 \bowtie \tau_2$ , so:

$$f(\tau_1 \bowtie \tau_2) = f(p_1 \bowtie \dots \bowtie p_i \bowtie p'_1 \bowtie \dots \bowtie p'_j) = p_1 \cdot \dots \cdot p_i \cdot p'_1 \cdot \dots \cdot p'_j = (p_1 \cdot \dots \cdot p_i) \cdot (p'_1 \cdot \dots \cdot p'_j) = f(\tau_1) \cdot f(\tau_2)$$

To prove  $f$  is surjective, let  $w \in \mathbb{T}_p^*$  be the concatenation of primes  $p_1 \cdot \dots \cdot p_i$ ; then by the definition  $f(p_1 \bowtie \dots \bowtie p_i) = w$ . To prove  $f$  is injective, suppose  $f(\tau_1) = f(\tau_2)$ . Let  $p_1, \dots, p_i$  be the prime factors of  $\tau_1$  and  $p'_1, \dots, p'_j$  be the prime factors of  $\tau_2$ . Accordingly we know that  $p_1 \cdot \dots \cdot p_i = p'_1 \cdot \dots \cdot p'_j$ , and since equality over words can only occur if the words have the same length and have the same letters, we know  $i = j$  and  $p_k = p'_k$  for all  $k$ .  $\blacktriangleleft$

► **Corollary 11** (Equations over Positive Tree Shares are Word Equations). Equations  $E_{\mathbb{T}^+}$  over  $(\mathbb{T}^+, \bowtie)$  contain both tree constants  $\tau \in \mathbb{T}^+$  and variables  $v \in V$ ; we can map these to word equations  $E_{\mathbb{W}}$  over  $(\mathbb{T}_p^*, \cdot)$  by mapping variables to themselves, constants to the concatenation of their prime factors, and multiplication  $\bowtie$  to concatenation  $\cdot$ . The resulting system is equivalent, i.e. if  $\rho : V \rightarrow \mathbb{T}^+$  satisfies  $E_{\mathbb{T}^+}$  then  $f \circ \rho$  satisfies  $E_{\mathbb{W}}$ , where  $\circ$  in this case means functional composition and  $f$  is the isomorphism constructed in Lemma 10.

We are now ready to start tackling Lemma 3. We start with the simplest:

**Proof of Lemma 3.3.** As previously mentioned, the first order theory over word equations is known to be undecidable [23, 18]. By Lemma 10 we know that this theory is isomorphic to the first order theory over tree shares with  $\bowtie$ , which accordingly must be undecidable.  $\blacktriangleleft$

To show Lemma 3.1 we need to know that tree factorization can be done within PSPACE. In fact we can do much better:

► **Lemma 12** (Factorization). Factoring an arbitrary positive tree share  $\tau$  is in P.

**Proof.** Let  $\mathbb{S}(\tau)$  be the set of all subtrees of  $\tau$  and  $\mathbb{S}_n(\tau) \subset \mathbb{S}(\tau)$  be the set of all subtrees of  $\tau$  with height exactly  $n$ .  $\mathbb{S}(\tau)$  can be computed recursively:  $\mathbb{S}(\circ) = \{\circ\}$ ,  $\mathbb{S}(\bullet) = \{\bullet\}$ ,  $\mathbb{S}(\begin{smallmatrix} \diagup \\ \tau_1 \end{smallmatrix} \begin{smallmatrix} \diagdown \\ \tau_2 \end{smallmatrix}) = \mathbb{S}(\tau_1) \cup \mathbb{S}(\tau_2) \cup \{\begin{smallmatrix} \diagup \\ \tau_1 \end{smallmatrix} \begin{smallmatrix} \diagdown \\ \tau_2 \end{smallmatrix}\}$ . If  $\tau = \tau_1 \bowtie \tau_2$  ( $\{\tau_1, \tau_2\} \subset \mathbb{T}^+ \setminus \{\bullet\}$ ), then there exists  $n \in \mathbb{N}$  such that  $\mathbb{S}_n(\tau) = \{\tau_2\}$ , that is,  $\mathbb{S}_{|\tau_2|}(\tau)$  is exactly the singleton set  $\{\tau_2\}$ . Additionally,  $\mathbb{S}(\tau) = \bigcup_{i=0}^{|\tau|} \mathbb{S}_i(\tau)$ .

Thus we can find all the prime factors of  $\tau$  (which is inspired from the well-known sieve of Eratosthenes) as follows: first we compute  $\mathbb{S}(\tau)$  and partition it into  $\mathbb{S}_0(\tau), \dots, \mathbb{S}_{|\tau|}(\tau)$ . Let  $i \in \mathbb{N}$  be the smallest number such that  $\mathbb{S}_i(\tau)$  is the singleton set  $\{\tau_1\}$  for some  $\tau_1 \in \mathbb{T}$  (note that  $i$  must be larger than 0 since  $\mathbb{S}_0(\tau) = \{\circ, \bullet\}$ ). If  $i = |\tau|$  then  $\tau$  itself is a prime, otherwise, we replace all subtrees  $\tau_1$  of  $\tau$  with  $\bullet$  and call the new tree  $\tau'$ . If all the “old”  $\bullet$  leaves of  $\tau$  are replaced and  $\tau'$  is in canonical form then  $\tau = \tau' \bowtie \tau_1$ ,  $\tau_1$  is a prime factor of  $\tau$ , and we can repeat the process with  $\tau'$  to find the next prime factor. Otherwise, we consider the next singleton set  $\mathbb{S}_j(\tau)$ .

If  $\tau$  has  $n$  leaves then its description requires  $O(n)$  bits and the time to compute  $\mathbb{S}(\tau)$  is  $O(n)$ . Note that  $|\mathbb{S}_k(\tau)| \leq \frac{n}{k+1}$  because there are at least  $k+1$  leaves in a tree of height  $k$ . Therefore, the number of subtrees from height 1 to  $n$  is at most  $\sum_{i=1}^n \frac{n}{i+1} \leq n^2$ . Computing the height of a subtree  $\tau'$  of  $\tau$  requires  $O(n)$ , thus the time to partition  $\mathbb{S}(\tau)$  is  $O(n^3)$ . The number of times we need to restart the process is  $O(n^2)$ . Consequently, the time for tree factorization is  $O(n^5)$ , polynomial in the description of  $\tau$  (more efficient solutions exist). ◀

Tree factorization is fundamentally simpler than integer factorization since the representation of a tree already contains the descriptions of all of its tree factors. In contrast, the connection between the representation of a number and the representation of its prime factors is vague: *e.g.* among the 24 factors of 74,611,647 are 333 (which does not appear at all in the representation of the original) and 8,290,183 (which only shares a single 1 with the original).

**Proof of Lemma 3.1.** We take the tree shares and factor them using Lemma 12 and then construct the isomorphic system of word equations using the calculated prime factors as the alphabet using Corollary 11. As mentioned, the best known complexity bound for the existential word equation problem is PSPACE [25, 26]. ◀

For Lemma 3.2 we need one final fact:

► **Lemma 13 (Existence of small primes).** *For any  $n$  (represented in unary) we can find a length- $n$  sequence of tree primes  $S$  in polynomial time of  $n$ .*

**Proof.** Consider the sequence  $S$  from Lemma 6: the description of  $p_i$  is only a constant size larger than the description of  $p_{i-1}$  so the description of  $S$  is quadratic in  $n$ . ◀

**Proof of Lemma 3.2.** Suppose we have an arbitrary problem  $Q$  in NP. We can reduce  $Q$  to word equations in polynomial time [25, 26]. We then use Lemma 13 to construct a set of primes the size of the number of alphabet letters that appear in the equations and map each letter in the word alphabet to a distinct prime, creating a set of word equations over  $\mathbb{T}_p^*$ . Since the representation of the constants does not affect the computational properties of the theory, we can conclude that  $\mathbb{T}^+$  is NP-hard. ◀

## 5 A reduction to Tree Automatic Structures

Theorem 2 shows that the first order theory (FO) over  $\mathcal{S}$  is undecidable, so of course any extension of  $\mathcal{S}$  – *e.g.* with  $(\sqcap, \sqcup, \bar{\square})$  – also has an undecidable FO. However, if we restrict the



form of  $\bowtie$ -equations to be  $\pi_1 \bowtie \tau = \pi_2$  where  $\tau \in \mathbb{T}$ , then the FO of  $\mathcal{S}$  is decidable because the relation is tree-automatic. This type of restriction is inspired by Jain *et al.*'s concept of *semi-automatic structures* [14], in which relations are restricted so that all input arguments are fixed constants except for one argument which is a variable. As a result, certain relations become automatic, *e.g.* multiplication in unary language.

Let  $\tau \bowtie : \mathbb{T} \rightarrow \mathbb{T}$  and  $\bowtie_\tau : \mathbb{T} \rightarrow \mathbb{T}$  be the *left* and *right restricted form* of  $\bowtie$  with respect to  $\tau \in \mathbb{T}$ , *i.e.*  $\tau \bowtie(\tau_1) \triangleq \tau \bowtie \tau_1$  and  $\bowtie_\tau(\tau_1) \triangleq \tau_1 \bowtie \tau$ . We will show  $\mathcal{T} \triangleq (\mathbb{T}, \sqcap, \sqcup, \bar{\sqcap}, \bowtie_{\sqcap})$ , where  $\bowtie_{\sqcap}$  denotes the family of all right-restricted forms of  $\bowtie$  indexed by tree constants, is tree-automatic by constructing bottom-up tree automata that recognize the domain  $\mathbb{T}$  and the relations in  $\mathcal{T}$ :

► **Theorem 14** (Decidability of  $(\mathbb{T}, \sqcap, \sqcup, \bar{\sqcap}, \bowtie_{\sqcap})$ ).  *$\mathcal{T}$  is tree-automatic. As a result, the first-order theory of  $\mathcal{T}$  is decidable.*

As indicated by equation (7) (from §2), one use for  $\bowtie$  is to split/join ownership of maps-to predicates in separation logic. Here the splitting/joining occurs on the right-hand side of the  $\bowtie$ . Moreover, many functions need to divide their ownership only a finite number of times before *e.g.* calling other functions or indeed themselves recursively. This is because the program text of functions is finite. Accordingly, we believe that  $\mathcal{T}$  is worthy of attention.

## 5.1 Tree automatic structures

Before proving Theorem 14, we recall the definition of tree automatic structures [27, 15].

**Tree automaton.** A *bottom up tree automaton* is a 4-tuple  $\mathcal{A} = (Q, F, Q_f, \Delta)$  where  $Q$  is the *set of states*,  $F$  is the *ranked alphabet*,  $Q_f \subset Q$  is the *set of accepting states* and  $\Delta$  is the *set of transitions*  $f(q_1(v_1), \dots, q_n(v_n)) \rightarrow q_{n+1}(f(v_1, \dots, v_n))$  where  $f \in F$  is a  $n$ -ary letter,  $v_i \in V$  is a variable and  $q_j \in Q$ . Let  $T$  be a tree constructed from  $F$  then  $\mathcal{A}$  runs on  $T$  by applying  $\Delta$  at each leaf of  $T$  spontaneously and proceeding upward.  $\mathcal{A}$  accepts  $T$  if the state associated with the root of  $T$  is in  $Q_f$ . We will put a temporary superscript number ( $n$ ) above each letter in the definition of  $F$  to indicate its arity, *i.e.*  $f^{(n)}$  means  $f$  is  $n$ -ary. For instance, the ranked alphabet for the tree domain  $\mathbb{T}$  is  $\mathcal{F} = \{\circ^{(0)}, \bullet^{(0)}, \text{Node}^{(2)}\}$ .

**Tree automatic structures.** Let  $\mathcal{R} \subset \mathcal{D}^k$  be a  $k$ -ary tree relation on some tree domain  $\mathcal{D}$ , its *convolution set* is constructed by overlapping all  $k$  trees of each element in  $\mathcal{R}$  to form a single tree. As trees can have different shapes and thus their convolution contains “holes”, we fill the holes with a special nullary character  $\diamond^{(0)}$  that is not in  $F$ , *e.g.*  $(\text{Node}(a_1, a_2), b, \text{Node}(c_1, c_2)) \mapsto [\text{Node}, b, \text{Node}][[a_1, \diamond, c_1], [a_2, \diamond, c_2]]$ . As a result, if  $F_{\mathcal{D}}$  is the ranked alphabet of  $\mathcal{D}$  then  $(F_{\mathcal{D}} \cup \{\diamond\})^k$  is the ranked alphabet for  $\mathcal{D}^k$  and the arity of each convolution letter is the maximal arity among its letter components.  $\mathcal{R}$  is *tree-automatic* if its convolution set is accepted by a tree automaton. Generally, a structure  $(\mathcal{D}, \mathcal{R}_1, \dots, \mathcal{R}_n)$  is tree-automatic if its domain  $\mathcal{D}$  and each of its relations  $\mathcal{R}_i$  are tree-automatic. We restate a well-known decidability result for tree automatic structures:

► **Lemma 15** ([5, 6]). *The first-order theory of a tree automatic structures is decidable.*

## 5.2 Tree automata construction for $\mathcal{T}$

**Construction of  $\mathcal{A}^{\mathbb{T}}$ .** For the domain  $\mathbb{T}$ , it suffices to check the canonical form. Let  $\mathcal{A}^{\mathbb{T}} = (Q^{\mathbb{T}}, F^{\mathbb{T}}, Q_f^{\mathbb{T}}, \Delta^{\mathbb{T}})$  such that  $Q^{\mathbb{T}} = \{q, q_{\circ}, q_{\bullet}\}$ ,  $F^{\mathbb{T}} = \mathcal{F}$ ,  $Q_f^{\mathbb{T}} = \{q\}$  and the transition relation  $\Delta^{\mathbb{T}}$  contains the following:



- $\circ \mapsto q(\circ)$  and  $\bullet \mapsto q(\bullet)$
- $\text{Node}(q_1(v_1), q_2(v_2)) \mapsto q(\text{Node}(v_1, v_2))$  where  $(q_1, q_2) \in \{(q_\circ, q_\bullet), (q_\bullet, q_\circ), (q, q)\}$

**Construction of  $\mathcal{A}^{\square}$ .** From now on, we assume that the input trees are in domain  $\mathbb{T}$ , which will make other constructions more pleasant. The automaton  $\mathcal{A}^{\square}$  for complement  $\square$  is easy: we need to verify the opposite values leaf-wise between two trees. To be precise, let  $(Q^{\square}, F^{\square}, Q_f^{\square}, \Delta^{\square})$  such that  $Q^{\square} = Q_f^{\square} = \{q\}$ ,  $F^{\square} = (\mathcal{F} \cup \{\diamond\})^2$  and  $\Delta^{\square}$  is defined as:

- $[\bullet, \circ] \mapsto q([\bullet, \circ])$  and  $[\circ, \bullet] \mapsto q([\circ, \bullet])$
- $[\text{Node}, \text{Node}](q(v_1), q(v_2)) \mapsto q([\text{Node}, \text{Node}](v_1, v_2))$

**Construction of  $\mathcal{A}^{\sqcup}$  and  $\mathcal{A}^{\sqcap}$ .**  $\mathcal{A}^{\sqcup}$  and  $\mathcal{A}^{\sqcap}$  are more sophisticated as trees can have different shapes and thus are inapplicable for leaf-wise comparison. For instance, the convolution of the relation  $\widehat{\circ} \sqcup \bullet = \bullet$  is  $[\text{Node}, \bullet, \bullet](\circ, \diamond, \diamond), [\bullet, \diamond, \diamond]$ . When we proceed upward, the leaf values of the second and third tree are initially unknown (denoted by  $\diamond$ ) but later recognized as  $\bullet$ . The trick to build the automaton is by *guessing*: when moving bottom-up, the states of the automaton record the set of all possible values for the unknown leaves and later check whether the observed values belong to the guessing set. When proceeding upward, if two guessing sets meet at a certain step, they are *unified* into a single guessing set: first we compute their intersection and then unify it with the observed values at the current node. In details, we define  $F^{\sqcup} = (\mathcal{F} \cup \{\diamond\})^3$ ,  $Q^{\sqcup} = \{q_S \mid S \subseteq \{(\tau_1, \tau_2, \tau_3) \mid \tau_i \in \{\circ, \bullet, \star\}\}\}$  and  $Q_f^{\sqcup} = \{q_{\{(\star, \star, \star)\}}\}$  where  $\star$  indicates the value was already seen. Let  $D = \{\circ, \bullet, \diamond, \text{Node}, \star\}$  and  $\phi : D \times D \rightarrow \{\star, \diamond\}$  be the *unit unification*:

- $\phi(\text{Node}, \star) = \phi(\circ, \circ) = \phi(\bullet, \bullet) = \star, \phi(\diamond, \diamond) = \diamond$
- $\phi(\tau_1, \tau_2)$  is undefined otherwise

We extend  $\phi$  to  $\phi' : D^3 \times \mathbb{P}(D^3) \rightarrow \mathbb{P}(D^3)$  to handle the convolution form of  $\sqcup$ :  $\phi'((\tau_1, \tau_2, \tau_3), S) = \{(\tau'_1, \tau'_2, \tau'_3) \mid \exists k_1, k_2, k_3. ((k_1, k_2, k_3) \in S \wedge_{i=1}^3 \phi(\tau_i, k_i) = \tau'_i)\}$ . Finally, the transition relation is defined to be  $\Delta^{\sqcup} = S_g \cup S_u$  where  $S_g$  contains all *guessing rules* at leaf level for  $\sqcup$  and  $S_u$  is the transition part for unification that contains relations of the form  $[\tau_1, \tau_2, \tau_3](q_{S_1}(v_1), q_{S_2}(v_2)) \mapsto q_S([\tau_1, \tau_2, \tau_3](v_1, v_2))$  such that  $S = \phi'((\tau_1, \tau_2, \tau_3), S_1 \cap S_2)$ . Similarly, the automaton for  $\sqcap$  can be constructed by modifying  $S_g$ . For demonstration, consider  $\widehat{\circ} \sqcap \bullet = \bullet$  whose convolution  $[\text{Node}, \bullet, \bullet](\circ, \diamond, \diamond), [\bullet, \diamond, \diamond]$  has the following run:

- $[\circ, \diamond, \diamond] \mapsto q_{S_1}([\circ, \diamond, \diamond])$  where  $S_1 = \{(\star, \bullet, \bullet), (\star, \circ, \circ)\}$
- $[\bullet, \diamond, \diamond] \mapsto q_{S_2}([\bullet, \diamond, \diamond])$  where  $S_2 = \{(\star, \bullet, \bullet), (\star, \circ, \bullet)\}$
- As  $S = S_1 \cap S_2 = \{(\star, \bullet, \bullet)\}$  and  $\phi'([\text{Node}, \bullet, \bullet], S) = \{(\star, \star, \star)\}$ , we have:  
 $[\text{Node}, \bullet, \bullet](q_{S_1}([\circ, \diamond, \diamond]), q_{S_2}([\bullet, \diamond, \diamond])) \mapsto q_{\{(\star, \star, \star)\}}([\text{Node}, \bullet, \bullet](\circ, \diamond, \diamond), [\bullet, \diamond, \diamond]))$

**Construction of  $\mathcal{A}^{\boxtimes\tau}$ .** Next, we give the description of bottom-up tree automaton  $\mathcal{A}^{\boxtimes\tau}$  that recognizes  $\boxtimes\tau$ . Let  $\mathbb{S} : \mathbb{T} \rightarrow \mathcal{P}(\mathbb{T})$  be the function that extracts all subtrees:  $\mathbb{S}(\bullet) = \{\bullet\}$ ,  $\mathbb{S}(\circ) = \{\circ\}$ ,  $\mathbb{S}(\widehat{\tau_1 \tau_2}) = \{\widehat{\tau_1 \tau_2}\} \cup \mathbb{S}(\tau_1) \cup \mathbb{S}(\tau_2)$ . The ranked alphabet for  $\mathcal{A}^{\boxtimes\tau}$  is  $F^{\boxtimes\tau} = (\mathcal{F} \cup \{\diamond\})^2$ . The state space for  $\mathcal{A}^{\boxtimes\tau}$  is  $Q^{\boxtimes\tau} = \{q_{\tau'} \mid \tau' \in \mathbb{S}(\tau)\} \cup \{q_f\}$  and  $Q_f^{\boxtimes\tau} = \{q_f\}$ . W.l.o.g., we assume that  $\tau \notin \{\circ, \bullet\}$  as those cases can be trivially reduced to equalities, which is a special case of  $\sqcup$ :  $\tau_1 = \tau_2 \Leftrightarrow \tau_1 \sqcup \circ = \tau_2$ . The transition relation  $\Delta^{\boxtimes\tau}$  is defined as follows:

- a1.  $[\circ, \circ] \mapsto q_f([\circ, \circ])$
- a2.  $[\diamond, \tau'] \mapsto q_{\tau'}([\diamond, \tau'])$  if  $\tau' \in \mathbb{S}(\tau)$

## 19:12 Decidability and Complexity of Tree Share Formulas

- $a_3$ .  $[\diamond, \text{Node}](q_{\tau_1}(v_1), q_{\tau_2}(v_2)) \mapsto q_{\tau_3}([\diamond, \text{Node}](v_1, v_2))$  if  $\tau_3 = \begin{array}{c} \wedge \\ \tau_1 \quad \tau_2 \end{array} \in \mathbb{S}(\tau)$  and  $\tau_3 \neq \tau$
- $b_1$ .  $[\bullet, \text{Node}](q_{\tau_1}(v_1), q_{\tau_2}(v_2)) \mapsto q_f([\bullet, \text{Node}](v_1, v_2))$  if  $\tau = \begin{array}{c} \wedge \\ \tau_1 \quad \tau_2 \end{array}$
- $b_2$ .  $[\text{Node}, \text{Node}](q_f(v_1), q_f(v_2)) \mapsto q_f([\text{Node}, \text{Node}](v_1, v_2))$

Briefly speaking, we traverse upward and check whether  $\tau$  is a subtree of the second tree ( $a_1 - a_3$ ). When  $\bullet$  is seen from the first tree ( $b_1$ ), we check whether the two trees have similar shape ( $b_2$ ).

► **Remark.** As we are about to prove, the other relation  $\tau \bowtie$  is not tree-automatic in this representation. We leave an open question whether there exists a representation in which both  $\bowtie_\tau$  and  $\tau \bowtie$  are automatic *i.e.* whether  $\bowtie$  is semi-automatic.

► **Proposition 16** ( $\tau \bowtie$ ). *In the current representation of tree shares, there exists infinitely many  $\tau$  such that  $\tau \bowtie$  is not tree-automatic.*

First, we recall the Pumping Lemma for tree automata:

► **Definition 17** (Term, context and substitution [9]). Let  $\mathcal{A} = (\mathcal{Q}, \mathcal{F}, \mathcal{Q}_f, \Delta)$  be a tree automaton and  $V$  the set of variables. We define  $T(\mathcal{F}, V)$  the set of all tree terms derived from  $\mathcal{F} \cup V$  and  $T(\mathcal{F}, \emptyset)$  is the set of ground terms. A term  $t$  is linear if each variable appears at most once in  $t$ . A context  $C$  is a linear term of  $T(\mathcal{F}, V)$  and  $\mathcal{C}(\mathcal{F})$  denotes the set of all contexts with single variable. A context is trivial if it is reduced to a variable. Let  $C[t]$  be the substitution of  $C \in \mathcal{C}(\mathcal{F})$  by replacing the variable in  $C$  with the term  $t$ . We define  $C^0[t] = v$  where  $v$  is the variable in  $C$ ,  $C^1[t] = C[t]$  and  $C^{n+1}[t] = C[C^n[t]]$ .

► **Lemma 18** (Pumping Lemma for Tree Automata [9]). *Let  $\mathcal{L}$  be the set of all ground terms recognizable by a tree automaton. There is a constant  $k \in \mathbb{N}^+$  satisfying the following condition: for all ground term  $t \in \mathcal{L}$  and  $|t| > k$ , there exists a context  $C \in \mathcal{C}(\mathcal{F})$ , a nontrivial context  $C' \in \mathcal{C}(\mathcal{F})$  and a ground term  $t'$  such that  $t = C[C'[t']]$  and  $\forall n \in \mathbb{N}$ .  $C[C'^n[t']] \in \mathcal{L}$ .*

**Proof of Proposition 16.** Let  $\tau \bowtie$  where  $\tau = \begin{array}{c} \wedge \\ \bullet \quad \circ \quad \bullet \quad \circ \end{array}$ . For an input tree  $\tau' \in \mathbb{T}^+$ , the

result tree is  $\begin{array}{c} \wedge \\ \tau' \quad \circ \quad \tau' \quad \circ \end{array}$  in which the left and right subtree are identical. If  $\tau \bowtie$  is automatic

then it satisfies the Pumping Lemma. However, the Pumping Lemma only allows us to pump either the left or the right subtree and thus they will be different after pumping, which is a contradiction. Now consider the following sequence:  $\tau_1 = \begin{array}{c} \wedge \\ \bullet \quad \circ \quad \bullet \quad \circ \end{array}$ ,  $\tau_{n+1} = \begin{array}{c} \wedge \\ \tau_n \quad \tau_n \end{array}$  then

each of the  $\tau_i \bowtie$  is not automatic. ◀

## 6 Future Work

We identify several directions for future research. One obvious question is the decidability of the existential theory of the tree share model with  $\bowtie$ ,  $\sqcup$ ,  $\sqcap$ , and  $\bar{\sqcap}$ , where  $\bowtie$  is unrestricted. Our connection to word equations does not seem to provide an answer to decidability of this problem. Another question is the computational complexity of adding more general constants (*i.e.*, other than  $\circ$  and  $\bullet$ ) to the first order theory of  $\mathcal{M}$ . Although such constants are in some way “definable” (*e.g.* in first-order logic over the vocabulary with  $\circ$  and  $\bullet$ ), the definition in general requires a formula of superpolynomial size. For this reason, connections to CABA do not immediately yield the same computational complexity. A final direction for future work is to investigate the integration of  $\bowtie_{\square}$  into a practical program logic.

## 7 Conclusion

We have demonstrated the decidability and complexity of a first-order theory over the Boolean logic of tree shares by pinpointing the connection to countable atomless Boolean algebras. We have provided the first serious look at the complexity of the tree multiplication  $\bowtie$  operator and by way of an isomorphism to word equations prove that the existential theory is in PSPACE while the general first-order theory is undecidable. We have identified a restricted version of  $\bowtie$  that takes constants on the right-hand side  $\bowtie_{\square}$  and have proven that the system  $(\mathbb{T}, \sqcup, \sqcap, \bar{\square}, \bowtie_{\square})$  has a decidable first-order theory via an embedding to tree-automatic structures.

---

### References

- 1 Parosh Aziz Abdulla, Yu-Fang Chen, Lukás Holík, Richard Mayr, and Tomás Vojnar. When simulation meets antichains. In *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, pages 158–174, 2010. doi:10.1007/978-3-642-12002-2\_14.
- 2 H. Abdulrab and J.P Pechuchet. Solving word equations. In *Journal of Symbolic Computation*, pages 499–521, 1989.
- 3 Andrew W. Appel, Robert Dockins, Aquinas Hobor, Lennart Beringer, Josiah Dodds, Gordon Stewart, Sandrine Blazy, and Xavier Leroy. *Program Logics for Certified Compilers*. Cambridge University Press, New York, NY, USA, 2014.
- 4 Leonard Berman. The complexity of logical theories. *Theoretical Computer Science*, 11(1):71–77, May 1980. doi:10.1016/0304-3975(80)90037-7.
- 5 A. Blumensath. *Automatic Structures*. PhD thesis, RWTH Aachen, 1999.
- 6 A. Blumensath and E. Grade. Finite presentations of infinite structures: automata and interpretations. In *Theory of Computer Systems*, pages 641–674, 2004.
- 7 John Boyland. Checking interference with fractional permissions. In *Static Analysis, 10th International Symposium, SAS 2003, San Diego, CA, USA, June 11-13, 2003, Proceedings*, pages 55–72, 2003. doi:10.1007/3-540-44898-5\_4.
- 8 J Richard Büchi and Steven Senger. Definability in the existential theory of concatenation and undecidable extensions of this theory. In *The Collected Works of J. Richard Büchi*, pages 671–683. Springer, 1990.
- 9 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: <http://www.grappa.univ-lille3.fr/tata>, 2007. release October, 12th 2007.
- 10 Robert Dockins, Aquinas Hobor, and Andrew W. Appel. A fresh look at separation algebras and share accounting. In *Programming Languages and Systems, 7th Asian Symposium, APLAS 2009, Seoul, Korea, December 14-16, 2009. Proceedings*, pages 161–177, 2009. doi:10.1007/978-3-642-10672-9\_13.
- 11 Aquinas Hobor, Andrew W. Appel, and Francesco Zappa Nardelli. Oracle semantics for concurrent separation logic. In *17th European Symposium of Programming (ESOP 2008)*, pages 353–367, April 2008. URL: [http://www.comp.nus.edu.sg/~hobor/publications/concurrent\\_esop.pdf](http://www.comp.nus.edu.sg/~hobor/publications/concurrent_esop.pdf).
- 12 Aquinas Hobor and Cristian Gherghina. Barriers in concurrent separation logic. In *Programming Languages and Systems – 20th European Symposium on Programming, ESOP 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 276–296, 2011. doi:10.1007/978-3-642-19718-5\_15.

- 13 Joxan Jaffar. Minimal and complete word unification. *J. ACM*, 37(1):47–85, 1990. doi:10.1145/78935.78938.
- 14 Sanjay Jain, Bakhadyr Khoussainov, Frank Stephan, Dan Teng, and Siyuan Zou. Semi-automatic structures. In *Computer Science – Theory and Applications – 9th International Computer Science Symposium in Russia, CSR 2014, Moscow, Russia, June 7-11, 2014. Proceedings*, pages 204–217, 2014. doi:10.1007/978-3-319-06686-8\_16.
- 15 Bakhadyr Khoussainov and Mia Minnes. Three lectures on automatic structures. In *Logic Colloquium 2007 (F. Delon, U. Kohlenbach, P. Maddy, and F. Stephan, editors), Lecture Notes in Logic, vol. 35, Association for Symbolic Logic*, pages 132–176, 2007.
- 16 Antoni Koscielski and Leszek Pacholski. Complexity of Makanin’s algorithm. *J. ACM*, 43(4):670–684, 1996. doi:10.1145/234533.234543.
- 17 Dexter Kozen. Complexity of boolean algebras. In *Theoretical Computer Science 10*, pages 221–247, 1980.
- 18 D. Kuske. Theories of orders on the set of words. In *Theoretical Informatics and Applications 40*, pages 53–74, 2006.
- 19 Xuan Bach Le, Cristian Gherghina, and Aquinas Hobor. Decision procedures over sophisticated fractional permissions. In *Programming Languages and Systems – 10th Asian Symposium, APLAS 2012, Kyoto, Japan, December 11-13, 2012. Proceedings*, pages 368–385, 2012. doi:10.1007/978-3-642-35182-2\_26.
- 20 G. S Makanin. The problem of solvability of equations in a free semigroup. In *Mat. Sbornik*, pages 147–236, 1977.
- 21 G. S Makanin. Equations in a free group. In *Izvestiya AN SSSR*, pages 1199–1273, 1982–1983.
- 22 G.S Makanin. Decidability of the universal and positive theories of a free group. In *Izvestiya AN SSSR*, pages 735–749, 1984–1985.
- 23 S. Marchenkov. Unsolvability of positive  $\forall\exists$ -theory of free semi-group. In *Sibirsky matematichesky jurnal*, pages 196–198, 1982.
- 24 Kim Marriott and Martin Odersky. Negative boolean constraints. In *Theoretical Computer Science 160*, pages 365–380, 1996.
- 25 W. Plandowski. Satisfiability of word equations with constants is in PSPACE. In *Journal of the Association for Computing Machinery*, pages 483–496, 2004.
- 26 W. Plandowski. An efficient algorithm for solving word equations. In *Proc 38th Annual Symposium on Theory of Computing*, pages 467–476, 2006.
- 27 M.O. Rabin. Decidability of second-order theories and automata on infinite trees. In *Trans AMS*, pages 341–360, 1960.
- 28 Stijn Vermeeren. Embeddings into the countable atomless Boolean algebra. 2010. URL: <http://stijnvermeeren.be/download/mathematics/embeddings.pdf>.
- 29 Jules Villard. *Heaps and Hops*. Ph.D. thesis, Laboratoire Spécification et Vérification, École Normale Supérieure de Cachan, France, February 2011.

# One-Counter Automata with Counter Observability

Benedikt Bollig

LSV, ENS Cachan, CNRS, Inria, Université Paris-Saclay, France  
bollig@lsv.fr

---

## Abstract

In a one-counter automaton (OCA), one can produce a letter from some finite alphabet, increment and decrement the counter by one, or compare it with constants up to some threshold. It is well-known that universality and language inclusion for OCAs are undecidable. In this paper, we consider OCAs with counter observability: Whenever the automaton produces a letter, it outputs the current counter value along with it. Hence, its language is now a set of words over an infinite alphabet. We show that universality and inclusion for that model are PSPACE-complete, thus no harder than the corresponding problems for finite automata. In fact, by establishing a link with visibly one-counter automata, we show that OCAs with counter observability are effectively determinizable and closed under all boolean operations. Moreover, it turns out that they are expressively equivalent to strong automata, in which transitions are guarded by MSO formulas over the natural numbers with successor.

**1998 ACM Subject Classification** F.1.1 Models of Computation

**Keywords and phrases** One-counter automata, inclusion checking, observability, visibly one-counter automata, strong automata

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.20

## 1 Introduction

One-counter automata (OCAs) are a fundamental model of infinite-state systems. Their expressive power resides between finite automata and pushdown automata. Unlike finite automata, however, OCAs are not robust: They lack closure under complementation and have an undecidable universality, equivalence, and inclusion problem [12, 14]. Several directions to overcome this drawback have been taken. One may underapproximate the above decision problems in terms of bisimilarity [15] or overapproximate the system behavior by a finite-state abstraction, e.g., in terms of the downward closure or preserving the Parikh image [25, 20].

In this paper, we consider a new and simple way of obtaining a robust model of one-counter systems. Whenever the automaton produces a letter from a finite alphabet  $\Sigma$ , it will also output the *current* counter value along with it (transitions that manipulate the counter are not concerned). Hence, its language is henceforth a subset of  $(\Sigma \times \mathbb{N})^*$ . For obvious reasons, we call this variant *OCAs with counter observability*. We will show that, under the observability semantics, OCAs form a robust automata model: They are closed under all boolean operations. Moreover, their universality and inclusion problem are in PSPACE and, as a simple reduction from universality for finite automata shows, PSPACE-complete.

These results may come as a surprise given that universality for OCAs is undecidable and introducing counter observability seems like an extension of OCAs. But, actually, the problem becomes quite different. The fact that a priori hidden details from a run (in terms of the counter values) are revealed makes the model more robust and the decision problems easier. Note that this is also what happens in input-driven/visibly pushdown automata [16, 3]



© Benedikt Bollig;

licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 20; pp. 20:1–20:14



Leibniz International Proceedings in Informatics

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

or their restriction of visibly OCAs [5, 22]. They all recognize languages over a *finite* alphabet and the stack/counter operation can be deduced from the letter that is read. Interestingly, our proofs establish a link between the observability semantics and visibly OCAs. This link is not immediate and relies on a couple of technical lemmas. However, it somehow explains the robustness of OCAs under the observability semantics.

It is worth noting that revealing details from a system configuration does not always help, quite the contrary: Though timed automata and counter automata are closely related [13], the universality problem of timed automata is decidable only if time stamps are excluded from the semantics [1].

Note that it is not only for the pure fact that we obtain a robust model that we consider counter observability. Counter values usually have a *meaning*, such as energy level, value of a variable, or number of items yet to be produced (cf. Example 2). In those contexts, it is natural to include them in the semantics, just like including time stamps in timed automata.

Apart from the connection with visibly OCAs, another model closely related to ours is that of *strong automata* [9]. Strong automata operate on infinite alphabets and were introduced as an extension of *symbolic automata* [6, 10]. Essentially, a transition of a strong automaton is labeled with a formula from monadic second-order (MSO) logic over some infinite structure, say  $(\mathbb{N}, +1)$ . In fact, the formula has two free first-order variables so that it defines a binary relation over  $\mathbb{N}$ . This relation is interpreted as a constraint between successive letters from the infinite alphabet. We will show that OCAs with the observability semantics and strong automata over  $(\mathbb{N}, +1)$  (extended by a component for the finite alphabet  $\Sigma$ ) are expressively equivalent. This underpins a certain naturalness of the observability semantics. Note that the universality and the inclusion problem have been shown decidable for strong automata over  $(\mathbb{N}, +1)$  [9]. However, strong automata do not allow us to derive any elementary complexity upper bounds. In fact, our model can be seen as an operational counterpart of strong automata over  $(\mathbb{N}, +1)$ .

The outline of the paper is as follows. Section 2 defines OCAs and their different semantics. Section 3 relates the observability semantics with visibly OCAs and shows that, under the new semantics, OCAs are closed under boolean operations and have a PSPACE-complete universality and inclusion problem. In Section 4, we show expressive equivalence of strong automata and OCAs with counter observability. We conclude in Section 5. Omitted proofs or proof details can be found in the full version of this paper, which is available at the following link: <https://arxiv.org/abs/1602.05940>

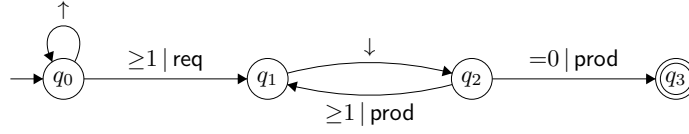
## 2 One-Counter Automata with Counter Observability

For  $n \in \mathbb{N} = \{0, 1, 2, \dots\}$ , we set  $[n] := \{1, \dots, n\}$  and  $[n]_0 := \{0, 1, \dots, n\}$ . Given an alphabet  $\Sigma$ , the set of finite words over  $\Sigma$ , including the empty word  $\varepsilon$ , is denoted by  $\Sigma^*$ .

### 2.1 One-Counter Automata and Their Semantics

We consider ordinary one-counter automata over some nonempty finite alphabet  $\Sigma$ . In addition to a finite-state control and transitions that produce a letter from  $\Sigma$ , they have a counter that can be incremented, decremented, or tested for values up to some threshold  $m \in \mathbb{N}$  (as defined in [5]). Accordingly, the set of *counter operations* is  $\text{Op} = \{\uparrow, \downarrow\}$ , where  $\uparrow$  stands for “increment the counter by one” and  $\downarrow$  for “decrement the counter by one”. A transition is of the form  $(q, k, \sigma, q')$  where  $q, q'$  are states,  $k \in [m]_0$  is a counter test, and  $\sigma \in \Sigma \cup \text{Op}$ . It leads from  $q$  to  $q'$ , while  $\sigma$  either produces a letter from  $\Sigma$  or modifies the





■ **Figure 1** A one-counter automaton with threshold  $m = 1$ .

counter. However, the transition can only be taken if the current counter value  $x \in \mathbb{N}$  satisfies  $k = \min\{x, m\}$ . That is, counter values can be checked against any number strictly below  $m$  or for being at least  $m$ . In particular, if  $m = 1$ , then we deal with the classical definition of one-counter automata, which only allows for zero and non-zero tests.

► **Definition 1** (OCA, cf. [5]). A *one-counter automaton* (or simply OCA) is a tuple  $\mathcal{A} = (Q, \Sigma, \iota, F, m, \Delta)$  where  $Q$  is a finite set of *states*,  $\Sigma$  is a nonempty finite alphabet (disjoint from  $\text{Op}$ ),  $\iota \in Q$  is the *initial state*,  $F \subseteq Q$  is the set of *final states*,  $m \in \mathbb{N}$  is the *threshold*, and  $\Delta \subseteq Q \times [m]_0 \times (\Sigma \cup \text{Op}) \times Q$  is the *transition relation*. We also say that  $\mathcal{A}$  is an  $m$ -OCA. Its size is defined as  $|Q| + |\Sigma| + m + |\Delta|$ .

An OCA  $\mathcal{A} = (Q, \Sigma, \iota, F, m, \Delta)$  can have several different semantics:

- $L_{oca}(\mathcal{A}) \subseteq \Sigma^*$  is the classical semantics when  $\mathcal{A}$  is seen as an ordinary OCA.
- $L_{vis}(\mathcal{A}) \subseteq (\Sigma \cup \text{Op})^*$  is the *visibly semantics* where, in addition to the letters from  $\Sigma$ , all counter movements are made apparent.
- $L_{obs}(\mathcal{A}) \subseteq (\Sigma \times \mathbb{N})^*$  is the *semantics with counter observability* where the current counter value is output each time a  $\Sigma$ -transition is taken.

We define all three semantics in one go. Let  $\text{Conf}_{\mathcal{A}} := Q \times \mathbb{N}$  be the set of *configurations* of  $\mathcal{A}$ . In a configuration  $(q, x)$ ,  $q$  is the current state and  $x$  is the current counter value. The *initial* configuration is  $(\iota, 0)$ , and a configuration  $(q, x)$  is *final* if  $q \in F$ .

We determine a global transition relation  $\Longrightarrow_{\mathcal{A}} \subseteq \text{Conf}_{\mathcal{A}} \times ((\Sigma \times \mathbb{N}) \cup \text{Op}) \times \text{Conf}_{\mathcal{A}}$ . For two configurations  $(q, x), (q', x') \in \text{Conf}_{\mathcal{A}}$  and  $\tau \in (\Sigma \times \mathbb{N}) \cup \text{Op}$ , we have  $(q, x) \xrightarrow{\tau}_{\mathcal{A}} (q', x')$  if one of the following holds:

- $\tau = \uparrow$  and  $x' = x + 1$  and  $(q, \min\{x, m\}, \uparrow, q') \in \Delta$ , or
- $\tau = \downarrow$  and  $x' = x - 1$  and  $(q, \min\{x, m\}, \downarrow, q') \in \Delta$ , or
- $x' = x$  and there is  $a \in \Sigma$  such that  $\tau = (a, x)$  and  $(q, \min\{x, m\}, a, q') \in \Delta$ .

A *partial run* of  $\mathcal{A}$  is a sequence  $\rho = (q_0, x_0) \xrightarrow{\tau_1}_{\mathcal{A}} (q_1, x_1) \xrightarrow{\tau_2}_{\mathcal{A}} \dots \xrightarrow{\tau_n}_{\mathcal{A}} (q_n, x_n)$ , with  $n \geq 0$ . If, in addition,  $(q_0, x_0)$  is the initial configuration, then we say that  $\rho$  is a *run*. We call  $\rho$  *accepting* if its last configuration  $(q_n, x_n)$  is final.

Now, the semantics of  $\mathcal{A}$  that we consider depends on what we would like to extract from  $\text{trace}(\rho) := \tau_1 \dots \tau_n \in ((\Sigma \times \mathbb{N}) \cup \text{Op})^*$ . We let (given  $(a, x) \in \Sigma \times \mathbb{N}$ ):

- $oca((a, x)) = a$  and  $oca(\uparrow) = oca(\downarrow) = \varepsilon$
- $vis((a, x)) = a$  and  $vis(\uparrow) = \uparrow$  and  $vis(\downarrow) = \downarrow$
- $obs((a, x)) = (a, x)$  and  $obs(\uparrow) = obs(\downarrow) = \varepsilon$

Moreover, we extend each such mapping  $\eta \in \{oca, vis, obs\}$  to  $\tau_1 \dots \tau_n \in ((\Sigma \times \mathbb{N}) \cup \text{Op})^*$  letting  $\eta(\tau_1 \dots \tau_n) := \eta(\tau_1) \cdot \dots \cdot \eta(\tau_n)$ . Note that, hereby  $u \cdot \varepsilon = \varepsilon \cdot u = u$  for any word  $u$ .

Finally, we let  $L_{\eta}(\mathcal{A}) = \{\eta(\text{trace}(\rho)) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$ .

► **Example 2.** Consider the 1-OCA  $\mathcal{A}$  from Figure 1 over  $\Sigma = \{\text{req}, \text{prod}\}$ . For readability, counter tests 0 and 1 are written as  $=0$  and  $\geq 1$ , respectively. A transition without counter test stands for two distinct transitions, one for  $=0$  and one for  $\geq 1$  (i.e., the counter value may actually be arbitrary). When looking at the semantics  $L_{obs}(\mathcal{A})$ , i.e., with counter

observability, we can think of  $(\text{req}, n)$  signaling that the production of  $n \geq 1$  items is required (where  $n$  is the current counter value). Moreover,  $\text{prod}$  indicates that an item has been produced so that, along a run, the counter value represents the number of items yet to be produced. It is thus natural to include it in the semantics. Concretely, we have the following:

- $L_{oca}(\mathcal{A}) = \{\text{req prod}^n \mid n \geq 1\}$
- $L_{vis}(\mathcal{A}) = \{\uparrow^n \text{req} (\downarrow \text{prod})^n \mid n \geq 1\}$
- $L_{obs}(\mathcal{A}) = \{(\text{req}, n)(\text{prod}, n-1)(\text{prod}, n-2) \dots (\text{prod}, 0) \mid n \geq 1\}$

Apparently,  $L_{vis}(\mathcal{A})$  and  $L_{obs}(\mathcal{A})$  are the only meaningful semantics in the context described above.

► **Remark.** Visibly OCAs [5, 22] usually allow for general input alphabets  $\Gamma$ , which are partitioned into  $\Gamma = \Gamma_{\text{inc}} \uplus \Gamma_{\text{dec}} \uplus \Gamma_{\text{nop}}$  so that every  $\gamma \in \Gamma$  is associated with a unique counter operation (or “no counter operation” if  $\gamma \in \Gamma_{\text{nop}}$ ). In fact, we consider here (wrt. the visibly semantics) a particular case where  $\Gamma = \Sigma \cup \text{Op}$  with  $\Gamma_{\text{inc}} = \{\uparrow\}$ ,  $\Gamma_{\text{dec}} = \{\downarrow\}$ , and  $\Gamma_{\text{nop}} = \Sigma$ .

## 2.2 Standard Results for OCAs

Let us recall some well-known results for classical OCAs and visibly OCAs. For  $\eta \in \{oca, vis, obs\}$ , the *nonemptiness problem for OCAs* wrt. the  $\eta$ -semantics is defined as follows: Given an OCA  $\mathcal{A}$ , do we have  $L_\eta(\mathcal{A}) \neq \emptyset$ ? Of course, this reduces to a reachability problem that is independent of the actual choice of the semantics:

► **Theorem 3** ([24]). *The nonemptiness problem for OCAs is NL-complete, wrt. any of the three semantics.*

However, the universality (and, therefore, inclusion) problem for classical OCAs is undecidable:

► **Theorem 4** ([12, 14]). *The following problem is undecidable: Given an OCA  $\mathcal{A}$  with alphabet  $\Sigma$ , do we have  $L_{oca}(\mathcal{A}) = \Sigma^*$ ?*

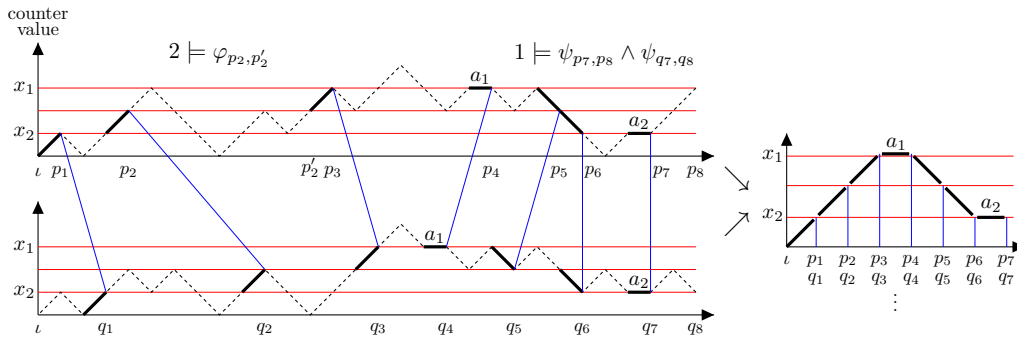
In this paper, we show that universality and inclusion are decidable when considering counter observability. To do so, we make use of the theory of the visibly semantics. Concretely, we exploit determinizability as well as closure under complementation and intersection. In fact, the following definition of determinism only makes sense for the visibly semantics, but we will see later that a subclass of deterministic OCAs gives a natural notion of determinism for the observability semantics as well.

► **Definition 5** (deterministic OCA). An OCA  $\mathcal{A} = (Q, \Sigma, \iota, F, m, \Delta)$  is called *deterministic* (dOCA or  $m$ -dOCA) if, for all  $(q, k, \sigma) \in Q \times [m]_0 \times (\Sigma \cup \text{Op})$ , there is exactly one  $q' \in Q$  such that  $(q, k, \sigma, q') \in \Delta$ . In that case,  $\Delta$  represents a (total) function  $\delta : Q \times [m]_0 \times (\Sigma \cup \text{Op}) \rightarrow Q$  so that we rather consider  $\mathcal{A}$  to be the tuple  $(Q, \Sigma, \iota, F, m, \delta)$ .

A powerset construction like for finite automata can be used to determinize OCAs wrt. the visibly semantics [5]. That construction also preserves the two other semantics. However, Definition 5 only guarantees uniqueness of runs for words from  $(\Sigma \cup \text{Op})^*$ . That is, for complementation, we have to restrict to the visibly semantics (cf. Lemma 7 below).

► **Lemma 6** (cf. [5]). *Let  $\mathcal{A}$  be an  $m$ -OCA. There is an  $m$ -dOCA  $\mathcal{A}^{det}$  of exponential size such that  $L_\eta(\mathcal{A}^{det}) = L_\eta(\mathcal{A})$  for all  $\eta \in \{oca, vis, obs\}$ .*





■ **Figure 2** Decompositions of two runs on  $(a_1, 3)(a_2, 1)$ , and corresponding runs in normal form.

A (visibly) dOCA can be easily complemented wrt. the set of *well-formed words*  $WF_\Sigma := \{w \in (\Sigma \cup \text{Op})^* \mid \text{no prefix of } w \text{ contains more } \downarrow\text{'s than } \uparrow\text{'s}\}$ . In fact, for all OCAs  $\mathcal{A}$  with alphabet  $\Sigma$ , we have  $L_{vis}(\mathcal{A}) \subseteq WF_\Sigma$ .

► **Lemma 7.** *Let  $\mathcal{A} = (Q, \Sigma, \iota, F, m, \delta)$  be a dOCA and define  $\bar{\mathcal{A}}$  as the dOCA  $(Q, \Sigma, \iota, Q \setminus F, m, \delta)$ . Then,  $L_{vis}(\bar{\mathcal{A}}) = WF_\Sigma \setminus L_{vis}(\mathcal{A})$ .*

Finally, visibility of counter operations allows us to simulate two OCAs in sync by a straightforward product construction:

► **Lemma 8** (cf. [3]). *Let  $\mathcal{A}_1$  be an  $m_1$ -OCA and  $\mathcal{A}_2$  be an  $m_2$ -OCA over the same alphabet. There is a  $\max\{m_1, m_2\}$ -OCA  $\mathcal{A}_1 \times \mathcal{A}_2$  of polynomial size such that  $L_{vis}(\mathcal{A}_1 \times \mathcal{A}_2) = L_{vis}(\mathcal{A}_1) \cap L_{vis}(\mathcal{A}_2)$ . Moreover, if  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are deterministic, then so is  $\mathcal{A}_1 \times \mathcal{A}_2$ .*

### 3 Determinizing and Complementing OCAs

In this section, we will show that, under the *observability semantics*, OCAs are effectively closed under all boolean operations. The main ingredient of the proof is a determinization procedure, which we first describe informally.

Let  $\mathcal{A} = (Q, \Sigma, \iota, F, m, \Delta)$  be the OCA to be determinized (wrt. the observability semantics). Moreover, let  $w = (a_1, x_1) \dots (a_n, x_n) \in (\Sigma \times \mathbb{N})^*$ . Every run  $\rho$  of  $\mathcal{A}$  such that  $obs(trace(\rho)) = w$  has to have reached the counter value  $x_1$  by the time it reads the first letter  $a_1$ . In particular, it has to perform at least  $x_1$  counter increments. In other words, we can identify  $x_1$  transitions that lift the counter value from 0 to 1, from 1 to 2, and, finally, from  $x_1 - 1$  to  $x_1$ , respectively, and that are separated by partial runs that “oscillate” around the current counter value but, at the end, return to their original level. Similarly, before reading the second letter  $a_2$ ,  $\mathcal{A}$  will perform  $|x_2 - x_1|$ -many *identical* counter operations to reach  $x_2$ , again separated by some oscillation phases, and so on. This is illustrated on the left hand side of Figure 2 for two runs on the word  $(a_1, 3)(a_2, 1)$ .

We will transform  $\mathcal{A}$  into another automaton that decomposes a run into oscillations and increment/decrement/letter transitions, but, in fact, abstracts away oscillations. Thus, the automaton starts in an increasing mode and goes straight to the value  $x_1$ . Once it reads letter  $a_1$ , it may go into an increasing or decreasing mode, and so on. Observe that a run  $\rho$  of this new automaton is in a sort of normal form as illustrated on the right hand side of Figure 2. The crux is that  $vis(trace(\rho))$  is a *unique* encoding of  $w$ : Of course, it determines the counter values output when a letter is read; and it is unique, since it continues incrementing (decrementing, respectively) until a letter is read. This normalization and encoding finally

allows us to apply known results on visibly one-counter automata for determinization and complementation.

There is a little issue here, since the possibility of performing an oscillation leading from  $p_2$  to  $p'_2$  (cf. left hand side of Figure 2) depends on the current counter value. However, it was shown in [11] that the set of counter values allowing for such a shortcut can be described as a boolean combination of arithmetic progressions that can be computed in polynomial time. We will, therefore, work with an extended version of OCAs that includes arithmetic-progression tests (but is no more expressive, as we show afterwards).

The outline of this section is as follows: We present extended OCAs in Section 3.1 and the link between the observability and the visibly semantics in Section 3.2. In Section 3.3, we solve the universality and inclusion problem for OCAs wrt. the observability semantics.

### 3.1 Extended One-Counter Automata

While OCAs can only test a counter value up to some threshold, *extended OCAs* have access to boolean combinations of modulo constraints. The set  $\mathbf{Guards}^{\text{mod}}$  is given by the grammar  $\varphi ::= c + d\mathbb{N} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$  where  $c, d \in \mathbb{N}$ . We call  $c + d\mathbb{N}$  an *arithmetic-progression formula* and assume that  $c$  and  $d$  are encoded in unary. For  $x \in \mathbb{N}$  (a counter value), we define  $x \models c + d\mathbb{N}$  if  $x = c + d \cdot i$  for some  $i \in \mathbb{N}$ . Thus, we may use *true* as an abbreviation for  $0 + 1\mathbb{N}$ . The other formulas are interpreted as expected. Moreover, given  $\varphi \in \mathbf{Guards}^{\text{mod}}$ , we set  $\llbracket \varphi \rrbracket := \{x \in \mathbb{N} \mid x \models \varphi\}$ .

Before we introduce extended OCAs, we will state a lemma saying that the “possibility” of a shortcut in terms of an oscillation (see above) is definable in  $\mathbf{Guards}^{\text{mod}}$ . Let  $\mathcal{A} = (Q, \Sigma, \iota, F, m, \Delta)$  be an OCA and  $p, q \in Q$ . By  $X_{p,q}^{\mathcal{A}}$ , we denote the set of natural numbers  $x \in \mathbb{N}$  such that  $(p, x) (\xrightarrow{\uparrow}_{\mathcal{A}} \cup \xrightarrow{\downarrow}_{\mathcal{A}})^* (q, x)$ , i.e., there is a partial run from  $(p, x)$  to  $(q, x)$  that performs only counter operations. Moreover, we define  $Y_{p,q}^{\mathcal{A}}$  to be the set of natural numbers  $x \in \mathbb{N}$  such that  $(p, x) (\xrightarrow{\uparrow}_{\mathcal{A}} \cup \xrightarrow{\downarrow}_{\mathcal{A}})^* (q, x')$  for some  $x' \in \mathbb{N}$ . Note that  $X_{p,q}^{\mathcal{A}} \subseteq Y_{p,q}^{\mathcal{A}}$ . The following result is due to [11, Lemmas 6–9]:

► **Lemma 9** ([11]). *Let  $\mathcal{A} = (Q, \Sigma, \iota, F, m, \Delta)$  be an OCA and  $p, q \in Q$ . We can compute, in polynomial time, guards  $\varphi_{p,q}, \psi_{p,q} \in \mathbf{Guards}^{\text{mod}}$  such that  $\llbracket \varphi_{p,q} \rrbracket = X_{p,q}^{\mathcal{A}}$  and  $\llbracket \psi_{p,q} \rrbracket = Y_{p,q}^{\mathcal{A}}$ . In particular, the constants in  $\varphi_{p,q}$  and  $\psi_{p,q}$  are all polynomially bounded.*

► **Definition 10** (extended OCA). An *extended OCA* (eOCA) is a tuple  $\mathcal{A} = (Q, \Sigma, \iota, f, \Delta)$  where  $Q, \Sigma, \iota$  are like in an OCA,  $f : Q \rightarrow \mathbf{Guards}^{\text{mod}}$  is the *acceptance condition*, and  $\Delta$  is the *transition relation*: a finite subset of  $Q \times \mathbf{Guards}^{\text{mod}} \times (\Sigma \cup \text{Op}) \times Q$

Runs and the languages  $L_\eta(\mathcal{A})$ , with  $\eta \in \{oca, vis, obs\}$ , of an eOCA  $\mathcal{A} = (Q, \Sigma, \iota, f, \Delta)$  are defined very similarly to OCAs. In fact, there are only two (slight) changes:

1. The definition of  $\xRightarrow{\mathcal{A}} \subseteq \text{Conf}_{\mathcal{A}} \times ((\Sigma \times \mathbb{N}) \cup \text{Op}) \times \text{Conf}_{\mathcal{A}}$  is now as follows: For  $(q, x), (q', x') \in \text{Conf}_{\mathcal{A}}$  and  $\tau \in (\Sigma \times \mathbb{N}) \cup \text{Op}$ , we have  $(q, x) \xRightarrow{\tau}_{\mathcal{A}} (q', x')$  if there is  $\varphi \in \mathbf{Guards}^{\text{mod}}$  such that  $x \models \varphi$  and one of the following holds:
  - $\tau = \uparrow$  and  $x' = x + 1$  and  $(q, \varphi, \uparrow, q') \in \Delta$ , or
  - $\tau = \downarrow$  and  $x' = x - 1$  and  $(q, \varphi, \downarrow, q') \in \Delta$ , or
  - $x' = x$  and there is  $a \in \Sigma$  such that  $\tau = (a, x)$  and  $(q, \varphi, a, q') \in \Delta$ .
2. A run is now *accepting* if its last configuration  $(q, x)$  is such that  $x \models f(q)$ .

Apart from these modifications, the languages  $L_{oca}(\mathcal{A})$ ,  $L_{vis}(\mathcal{A})$ , and  $L_{obs}(\mathcal{A})$  are defined in exactly the same way as for OCAs.

### 3.2 From OCAs with Counter Observability to Visibly OCAs

To establish a link between the observability and the visibly semantics, we will encode a word  $w = (a_1, x_1)(a_2, x_2) \dots (a_n, x_n) \in (\Sigma \times \mathbb{N})^*$  as a word  $\text{enc}(w) \in (\Sigma \cup \text{Op})^*$  as follows:

$$\text{enc}(w) := \uparrow^{x_1} a_1 \text{sign}(x_2 - x_1)^{|x_2 - x_1|} a_2 \dots \text{sign}(x_n - x_{n-1})^{|x_n - x_{n-1}|} a_n$$

where, for an integer  $z \in \mathbb{Z}$ , we let  $\text{sign}(z) = \uparrow$  if  $z \geq 0$ , and  $\text{sign}(z) = \downarrow$  if  $z < 0$ . For example,  $\text{enc}(\varepsilon) = \varepsilon$  and  $\text{enc}((a, 5)(b, 2)(c, 4)) = \uparrow^5 a \downarrow^3 b \uparrow^2 c$ . The mapping  $\text{enc}$  is extended to sets  $L \subseteq (\Sigma \times \mathbb{N})^*$  by  $\text{enc}(L) = \{\text{enc}(w) \mid w \in L\}$ . Let  $\text{Enc}_\Sigma := \text{enc}((\Sigma \times \mathbb{N})^*)$  denote the set of *valid encodings*. Note that  $\text{enc}$  is a *bijection* between  $(\Sigma \times \mathbb{N})^*$  and  $\text{Enc}_\Sigma$ , and that  $\text{Enc}_\Sigma$  is the set of *well-formed* words of the form  $u_1 a_1 u_2 a_2 \dots u_n a_n$  where  $a_i \in \Sigma$  and  $u_i \in \{\uparrow\}^* \cup \{\downarrow\}^*$  for all  $i \in \{1, \dots, n\}$ .

Obviously, there is a small dOCA whose visibly semantics is  $\text{Enc}_\Sigma$ . It will be needed later for complementation of OCAs wrt. the observability semantics.

► **Lemma 11.** *There is a 0-dOCA  $\mathcal{B}_{\text{enc}}$  with only four states such that  $L_{\text{vis}}(\mathcal{B}_{\text{enc}}) = \text{Enc}_\Sigma$ .*

The idea is that  $\mathcal{B}_{\text{enc}}$  enters an “increasing” or “decreasing” mode as soon as it performs  $\uparrow$  or, respectively,  $\downarrow$ . Such a mode can only be quit by reading a letter from  $\Sigma$  or entering a sink state. This avoids forbidden reversals between  $\uparrow$  and  $\downarrow$ . Finally, it is easy to ensure that any nonempty accepted word ends in a letter from  $\Sigma$ .

In fact, there is a tight link between the visibly and the observability semantics of OCAs provided the visibly semantics contains only valid encodings:

► **Lemma 12.** *Let  $\mathcal{A}$  be an OCA with alphabet  $\Sigma$  such that  $L_{\text{vis}}(\mathcal{A}) \subseteq \text{Enc}_\Sigma$ . Then, we have  $L_{\text{vis}}(\mathcal{A}) = \text{enc}(L_{\text{obs}}(\mathcal{A}))$  and, equivalently,  $L_{\text{obs}}(\mathcal{A}) = \text{enc}^{-1}(L_{\text{vis}}(\mathcal{A}))$ .*

Lemmas 8 and 12 imply the following closure property, which will later be exploited to solve the inclusion problem:

► **Proposition 13.** *Let  $\mathcal{A}_1$  and  $\mathcal{A}_2$  be OCAs over  $\Sigma$  such that  $L_{\text{vis}}(\mathcal{A}_1) \subseteq \text{Enc}_\Sigma$  and  $L_{\text{vis}}(\mathcal{A}_2) \subseteq \text{Enc}_\Sigma$ . Then,  $L_{\text{obs}}(\mathcal{A}_1 \times \mathcal{A}_2) = L_{\text{obs}}(\mathcal{A}_1) \cap L_{\text{obs}}(\mathcal{A}_2)$  (where  $\mathcal{A}_1 \times \mathcal{A}_2$  is due to Lemma 8).*

The next lemma constitutes the main ingredient of the determinization procedure. It will eventually allow us to rely on OCAs whose visibly semantics consists only of valid encodings.

► **Lemma 14.** *Let  $\mathcal{A}$  be an OCA. We can compute, in polynomial time, an eOCA  $\mathcal{A}^{\text{ext}}$  such that  $L_{\text{obs}}(\mathcal{A}^{\text{ext}}) = L_{\text{obs}}(\mathcal{A})$  and, for all  $w \in L_{\text{obs}}(\mathcal{A}^{\text{ext}})$ , we have  $\text{enc}(w) \in L_{\text{vis}}(\mathcal{A}^{\text{ext}})$ .*

**Proof.** Suppose  $\mathcal{A} = (Q, \Sigma, \iota, F, m, \Delta)$  is the given OCA. We first translate a simple “threshold constraint” into an arithmetic expression that can be used as a guard in the eOCA  $\mathcal{A}^{\text{ext}}$ : Let  $\pi_m = m + 1\mathbb{N}$ , and  $\pi_k = k + 0\mathbb{N}$  for all  $k \in \{0, \dots, m-1\}$ .

We define  $\mathcal{A}^{\text{ext}} = (Q, \Sigma, \iota, f, \Delta')$  as follows: Essentially,  $\mathcal{A}^{\text{ext}}$  simulates  $\mathcal{A}$  so that it has the same state space. However, when  $\mathcal{A}$  allows for a shortcut (oscillation) from state  $p$  to state  $q$  (which will be checked in terms of  $\varphi_{p,q}$  from Lemma 9) and there is a transition  $(q, k, \sigma, q')$  of  $\mathcal{A}$ , then  $\mathcal{A}^{\text{ext}}$  may perform  $\sigma$  and go directly from  $p$  to  $q'$ , provided  $\pi_k$  is satisfied as well. Formally, the transition relation is given as

$$\Delta' = \{(p, \varphi_{p,q} \wedge \pi_k, \sigma, q') \mid p \in Q \text{ and } (q, k, \sigma, q') \in \Delta\}.$$

Moreover, a configuration  $(p, x)$  is “final” in  $\mathcal{A}^{\text{ext}}$  if the current counter value  $x$  satisfies  $\psi_{p,q}$  for some  $q \in F$  (cf. Lemma 9). That is, for all  $p \in Q$ , we let  $f(p) = \bigvee_{q \in F} \psi_{p,q}$ . ◀

To transform an eOCA back into an ordinary OCA while determinizing it and preserving its observability semantics, we will need a dOCA that takes care of the modulo constraints:

► **Lemma 15.** *Let  $\Omega \subseteq \text{Guards}^{\text{mod}}$  be a nonempty finite set. Set  $m_\Omega := \max\{c \mid c + d\mathbb{N} \text{ is an atomic subformula of some } \varphi \in \Omega\} + 2$ . There are a dOCA  $\mathcal{B}_\Omega = (Q, \Sigma, \iota, Q, m_\Omega, \delta)$  of exponential size and  $\lambda : Q \rightarrow 2^\Omega$  such that, for all  $(q, x) \in \text{Conf}_{\mathcal{B}_\Omega}$  and all runs of  $\mathcal{B}_\Omega$  ending in  $(q, x)$ , we have  $\lambda(q) = \{\varphi \in \Omega \mid x \models \varphi\}$ .*

**Proof.** We sketch the idea. For every arithmetic-progression formula  $c + d\mathbb{N}$  that occurs in  $\Omega$  (for simplicity, let us assume  $d \geq 1$ ), we introduce a state component  $\{0, 1, \dots, c\} \times \{0, 1, \dots, d-1\}$ . Increasing the counter, we increment the first component until  $c$  and then count modulo  $d$  in the second. We proceed similarly when decreasing the counter. The current state  $(x, y) \in [c]_0 \times [d-1]_0$  will then tell us whether  $c + d\mathbb{N}$  holds, namely iff  $x = c$  and  $y = 0$ . Finally, the mapping  $\lambda$  evaluates a formula based on the outcome for its atomic subformulas. Note that  $\mathcal{B}_\Omega$  can be computed in exponential time. Its size is exponential in the number of arithmetic-progression formulas that occur in  $\Omega$ . ◀

We will now apply Lemma 15 to transform an eOCA into a dOCA (cf. also Lemma 6).

► **Lemma 16.** *Let  $\mathcal{A}$  be an eOCA. We can compute, in exponential time, a dOCA  $\mathcal{A}'$  (deterministic according to Definition 5) such that  $L_\eta(\mathcal{A}') = L_\eta(\mathcal{A})$  for all  $\eta \in \{\text{oca}, \text{vis}, \text{obs}\}$ .*

**Proof.** Suppose  $\mathcal{A} = (Q, \Sigma, \iota, f, \Delta)$  is the given eOCA. Let  $\Omega \subseteq \text{Guards}^{\text{mod}}$  be the set of formulas that occur in  $\Delta$  or  $f$ , and let  $\mathcal{B}_\Omega = (\hat{Q}, \Sigma, \hat{\iota}, \hat{Q}, m_\Omega, \hat{\delta})$  be the dOCA along with the function  $\lambda$  according to Lemma 15.

We build the dOCA  $\mathcal{A}' = (Q', \Sigma, \iota', F', m_\Omega, \delta')$  as follows. Essentially, we perform a simple powerset construction for  $\mathcal{A}$ . Moreover, to eliminate modulo guards, we run  $\mathcal{B}_\Omega$  in parallel. Thus, the set of states is  $Q' = 2^Q \times \hat{Q}$ , with initial state  $\iota' = (\{\iota\}, \hat{\iota})$  and set of final states  $F' = \{(P, q) \in Q' \mid f(p) \in \lambda(q) \text{ for some } p \in P\}$ . Finally, the transition function is given by  $\delta'((P, q), k, \sigma) = (P', \hat{\delta}(q, k, \sigma))$  where  $P' = \{p' \mid (p, \varphi, \sigma, p') \in \Delta \cap (P \times \lambda(q) \times \{\sigma\} \times Q)\}$ . ◀

There is a “nondeterministic version” of Lemma 16, which does not perform a powerset construction but rather computes a nondeterministic OCA. The latter is then still of exponential size, but only wrt. to the number of arithmetic-progression formulas in  $\mathcal{A}$ .

With Theorem 3, it follows that nonemptiness for eOCAs can be solved in PSPACE. We do not know if this upper bound is tight.

Let  $\mathcal{A}$  be a dOCA with alphabet  $\Sigma$  and let  $w \in (\Sigma \times \mathbb{N})^*$ . By  $\rho_{\mathcal{A}}(w)$ , we denote the unique run of  $\mathcal{A}$  such that  $\text{vis}(\text{trace}(\rho_{\mathcal{A}}(w))) = \text{enc}(w)$ .

By the following observation, which follows directly from Lemma 12, it is justified to call any dOCA  $\mathcal{A}$  with  $L_{\text{vis}}(\mathcal{A}) \subseteq \text{Enc}_\Sigma$  *deterministic wrt. the observability semantics*:

► **Lemma 17.** *Let  $\mathcal{A}$  be a dOCA such that  $L_{\text{vis}}(\mathcal{A}) \subseteq \text{Enc}_\Sigma$ . For every word  $w \in (\Sigma \times \mathbb{N})^*$ , we have  $w \in L_{\text{obs}}(\mathcal{A})$  iff  $\rho_{\mathcal{A}}(w)$  is accepting.*

Altogether, we obtain that OCAs are determinizable wrt. the observability semantics.

► **Theorem 18** (determinizability). *Let  $\mathcal{A}$  be an OCA over  $\Sigma$ . We can compute, in exponential time, an  $m$ -dOCA  $\mathcal{A}'$  (with  $m$  only polynomial) such that  $L_{\text{obs}}(\mathcal{A}') = L_{\text{obs}}(\mathcal{A})$  and  $L_{\text{vis}}(\mathcal{A}') \subseteq \text{Enc}_\Sigma$ .*

**Proof.** Let  $\mathcal{A}$  be the given OCA. We apply Lemmas 14 and 16 to obtain a dOCA  $\tilde{\mathcal{A}}$  of exponential size such that  $L_{\text{obs}}(\tilde{\mathcal{A}}) = L_{\text{obs}}(\mathcal{A})$  and, for all  $w \in L_{\text{obs}}(\tilde{\mathcal{A}})$ , we have  $\text{enc}(w) \in L_{\text{vis}}(\tilde{\mathcal{A}})$ . We set  $\mathcal{A}' = \tilde{\mathcal{A}} \times \mathcal{B}_{\text{enc}}$  (cf. Lemmas 8 and 11) and obtain  $L_{\text{obs}}(\mathcal{A}') = L_{\text{obs}}(\mathcal{A})$  and  $L_{\text{vis}}(\mathcal{A}') \subseteq \text{Enc}_\Sigma$ . ◀

We conclude that OCAs are complementable wrt. the observability semantics:

► **Theorem 19** (complementability). *Let  $\mathcal{A}$  be an OCA with alphabet  $\Sigma$ . We can compute, in exponential time, a dOCA  $\bar{\mathcal{A}}$  such that  $L_{obs}(\bar{\mathcal{A}}) = (\Sigma \times \mathbb{N})^* \setminus L_{obs}(\mathcal{A})$ .*

**Proof.** We first transform  $\mathcal{A}$  into the dOCA  $\mathcal{A}' = \tilde{\mathcal{A}} \times \mathcal{B}_{enc}$  according to (the proof of) Theorem 18. Suppose  $\tilde{\mathcal{A}} = (Q, \Sigma, \iota, F, m, \delta)$ . Then, we set  $\bar{\mathcal{A}} = (Q, \Sigma, \iota, Q \setminus F, m, \delta) \times \mathcal{B}_{enc}$ . Note that  $\bar{\mathcal{A}}$  is indeed a dOCA and that  $L_{vis}(\bar{\mathcal{A}}) \subseteq \text{Enc}_\Sigma$ . For  $w \in (\Sigma \times \mathbb{N})^*$ , we have:

$$w \in L_{obs}(\bar{\mathcal{A}}) \stackrel{\text{Lem. 17}}{\iff} \rho_{\bar{\mathcal{A}}}(w) \text{ is accepting} \stackrel{(*)}{\iff} \rho_{\mathcal{A}'}(w) \text{ is not accepting} \stackrel{\text{Lem. 17}}{\iff} w \notin L_{obs}(\mathcal{A}')$$

Equivalence  $(*)$  holds as  $\rho_{\bar{\mathcal{A}}}(w)$  and  $\rho_{\mathcal{A}'}(w)$  have the same projection to the  $Q$ -component. ◀

Determinization and complementation of *extended* OCAs are a priori more expensive: Lemmas 9 and 14 only apply to OCAs so that one has to go through Lemma 16 first.

### 3.3 Universality and Inclusion Problem wrt. Observability Semantics

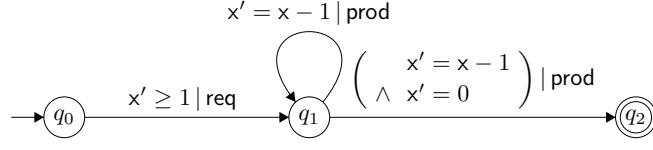
We are now ready to solve the universality and the inclusion problem for OCAs wrt. the observability semantics. The *universality problem* is defined as follows: Given an OCA  $\mathcal{A}$  over some alphabet  $\Sigma$ , do we have  $L_{obs}(\mathcal{A}) = (\Sigma \times \mathbb{N})^*$ ? The *inclusion problem* asks whether, given OCAs  $\mathcal{A}_1$  and  $\mathcal{A}_2$ , we have  $L_{obs}(\mathcal{A}_1) \subseteq L_{obs}(\mathcal{A}_2)$ .

► **Theorem 20.** *The universality problem and the inclusion problem for OCAs wrt. the observability semantics are PSPACE-complete. In both cases, PSPACE-hardness already holds when  $|\Sigma| = 1$ .*

**Proof.** To solve the universality problem for a given OCA  $\mathcal{A} = (Q, \Sigma, \iota, F, m, \Delta)$  in (non-deterministic) polynomial space, we apply the construction from Theorem 19 (and, in particular, Theorem 18) on the fly to obtain a dOCA  $\bar{\mathcal{A}}$  such that  $L_{obs}(\bar{\mathcal{A}}) = (\Sigma \times \mathbb{N})^* \setminus L_{obs}(\mathcal{A})$ . That is, we have to keep in memory a state of the form  $(P, q, r)$ , where  $P \subseteq Q$ ,  $q$  is the modulo-counting component (Lemma 15), and  $r$  is a state of  $\mathcal{B}_{enc}$  (Lemma 11). In addition, we will maintain a component for the current counter value. In fact, the latter can be supposed to be polynomially bounded (cf. [8] for a tight upper bound) in the size of  $\bar{\mathcal{A}}$ . The size of  $\bar{\mathcal{A}}$  is exponential in the size of  $\mathcal{A}$ , and so the required information can be stored in polynomial space. To compute a successor state of  $(P, q, r)$ , we first guess an operation  $\sigma \in \Sigma \cup \text{Op}$ . We then compute  $(P', q')$  according to the proof of Lemma 16 and update  $r$  to  $r'$  according to the type of  $\sigma$ . Note that this takes polynomial time only, since the function  $\lambda$  as required in Lemma 15 can be computed on the fly. Finally, the algorithm outputs “non-universal” when we find a final state of  $\bar{\mathcal{A}}$ .

For the inclusion problem, we rely on Proposition 13 and perform the determinization procedure on-the-fly for *both* of the given OCAs.

For the lower bound, we will restrict to the universality problem, since it is a special case of the inclusion problem. We reduce from the universality problem for ordinary finite automata, which is known to be PSPACE-complete [17]. If we suppose that  $\Sigma$  is part of the input, then there is a straightforward reduction, which essentially takes the (ordinary) finite automaton and adds self-looping increment/decrement transitions to each state. Assuming  $|\Sigma| = 1$ , the reduction is as follows. Let  $\mathcal{A}$  be a finite automaton over some finite alphabet  $\Gamma = \{a_0, \dots, a_{n-1}\}$ . We construct an OCA  $\mathcal{A}'$  over the singleton alphabet  $\Sigma$  such that  $L(\mathcal{A}) = \Gamma^*$  iff  $L(\mathcal{A}') = (\Sigma \times \mathbb{N})^*$ . The idea is to represent letter  $a_i$  by (counter) value  $i$ . To obtain  $\mathcal{A}'$ , an  $a_i$ -transition in  $\mathcal{A}$  is replaced with a gadget that nondeterministically outputs  $i$  or any other natural number strictly greater than  $n - 1$ . ◀



■ **Figure 3** A strong automaton over  $(\mathbb{N}, +1)$ .

#### 4 Relation with Strong Automata

In this section, we show that OCAs with counter observability are expressively equivalent to strong automata over  $(\mathbb{N}, +1)$  [9]. As the latter are descriptive in spirit, OCAs can thus be seen as their operational counterpart.

Let us first give a short account of monadic second-order (MSO) logic over  $(\mathbb{N}, +1)$  (see [23] for more details). We have infinite supplies of first-order variables, ranging over  $\mathbb{N}$ , and second-order variables, ranging over subsets of  $\mathbb{N}$ . The atomic formulas are *true*,  $x' = x + 1$ ,  $x' = x$ , and  $x \in X$  where  $x$  and  $x'$  are first-order variables and  $X$  is a second-order variable. Those formulas have the expected meaning. Further, MSO logic includes all boolean combinations, first-order quantification  $\exists x\Phi$ , and second-order quantification  $\exists X\Phi$  (with  $\Phi$  an MSO formula). The latter requires that there is a (possibly infinite) subset of  $\mathbb{N}$  satisfying  $\Phi$ . As abbreviations, we may also employ  $x' = x - 1$  and formulas of the form  $x' \in (x + c + d\mathbb{N})$ , where  $c, d \in \mathbb{N}$ . This does not change the expressive power of MSO logic.

In the following, we assume that  $x$  and  $x'$  are two distinguished first-order variables. We write  $\Phi(x, x')$  to indicate that the free variables of  $\Phi$  are among  $x$  and  $x'$ . If  $\Phi(x, x')$  is evaluated to true when  $x$  is interpreted as  $x \in \mathbb{N}$  and  $x'$  is interpreted as  $x' \in \mathbb{N}$ , then we write  $(x, x') \models \Phi$ . In fact, a transition of a strong automaton is labeled with a formula  $\Phi(x, x')$  and can only be executed if  $(x, x') \models \Phi$  where  $x$  and  $x'$  are the natural numbers read at the previous and the current position, respectively. Thus, two successive natural numbers in a word can be related explicitly in terms of an MSO formula.

► **Definition 21** ([9]). A *strong automaton* is a tuple  $\mathcal{S} = (Q, \Sigma, \iota, F, \Delta)$  where  $Q$  is the finite set of *states*,  $\Sigma$  is a nonempty finite alphabet,  $\iota \in Q$  is the *initial state*, and  $F \subseteq Q$  is the set of *final states*. Further,  $\Delta$  is a *finite* set of *transitions*, which are of the form  $(q, \Phi, a, q')$  where  $q, q' \in Q$  are the source and the target state,  $a \in \Sigma$ , and  $\Phi(x, x')$  is an MSO formula.

Similarly to an OCA,  $\mathcal{S}$  induces a relation  $\Longrightarrow_{\mathcal{S}} \subseteq \text{Conf}_{\mathcal{S}} \times (\Sigma \times \mathbb{N}) \times \text{Conf}_{\mathcal{S}}$ , where  $\text{Conf}_{\mathcal{S}} = Q \times \mathbb{N}$ . For  $(q, x), (q', x') \in \text{Conf}_{\mathcal{S}}$  and  $(a, y) \in \Sigma \times \mathbb{N}$ , we have  $(q, x) \xrightarrow{(a, y)}_{\mathcal{S}} (q', x')$  if  $y = x'$  and there is an MSO formula  $\Phi(x, x')$  such that  $(q, \Phi, a, q') \in \Delta$  and  $(x, x') \models \Phi$ . A *run* of  $\mathcal{S}$  on  $w = (a_1, x_1) \dots (a_n, x_n) \in (\Sigma \times \mathbb{N})^*$  is a sequence  $\rho = (q_0, x_0) \xrightarrow{(a_1, x_1)}_{\mathcal{S}} (q_1, x_1) \xrightarrow{(a_2, x_2)}_{\mathcal{S}} \dots \xrightarrow{(a_n, x_n)}_{\mathcal{S}} (q_n, x_n)$  such that  $q_0 = \iota$  and  $x_0 = 0$ . It is *accepting* if  $q_n \in F$ .

The language  $L(\mathcal{S}) \subseteq (\Sigma \times \mathbb{N})^*$  of  $\mathcal{S}$  is defined as the set of words  $w \in (\Sigma \times \mathbb{N})^*$  such that there is an accepting run of  $\mathcal{S}$  on  $w$ .

► **Example 22.** We refer to the OCA  $\mathcal{A}$  from Example 2. Figure 3 depicts a strong automaton  $\mathcal{S}$  such that  $L(\mathcal{S}) = L_{\text{obs}}(\mathcal{A}) = \{(\text{req}, n)(\text{prod}, n - 1)(\text{prod}, n - 2) \dots (\text{prod}, 0) \mid n \geq 1\}$ .

In fact, we can transform any OCA into an equivalent strong automaton preserving the observability semantics, and vice versa:

► **Theorem 23.** *Let  $L \subseteq (\Sigma \times \mathbb{N})^*$ . There is an OCA  $\mathcal{A}$  such that  $L_{\text{obs}}(\mathcal{A}) = L$  iff there is a strong automaton  $\mathcal{S}$  such that  $L(\mathcal{S}) = L$ .*

**Proof.** “ $\implies$ ”: Using the following observation, we can directly transform an OCA into a strong automaton: For all states  $q$  and  $q'$  of the given OCA  $\mathcal{A}$ , there is an MSO formula  $\Phi_{q,q'}(x, x')$  such that, for all  $x, x' \in \mathbb{N}$ , we have  $(x, x') \models \Phi_{q,q'}$  iff  $(q, x) (\xrightarrow{\uparrow}_{\mathcal{A}} \cup \xrightarrow{\downarrow}_{\mathcal{A}})^* (q', x')$ . The existence of  $\Phi_{q,q'}$  can be shown using a two-way automaton over infinite words [21], which simulates  $\mathcal{A}$  and can be translated into an MSO formula [21, 23].

More precisely, given  $q, q'$ , we build a two-way automaton  $\mathcal{T}_{q,q'}$  over the alphabet  $2^{\{\$, \$'\}}$ . The idea is that word positions represent counter values (the first position marking 0, the second 1, and so on), and  $\$$  and  $\$'$  represent  $x$  and  $x'$ , respectively. Thus, we are only interested in words in which  $\$$  and  $\$'$  each occur exactly once. Clearly, this is a regular property. At the beginning,  $\mathcal{T}_{q,q'}$  goes to the position carrying  $\$$ . It then simulates  $\mathcal{A}$  starting in  $q$ , and it accepts if it is on the position carrying  $\$'$  and in state  $q'$ . The simulation itself is straightforward: Counter increments and decrements of an OCA are simulated by going one step to the left or to the right, respectively, and a zero test simply checks whether the automaton is at the first position of the word. Note that  $\mathcal{T}_{q,q'}$  checks for the markers  $\$$  and  $\$'$  only at the beginning and at the end of an execution, but not during the actual simulation of  $\mathcal{A}$ . Let  $w = z_0 z_1 z_2 \dots \in (2^{\{\$, \$'\}})^\omega$  and  $x, x' \in \mathbb{N}$  be unique positions such that  $\$ \in z_x$  and  $\$' \in z_{x'}$ . Then,  $w$  is accepted by  $\mathcal{T}_{q,q'}$  iff  $(q, x) (\xrightarrow{\uparrow}_{\mathcal{A}} \cup \xrightarrow{\downarrow}_{\mathcal{A}})^* (q', x')$ .

It is well known that two-way word automata are expressively equivalent to one-way automata (cf. [21]). Therefore, by Büchi’s theorem, the word language accepted by  $\mathcal{T}_{q,q'}$  can be translated into a corresponding MSO formula without free variables but with subformulas of the form “position  $y$  carries  $\$$ ” and “position  $y$  carries  $\$'$ ” [23]. We replace the latter two by  $y = x$  and  $y = x'$ , respectively, and finally obtain  $\Phi_{q,q'}$  as required.

“ $\longleftarrow$ ”: We will transform a strong automaton  $\mathcal{S}$  into an equivalent OCA with super transitions. A super transition can perform counter operations of the form  $+\psi$  or  $-\psi$  where  $\psi \in \text{Guards}^{\text{mod}}$ . Operation  $+\psi$  allows the counter value to be increased by  $n \in \mathbb{N}$  provided  $n \models \psi$ . Similarly,  $-\psi$  allows the counter value to be decreased by  $n$  if  $n \models \psi$ .

► **Definition 24** (OCA with super transitions). An OCA with super transitions is a tuple  $\mathcal{A} = (Q, \Sigma, \iota, F, \Delta)$  where  $Q, \Sigma, \iota, F$  are like in an OCA and  $\Delta$  is the finite transition relation. A transition is of the form  $(q, \varphi, op, a, \varphi', q')$  where  $q, q' \in Q$  are the source and the target state,  $\varphi, \varphi' \in \text{Guards}^{\text{mod}}$  are guards checking the original and the modified counter value, respectively,  $a \in \Sigma$  is the output letter, and  $op$  is of the form  $+\psi$  or  $-\psi$  where  $\psi \in \text{Guards}^{\text{mod}}$ .

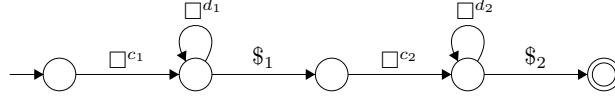
We define a global transition relation  $\implies_{\mathcal{A}} \subseteq \text{Conf}_{\mathcal{A}} \times (\Sigma \times \mathbb{N}) \times \text{Conf}_{\mathcal{A}}$  where, as usual,  $\text{Conf}_{\mathcal{A}} = Q \times \mathbb{N}$ . For  $(q, x), (q', x') \in \text{Conf}_{\mathcal{A}}$  and  $(a, y) \in \Sigma \times \mathbb{N}$ , we have  $(q, x) \xrightarrow{(a,y)}_{\mathcal{A}} (q', x')$  if  $y = x'$  and there are  $\varphi, \varphi', \psi \in \text{Guards}^{\text{mod}}$  such that  $x \models \varphi$ ,  $x' \models \varphi'$ , and one of the following holds:

- $x \leq x'$  and  $(q, \varphi, +\psi, a, \varphi', q') \in \Delta$  and  $x' - x \models \psi$ , or
- $x \geq x'$  and  $(q, \varphi, -\psi, a, \varphi', q') \in \Delta$  and  $x - x' \models \psi$ .

With this, the language  $L(\mathcal{A}) \subseteq (\Sigma \times \mathbb{N})^*$  is defined in the expected way, like for strong automata.

It is easily seen that  $\mathcal{A}$  can be translated into an eOCA  $\mathcal{A}'$  such that  $L_{\text{obs}}(\mathcal{A}') = L(\mathcal{A})$ : For every  $\psi \in \text{Guards}^{\text{mod}}$ , the set  $\{\square^n \mid n \in \llbracket \psi \rrbracket\}$  is a regular language over the unary alphabet  $\{\square\}$ . Thus, counter operations of the form  $+\psi$  or  $-\psi$  can be simulated by a finite-state gadget. Essentially, we take a finite automaton for  $\{\square^n \mid n \in \llbracket \psi \rrbracket\}$  and replace  $\square$  by  $\uparrow$  or, respectively,  $\downarrow$ . It will thus be enough to translate a strong automaton into an OCA with super transitions.





■ **Figure 4** Finite automaton for  $L_{\Phi}^+(c_1, d_1, c_2, d_2)$ .

Next, we demonstrate why super transitions are indeed useful to emulate an MSO formula  $\Phi(x, x')$ . Using Büchi's theorem (cf. [23]), we can transform  $\Phi$  into *finite automata*  $\mathcal{B}_{\Phi}^+$  and  $\mathcal{B}_{\Phi}^-$  recognizing the following regular languages over the alphabet  $\{\square, \$_1, \$_2\}$ :

- $L(\mathcal{B}_{\Phi}^+) = \{\square^x \$_1 \square^y \$_2 \mid x, y \in \mathbb{N} \text{ such that } (x, x+y) \models \Phi\}$
- $L(\mathcal{B}_{\Phi}^-) = \{\square^x \$_2 \square^y \$_1 \mid x, y \in \mathbb{N} \text{ such that } (x+y, x) \models \Phi\}$

Similarly to  $\$$  and  $\$'$  in the other proof direction, the positions of  $\$_1$  and  $\$_2$  in a word from  $L(\mathcal{B}_{\Phi}^+) \cup L(\mathcal{B}_{\Phi}^-)$  encode an interpretation of the free variables  $x$  and, respectively,  $x'$  that makes  $\Phi$  true. Note that  $L(\mathcal{B}_{\Phi}^+)$  can be written as a finite union of sets

$$L_{\Phi}^+(c_1, d_1, c_2, d_2) := \{\square^x \$_1 \square^y \$_2 \mid x \in [c_1 + d_1\mathbb{N}] \text{ and } y \in [c_2 + d_2\mathbb{N}]\}$$

with  $c_1, d_1, c_2, d_2 \in \mathbb{N}$ . This is achieved by determinizing  $\mathcal{B}_{\Phi}^+$  and splitting it into components as illustrated in Figure 4 (cf. also [26] for a polynomial transformation). Similarly,  $L(\mathcal{B}_{\Phi}^-)$  is the finite union of sets of the form

$$L_{\Phi}^-(c_1, d_1, c_2, d_2) := \{\square^x \$_2 \square^y \$_1 \mid x \in [c_1 + d_1\mathbb{N}] \text{ and } y \in [c_2 + d_2\mathbb{N}]\}.$$

In other words, there are finite sets  $D_{\Phi}^+, D_{\Phi}^- \subseteq \mathbb{N}^4$  such that

- $L(\mathcal{B}_{\Phi}^+) = \bigcup_{(c_1, d_1, c_2, d_2) \in D_{\Phi}^+} L_{\Phi}^+(c_1, d_1, c_2, d_2)$  and
- $L(\mathcal{B}_{\Phi}^-) = \bigcup_{(c_1, d_1, c_2, d_2) \in D_{\Phi}^-} L_{\Phi}^-(c_1, d_1, c_2, d_2)$ .

We now turn to the actual translation of a strong automaton  $\mathcal{S} = (Q, \Sigma, \iota, F, \Delta)$  into an OCA with super transitions  $\mathcal{A} = (Q, \Sigma, \iota, F, \Delta')$  such that  $L(\mathcal{A}) = L(\mathcal{S})$ . Note that the only change is in the transition relation: For all  $(q, \Phi, a, q') \in \Delta$  and  $(c_1, d_1, c_2, d_2) \in D_{\Phi}^+$ ,  $\Delta'$  contains  $(q, c_1 + d_1\mathbb{N}, +(c_2 + d_2\mathbb{N}), a, \text{true}, q')$ . Moreover, for all  $(q, \Phi, a, q') \in \Delta$  and  $(c_1, d_1, c_2, d_2) \in D_{\Phi}^-$ ,  $\Delta'$  contains  $(q, \text{true}, -(c_2 + d_2\mathbb{N}), a, c_1 + d_1\mathbb{N}, q')$ . This concludes the construction of  $\mathcal{A}$ .

To prove  $L(\mathcal{A}) = L(\mathcal{S})$ , it is enough to show that, for all configurations  $(q, x), (q', x') \in \text{Conf}_{\mathcal{A}}$  and all  $a \in \Sigma$ , the following are equivalent:

- (1)  $(q, x) \xrightarrow{(a, x')}_{\mathcal{S}} (q', x')$
- (2)  $(q, x) \xrightarrow{(a, x')}_{\mathcal{A}} (q', x')$

Suppose (1) holds. There is an MSO formula  $\Phi$  such that  $(q, \Phi, a, q') \in \Delta$  and  $(x, x') \models \Phi$ . We distinguish two (not necessarily disjoint) cases:

- Suppose  $x \leq x'$ . By  $(x, x') \models \Phi$ , we have  $\square^x \$_1 \square^{x'-x} \$_2 \in L(\mathcal{B}_{\Phi}^+)$ . Thus, there is  $(c_1, d_1, c_2, d_2) \in D_{\Phi}^+$  such that  $\square^x \$_1 \square^{x'-x} \$_2 \in L_{\Phi}^+(c_1, d_1, c_2, d_2)$ . The latter implies  $x \models c_1 + d_1\mathbb{N}$  and  $x' - x \models c_2 + d_2\mathbb{N}$ . Since we also have  $(q, c_1 + d_1\mathbb{N}, +(c_2 + d_2\mathbb{N}), a, \text{true}, q') \in \Delta'$ , (2) holds as well.
- Now, suppose  $x \geq x'$ . Then, by  $(x, x') \models \Phi$ , we have  $\square^{x'} \$_2 \square^{x-x'} \$_1 \in L(\mathcal{B}_{\Phi}^-)$ . Thus, there is  $(c_1, d_1, c_2, d_2) \in D_{\Phi}^-$  such that  $\square^{x'} \$_2 \square^{x-x'} \$_1 \in L_{\Phi}^-(c_1, d_1, c_2, d_2)$ . This implies  $x' \models c_1 + d_1\mathbb{N}$  and  $x - x' \models c_2 + d_2\mathbb{N}$ . Moreover,  $(q, \text{true}, -(c_2 + d_2\mathbb{N}), a, c_1 + d_1\mathbb{N}, q') \in \Delta'$ . We conclude that (2) holds.

Towards the other direction, suppose that (2) is true. Again, we will distinguish two (not necessarily disjoint) cases:



- Suppose  $x \leq x'$  and suppose there is  $(q, c_1 + d_1\mathbb{N}, +(c_2 + d_2\mathbb{N}), a, \text{true}, q') \in \Delta'$  such that  $x \models c_1 + d_1\mathbb{N}$  and  $x' - x \models c_2 + d_2\mathbb{N}$ . There is an MSO formula  $\Phi$  such that  $(q, \Phi, a, q') \in \Delta$  and  $(c_1, d_1, c_2, d_2) \in D_{\Phi}^+$ . By  $x \models c_1 + d_1\mathbb{N}$  and  $x' - x \models c_2 + d_2\mathbb{N}$ , we have  $\Box^x \$1 \Box^{x'-x} \$2 \in L_{\Phi}^+(c_1, d_1, c_2, d_2) \subseteq L(\mathcal{B}_{\Phi}^+)$ . Thus,  $(x, x') \models \Phi$ . We deduce that (1) holds.
- Assume  $x \geq x'$  and suppose there is a transition  $(q, \text{true}, -(c_2 + d_2\mathbb{N}), a, c_1 + d_1\mathbb{N}, q') \in \Delta'$  such that  $x - x' \models c_2 + d_2\mathbb{N}$  and  $x' \models c_1 + d_1\mathbb{N}$ . There is  $\Phi$  such that  $(q, \Phi, a, q') \in \Delta$  and  $(c_1, d_1, c_2, d_2) \in D_{\Phi}^-$ . Since  $x - x' \models c_2 + d_2\mathbb{N}$  and  $x' \models c_1 + d_1\mathbb{N}$ , we have  $\Box^{x'} \$2 \Box^{x-x'} \$1 \in L_{\Phi}^-(c_1, d_1, c_2, d_2) \subseteq L(\mathcal{B}_{\Phi}^-)$ . This implies  $(x, x') \models \Phi$ . Thus, (1) holds.

This concludes the correctness proof of  $\mathcal{A}$ . Finally, recall that one can easily transform  $\mathcal{A}$  into an OCA whose observability semantics coincides with  $L(\mathcal{A})$ . ◀

## 5 Conclusion

The observability semantics opens several directions for follow-up work. We may carry it over to other classes of infinite-state systems such as Petri nets. Are there infinite-state restrictions of Petri nets other than 1-VASS whose observability semantics is robust?

A direct application of our results is that the language  $L \subseteq (\Sigma \times \mathbb{N})^*$  of an OCA with observability semantics/strong automaton is learnable (in the sense of Angluin [4]) in terms of a visibly one-counter automaton for  $\text{enc}(L)$  [18]. It would be worthwhile to transfer results on visibly one-counter/pushdown automata that concern Myhill-Nerode congruences or minimization [2, 7].

Another interesting question is to which extent we can relax the requirement that the counter value be output with every letter  $a \in \Sigma$ . It may indeed be possible to deal with a bounded number of  $\Sigma$ -transitions between any two counter outputs. Note that there have been relaxations of the visibility condition in pushdown automata, albeit preserving closure under boolean operations [19].

**Acknowledgments.** The author is grateful to C. Aiswarya, Stefan Göller, Christoph Haase, and Arnaud Sangnier for numerous helpful discussions and pointers to the literature.

---

## References

- 1 R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- 2 R. Alur, V. Kumar, P. Madhusudan, and M. Viswanathan. Congruences for visibly pushdown languages. In *Proceedings of ICALP'05*, volume 3580 of *Lecture Notes in Computer Science*, pages 1102–1114, 2005.
- 3 R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3):1–43, 2009.
- 4 D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
- 5 V. Bárány, C. Löding, and O. Serre. Regularity problems for visibly pushdown languages. In *Proceedings of STACS'06*, volume 3884 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2006.
- 6 A. Bès. An application of the Feferman-Vaught theorem to automata and logics for words over an infinite alphabet. *Logical Methods in Computer Science*, 4(1), 2008.
- 7 P. Chervet and I. Walukiewicz. Minimizing variants of visibly pushdown automata. In *Proceedings of MFCS'07*, volume 4708 of *Lecture Notes in Computer Science*, pages 135–146, 2007.

- 8 D. Chistikov, W. Czerwiński, P. Hofman, M. Pilipczuk, and M. Wehar. Shortest paths in one-counter systems. In *Proceedings of FoSSaCS'16*, volume 9634 of *Lecture Notes in Computer Science*, pages 462–478. Springer, 2016.
- 9 C. Czyba, C. Spinrath, and W. Thomas. Finite automata over infinite alphabets: Two models with transitions for local change. In *Proceedings of DLT'15*, volume 9168 of *Lecture Notes in Computer Science*, pages 203–214. Springer, 2015.
- 10 L. D'Antoni and M. Veanes. Minimization of symbolic automata. In *Proceedings of POPL'14*, pages 541–554. ACM, 2014.
- 11 S. Göller, R. Mayr, and A. Widjaja To. On the computational complexity of verifying one-counter processes. In *Proceedings of LICS'09*, pages 235–244. IEEE Computer Society Press, 2009.
- 12 S. A. Greibach. An infinite hierarchy of context-free languages. *Journal of the ACM*, 16(1):91–106, 1969.
- 13 C. Haase, J. Ouaknine, and J. Worrell. Relating reachability problems in timed and counter automata. *Fundamenta Informaticae*, 143(3-4):317–338, 2016.
- 14 O. H. Ibarra. Restricted one-counter machines with undecidable universe problems. *Mathematical Systems Theory*, 13:181–186, 1979.
- 15 P. Jančar. Decidability of bisimilarity for one-counter processes. *Information and Computation*, 158(1):1–17, 2000.
- 16 K. Mehlhorn. Pebbling mountain ranges and its application of DCFL-recognition. In *Proceedings of ICALP'80*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980.
- 17 A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory*, pages 125–129, 1972.
- 18 D. Neider and C. Löding. Learning visibly one-counter automata in polynomial time. Technical Report AIB-2010-02, RWTH Aachen, January 2010.
- 19 D. Nowotka and J. Srba. Height-deterministic pushdown automata. In *Proceedings of MFCS'07*, volume 4708 of *Lecture Notes in Computer Science*, pages 125–134. Springer, 2007.
- 20 R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.
- 21 J.-P. Pécuchet. Automates boustrophédon et mots infinis. *Theoretical Computer Science*, 35:115–122, 1985.
- 22 J. Srba. Visibly pushdown automata: From language equivalence to simulation and bisimulation. In *Proceedings of CSL'06*, volume 4207 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2006.
- 23 W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.
- 24 L. G. Valiant and M. S. Paterson. Deterministic one-counter automata. *Journal of Computer and System Sciences*, 10(3):340–350, 1975.
- 25 J. van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21(3):237–252, 1978.
- 26 A. Widjaja To. Unary finite automata vs. arithmetic progressions. *Inf. Process. Lett.*, 109(17):1010–1014, 2009.

# Strong Parameterized Deletion: Bipartite Graphs

Ashutosh Rai<sup>1</sup> and M. S. Ramanujan<sup>2</sup>

1 The Institute of Mathematical Sciences, HBNI, Chennai, India

ashutosh@imsc.res.in

2 Vienna Institute of Technology, Vienna, Austria

ramanujan@ac.tuwien.ac.at

---

## Abstract

The purpose of this article is two fold: (a) to formally introduce a stronger version of graph deletion problems; and (b) to study this version in the context of bipartite graphs. Given a family of graphs  $\mathcal{F}$ , a typical instance of parameterized graph deletion problem consists of an undirected graph  $G$  and a positive integer  $k$  and the objective is to check whether we can delete at most  $k$  vertices (or  $k$  edges) such that the resulting graph belongs to  $\mathcal{F}$ . Another version that has been recently studied is the one where the input contains two integers  $k$  and  $\ell$  and the objective is to check whether we can delete at most  $k$  vertices and  $\ell$  edges such that the resulting graph belongs to  $\mathcal{F}$ . In this paper, we propose and initiate the study of a more general version which we call *strong deletion*. In this problem, given an undirected graph  $G$  and positive integers  $k$  and  $\ell$ , the objective is to check whether there exists a vertex subset  $S$  of size at most  $k$  such that *each connected component* of  $G - S$  can be transformed into a graph in  $\mathcal{F}$  by deleting at most  $\ell$  edges. In this paper we study this stronger version of deletion problems for the class of bipartite graphs. In particular, we study STRONG BIPARTITE DELETION, where given an undirected graph  $G$  and positive integers  $k$  and  $\ell$ , the objective is to check whether there exists a vertex subset  $S$  of size at most  $k$  such that each connected component of  $G - S$  can be made bipartite by deleting at most  $\ell$  edges. While fixed-parameter tractability when parameterizing by  $k$  or  $\ell$  alone is unlikely, we show that this problem is fixed-parameter tractable (FPT) when parameterized by both  $k$  and  $\ell$ .

**1998 ACM Subject Classification** G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

**Keywords and phrases** fixed parameter tractable, bipartite-editing, recursive understanding

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.21

## 1 Introduction

Graph-modification by either deleting vertices or deleting edges or adding edges such that the resulting graph satisfies certain properties or becomes a member of some well-understood graph class is one of the basic problems in graph theory and graph algorithms. However, most of these problems are NP-complete [18, 26] and thus they are subjected to intensive study in algorithmic paradigms that are meant for coping with NP-completeness [9, 10, 21, 22]. These paradigms among others include applying restrictions on inputs, approximation algorithms and parameterized complexity. The goal of this paper is to introduce a ‘stronger’ notion of graph deletion in the realm of parameterized complexity and illustrate the difficulties that arise when considering the family of bipartite graphs and provide an approach to obtain fixed-parameter tractability by overcoming these difficulties.

A typical instance of parameterized graph deletion is of the following form. Let  $\mathcal{F}$  be a family of graphs – such as edgeless graphs, forests, cluster graphs, chordal graphs, interval graphs, bipartite graphs, split graphs or planar graphs. The deletion problem corresponding to  $\mathcal{F}$  is formally stated as follows.



© Ashutosh Rai and M. S. Ramanujan;

licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

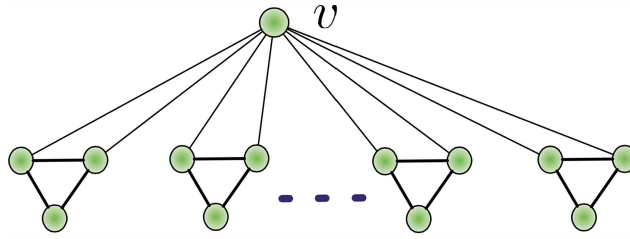
Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 21; pp. 21:1–21:14

Leibniz International Proceedings in Informatics



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

21:2 Strong Parameterized Deletion: Bipartite Graphs



■ **Figure 1** An example showing the strength of the new parameter for editing problems.

$\mathcal{F}$ -VERTEX (EDGE) DELETION **Parameters:**  $k$   
**Input:** An undirected graph  $G$  and a non-negative integer  $k$ .  
**Question:** Does there exist  $S \subseteq V(G)$  (or  $S \subseteq E(G)$ ), such that  $|S| \leq k$  and  $G - S$  is in  $\mathcal{F}$ ?

In other words, given a graph  $G$ , can we delete at most  $k$  vertices or  $k$  edges such that the resulting graph belongs to  $\mathcal{F}$ ? An algorithm for  $\mathcal{F}$ -VERTEX (EDGE) DELETION that runs in time  $f(k) \cdot |V(G)|^{\mathcal{O}(1)}$  is called *fixed-parameter tractable* (FPT) algorithm and the problem itself is said to be FPT.

The study of parameterized graph deletion problems together with their various restrictions and generalizations has been an extremely active sub-area over the last few years. In fact, just over the course of the last couple of years there have been results on parameterized algorithms for CHORDAL EDITING [6], UNIT VERTEX (EDGE) DELETION [3], INTERVAL VERTEX (EDGE) DELETION [5, 4], PLANAR  $\mathcal{F}$  DELETION [9, 17], PLANAR VERTEX DELETION [14], BLOCK GRAPH DELETION [16] and SIMULTANEOUS FEEDBACK VERTEX SET [1]. Several known parameterized algorithms for  $\mathcal{F}$ -EDGE DELETION or the version where the objective is to delete  $k$  vertices and  $\ell$  edges utilize the fact that given a yes-instance  $(G, k)$  or  $(G, k, \ell)$  to the problem, there exists a vertex set  $S^*$  of  $V(G)$  of size  $k^* = k + \ell$  such that  $G - S^*$  belongs to  $\mathcal{F}$ . Clearly, this is true for any hereditary family of graphs; that is, if  $G \in \mathcal{F}$  then all its induced subgraphs belong to  $\mathcal{F}$ . Thus, if the corresponding  $\mathcal{F}$ -VERTEX DELETION is FPT then we can apply this algorithm and find a vertex subset  $S^*$  of size at most  $k$  such that  $G - S^* \in \mathcal{F}$ . Having the set  $S^*$  allows one to infer numerous structural properties of the input which can then be utilized in non-trivial ways to solve the original  $\mathcal{F}$ -VERTEX (EDGE) DELETION problem. However, the existence of such a set is no longer guaranteed in the proposed stronger version of this problem.

Let  $\mathcal{F}$  be a polynomial-time recognizable family of graphs; that is, given a graph  $G$ , in polynomial time we can decide whether  $G$  belongs to  $\mathcal{F}$ . For a fixed integer  $\ell$ , let  $\mathcal{F} + \ell e$  denote the class of graphs that can be obtained by adding  $\ell$  edges to a graph in  $\mathcal{F}$  [2]. Furthermore, suppose that  $\mathcal{F}$ -EDGE DELETION is FPT with running time  $\mathcal{O}(g(\ell) \cdot n^c)$ . Here,  $n = |V(G)|$  and  $c$  is a fixed constant. That is, for any fixed integer  $\ell$ ,  $\mathcal{F} + \ell e$  can be recognized in time  $\mathcal{O}(n^c)$ . The STRONG  $\mathcal{F}$ -DELETION problem is defined as follows.

STRONG  $\mathcal{F}$ -DELETION **Parameters:**  $k, \ell$   
**Input:** An undirected graph  $G$  and non-negative integers  $k$  and  $\ell$ .  
**Question:** Does there exist  $S_1 \subseteq V(G)$  such that  $|S_1| \leq k$  and every *connected component* of  $G - S_1$  belongs to  $\mathcal{F} + \ell e$ ?

A close inspection easily shows that STRONG  $\mathcal{F}$ -DELETION is much stronger than  $\mathcal{F}$ -DELETION. For example, let  $\mathcal{F}$  be the family of bipartite graphs and consider the graph

$G$  depicted in Figure 1. The graph  $G$  has  $\frac{n-1}{3}$  vertex disjoint triangles and the vertex  $v$  is adjacent to two vertices from each of the triangles. Clearly, after deleting  $v$  every connected component can be made bipartite by deleting exactly one edge. Thus, in the traditional sense this is a yes-instance of BIPARTITE DELETION with parameters  $(1, \frac{n-1}{3})$ ; however this is already a yes-instance of STRONG BIPARTITE DELETION with parameters  $(1, 1)$ . Thus, these problems seem much harder than traditional editing problems as the parameters are much smaller.

An alternate viewpoint is that when  $G$  has a vertex set  $S$  of size at most  $k$  such that every connected component of  $G - S$  is in  $\mathcal{F} + \ell e$  then  $S$  can be considered a *modulator* into the graph class where every connected component of the graph belongs to  $\mathcal{F} + \ell e$ . While modulators to various polynomial-time recognizable graph classes have been studied in a very systematic way [8, 9, 13], the same is not true of modulators to NP-complete graph classes. Studying the STRONG  $\mathcal{F}$ -DELETION problem on the other hand allows us to do precisely this. In fact, it is not even necessary that the class  $\mathcal{F}$  is a conventional graph class and instead can be the ‘composition’ of various graph classes. For instance  $\mathcal{F}$  could be defined as the set of all graphs where each connected component is either chordal *or* a bipartite graph. The computation of such modulators is a problem with several algorithmic applications. For instance, in a recent work, Ganian et al. [11] used a similar notion in order to design algorithms for the classic CONSTRAINT SATISFACTION PROBLEM.

In this article we study the STRONG  $\mathcal{F}$ -DELETION, when  $\mathcal{F}$  is the family of bipartite graphs. Henceforth,  $\mathcal{F}$  denotes the family of bipartite graphs. We call a graph  $G$ , *ell-pseudobipartite*, if every connected component of  $G$  belongs to  $\mathcal{F} + \ell e$ . The problem we study is as follows.

STRONG BIPARTITE DELETION

**Parameters:**  $k, \ell$

**Input:** An undirected graph  $G$  and non-negative integers  $k$  and  $\ell$ .

**Question:** Does there exist  $S \subseteq V(G)$  such that  $|S| \leq k$  and every connected component of  $G - S$  belongs to  $\mathcal{F} + \ell e$ ?

We refer to the set  $S$  as the *solution* for this instance. The primary reason behind the selection of the family of bipartite graphs for our study is that the problems where we are required to delete vertices and/or edges to obtain a bipartite graph are some of the most basic and well studied problems in parameterized complexity and studying these problems has led to the discovery of several new techniques and tools. These problems are called ODD CYCLE TRANSVERSAL and EDGE BIPARTIZATION in literature and the algorithms with best dependence on the parameter have running time  $\mathcal{O}(2.3146^k n^{\mathcal{O}(1)})$  and  $\mathcal{O}(1.977^k n^{\mathcal{O}(1)})$ , respectively [19, 24]. We would like to add that the problem is unlikely to be FPT when parameterized by  $k$  or  $\ell$  alone, as it would imply polynomial time algorithms for EDGE BIPARTIZATION and ODD CYCLE TRANSVERSAL respectively. It is also important to mention that strong deletion to the class of forests, STRONG FOREST DELETION, was studied in related but different context in [23]. In that work, the authors used the concept of graph minors and Courcelle’s theorem to show that the strong deletion problem is FPT. As is the case with algorithms using Courcelle’s theorem, the dependence on the parameter is very high.

**Our Result and Methodology.** Our main result is the following theorem.

► **Theorem 1.** STRONG BIPARTITE DELETION is FPT.

The first big obstacle that needs to be overcome is the fact that the input graph can have a minimum odd cycle transversal of unbounded size. The first part of our algorithm is devoted

to overcoming this obstacle. We utilise the technique of iterative compression to reduce it to a bounded number of equivalent instances each having an odd cycle transversal of size  $\text{poly}(k, \ell)$ . Then we use the recursive understanding technique introduced by Grohe et al. [12] (also see [7]) to first find a small separator in the graph which separates the graph into two parts, each of sufficiently large size and then recursively solve a ‘border’ version of the same problem on one of the two sides. The subroutines that we use to compute the separators are those of Chitnis et al. [7] who built upon the work of Kawarabayashi and Thorup [15]. The border version of the problem is a generalization which also incorporates a special bounded set of vertices, called terminals. During the course of our algorithm, we will attempt to solve the border problem on various subgraphs of the input graph. The objective in the border problem is to find a bounded set of vertices contained within a particular subgraph such that any vertex in this subgraph *not* in the computed set is not required in *any* solution for the given instance irrespective of the vertices chosen outside this subgraph.

Given the output of the border problem, the standard approach is to either delete the remaining vertices or simply ‘bypass’ these vertices. In our case, no such simple reduction seems likely. However, we show that by blowing up the size of the computed set by a function of the parameter, we can use a ‘parity preserving’ bypassing operation to get rid of the remaining vertices.

This leaves us with the base case of the recursion, that is when we are unable to find a small separator. At this stage, we know that the graph has a bounded sized odd cycle transversal *and* has a highly connected structure. Interestingly, even this seemingly significant structural information regarding the input does not seem enough to imply a straightforward algorithm. Instead, we compute an oct solution and design a branching rule that has as its base case, the case when the oct is not separated by the solution. Here, we rephrase this problem as a MIXED MULTIWAY CUT-UNCUT (MMCU) problem and invoke the result in [25] to solve it. In the MMCU problem, the input is a multigraph  $G$ , integers  $k$  and  $\ell$ , a set of terminals  $T \subseteq V(G)$ , and equivalence relation  $\mathcal{R}$  on the set  $T$ . The objective is to output a solution  $\mathcal{X} = (X, F)$  such that  $X \subseteq (V(G) \setminus T)$ ,  $F \subseteq E(G)$ ,  $|X| \leq k$ ,  $|F| \leq \ell$  and for all  $u, v \in T$ ,  $u$  and  $v$  belong to the same connected component of  $G - \mathcal{X}$  if and only if  $(u, v) \in \mathcal{R}$  and  $\perp$  if no such solution exists. The structure of our presentation follows that of Chitnis et al. [7] for the most part, except for the high connectivity phase.

Due to space constraints, many proofs of results are omitted or shortened, and will appear in the full version of the paper.

## 2 Preliminaries

**Notations and Definitions:** For a graph  $G$ , we denote the set of vertices of the graph by  $V(G)$  and the set of edges of the graph by  $E(G)$ . We denote  $|V(G)|$  and  $|E(G)|$  by  $n$  and  $m$  respectively, where the graph is clear from context. For a set  $S \subseteq V(G)$ , the *subgraph of  $G$  induced by  $S$*  is denoted by  $G[S]$  and it is defined as the subgraph of  $G$  with vertex set  $S$  and edge set  $\{(u, v) \in E(G) : u, v \in S\}$  and the subgraph obtained after deleting  $S$  is denoted as  $G - S$ . For  $F \subseteq E(G)$ , by  $V(F)$  we denote the set  $\{v \mid \exists u \text{ such that } (u, v) \in F\}$ . For a tuple  $\mathcal{X} = (X, F)$  such that  $X \subseteq V(G)$  and  $F \subseteq E(G)$ , by  $G - \mathcal{X}$  we denote the graph  $G' = (V(G) \setminus X, E(G - X) \setminus F)$ . All vertices adjacent to a vertex  $v$  are called neighbours of  $v$  and the set of all such vertices is called *open neighbourhood* of  $v$ , denoted by  $N_G(v)$ . For a set of vertices  $S \subseteq V(G)$ , we define  $N_G(S) = (\cup_{v \in S} N(v)) \setminus S$ . We drop the subscript  $G$  when the graph is clear from the context.

An *oct* of a graph  $G$ , is a set  $X \subseteq V(G)$  such that  $G - X$  is bipartite. Similarly, an *edge-oct* of a graph  $G$  is set  $F \subseteq E(G)$  such that the graph  $G - F$  is bipartite. We call a

graph  $\ell$ -pseudobipartite, if each of the connected components of  $G$  has an edge-cut of size at most  $\ell$ . An  $\ell$ -pseudobipartite deletion set of a graph  $G$  is a set  $X \subseteq V(G)$  such that  $G - X$  is  $\ell$ -pseudobipartite. It is easy to see that  $\ell$ -pseudobipartite graphs can be recognized in time  $\mathcal{O}(1.977^k n^{\mathcal{O}(1)})$  using the algorithm for EDGE BIPARTIZATION given in [24]. Now we state the definitions of good node separations and flower separations from [7]. Then we state the lemmas that talk about the running time to find such separations and the properties of the graph if such separations do not exist.

► **Definition 2** (C.1 in [7]). Let  $G$  be a connected graph and  $V^\infty \subseteq V(G)$  be a set of undeletable vertices. A triple  $(Z, V_1, V_2)$  of subsets of  $V(G)$  is called a  $(q, k)$ -good node separation, if  $|Z| \leq k$ ,  $Z \cap V^\infty = \emptyset$ ,  $V_1$  and  $V_2$  are vertex sets of two different connected components of  $G - Z$  and  $|V_1 \setminus V^\infty|, |V_2 \setminus V^\infty| > q$ .

► **Definition 3** (C.2 in [7]). Let  $G$  be a connected graph,  $V^\infty \subseteq V(G)$  be a set of undeletable vertices, and  $T_b \subseteq V(G)$  a set of border terminals in  $G$ . A pair  $(Z, (V_i)_{i=1}^r)$  is called a  $(q, k)$ -flower separation in  $G$  (with regard to border terminals  $T_b$ ), if the following holds:

- $1 \leq |Z| \leq k$  and  $Z \cap V^\infty = \emptyset$ ; the set  $Z$  is the *core* of the flower separation  $(Z, (V_i)_{i=1}^r)$ ;
- $V_i$  are vertex sets of pairwise different connected components of  $G - Z$ , each set  $V_i$  is a *petal* of the flower separation  $(Z, (V_i)_{i=1}^r)$ ;
- $V(G) \setminus (Z \cup \bigcup_{i=1}^r V_i)$ , called a *stalk*, contains more than  $q$  vertices of  $V(G) \setminus V^\infty$ ;
- for each petal  $V_i$  we have  $V_i \cap T_b = \emptyset$ ,  $|V_i \setminus V^\infty| \leq q$  and  $N_G(V_i) = Z$ ;
- $|\bigcup_{i=1}^r V_i \setminus V^\infty| > q$ .

► **Lemma 4** (C.3 in [7]). Given a connected graph  $G$  with undeletable vertices  $V^\infty \subseteq V(G)$  and integers  $q$  and  $k$ , one may find in  $\mathcal{O}(2^{\mathcal{O}(\min(q,k) \log(q+k))} n^3 \log n)$  time a  $(q, k)$ -good node separation of  $G$ , or correctly conclude that no such separation exists.

► **Lemma 5** (C.4 in [7]). Given a connected graph  $G$  with undeletable vertices  $V^\infty \subseteq V(G)$  and border terminals  $T_b \subseteq V(G)$  and integers  $q$  and  $k$ , one may find a  $(q, k)$ -flower separation in  $G$  w.r.t.  $T_b$  in  $\mathcal{O}(2^{\mathcal{O}(\min(q,k) \log(q+k))} n^3 \log n)$  time, or correctly conclude that no such flower separation exists.

► **Lemma 6** (C.5 in [7]). If a connected graph  $G$  with undeletable vertices  $V^\infty \subseteq V(G)$  and border terminals  $T_b \subseteq V(G)$  does not contain a  $(q, k)$ -good node separation or a  $(q, k)$ -flower separation w.r.t.  $T_b$  then, for any  $Z \subseteq V(G) \setminus V^\infty$  of size at most  $k$ , the graph  $G - Z$  contains at most  $(2q + 2)(2^k - 1) + |T_b| + 1$  connected components containing a vertex of  $V(G) \setminus V^\infty$ , out of which at most one has more than  $q$  vertices not in  $V^\infty$ .

### 3 Overview of the algorithm

To solve STRONG BIPARTITE DELETION, we first reduce it to a slightly more general problem where we also have a set  $U$  of undeletable vertices and we are not allowed to select vertices in our potential solution from  $U$ . In particular the problem we will study is as follows.

PSEUDOBIPARTITE DELETION

Parameters:  $k, \ell$

**Input:** A graph  $G$ , integers  $k$  and  $\ell$  and a set  $U \subseteq V(G)$ .

**Question:** Does there exist  $X \subseteq V(G)$  such that  $|X| \leq k$ ,  $X \cap U = \emptyset$  and  $G - X$  is  $\ell$ -pseudobipartite?

Observe that when we set  $U = \emptyset$  in PSEUDOBIPARTITE DELETION, we get the STRONG BIPARTITE DELETION problem. In rest of the paper, we design an algorithm for PSEUDOBIPARTITE DELETION using the method of iterative compression and recursive understanding



## 21:6 Strong Parameterized Deletion: Bipartite Graphs

technique introduced by Grohe et al. [12]. We start off by deleting the connected components from the graph which are already  $\ell$ -pseudobipartite. Then using the technique of iterative compression, in Lemma 8, we reduce an instance of PSEUDOBIPARTITE DELETION to  $2^{\mathcal{O}(k)}$  many instances of the same problem such that the original instance is a YES instance if and only if at least one of the new instances is a YES instance. In addition to this, we also show that all the new instances have an oct of size bounded by a polynomial in  $k$  and  $\ell$ . This will help later in the *high connectivity* phase of the algorithm by giving some structure to the problem, as we will know that the graph has a reasonably small oct. Then we take care of the case when the graph has more than one connected component. This can be easily done by solving the problem optimally on each connected component. Then we define the border problem, where we are additionally provided with some border terminals, say  $T$ .

**BORDERED-PSEUDOBIPARTITE DELETION (B-PBD)** **Parameters:**  $k, \ell$   
**Input:** A PSEUDOBIPARTITE DELETION instance  $\mathcal{I} = (G, k, \ell, U)$  with  $G$  being connected and a set  $T \subseteq V(G)$  such that  $|T| \leq 2k$ ; denoted by  $\mathcal{I}_b = (G, k, \ell, U, T)$ .  
**Output:** Output one of the following three. (a) Find a *special* vertex  $v$  such that for each  $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$ , there is a minimum solution  $\text{sol}_{\mathcal{P}}^*$  which contains  $v$ , or (b) for each  $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$ , output  $\text{sol}_{\mathcal{P}} = X_{\mathcal{P}}$  which is a *minimum* solution to  $(\mathcal{I}_b, \mathcal{P})$ , or (c) output  $\text{sol}_{\mathcal{P}} = \perp$  if none of the earlier two cases apply.

Here, the set  $\mathbb{P}(\mathcal{I}_b)$  denotes the set of ‘interactions’ of the border terminals of instance  $\mathcal{I}_b$  with a solution and the objective is to find a solution that corresponds to each possible interaction or to find a *special* vertex which is part of a solution for each possible interaction. It can be easily seen that for a PSEUDOBIPARTITE DELETION instance  $\mathcal{I} = (G, k, \ell, U)$ , solving the B-PBD instance  $(G, k, \ell, U, \emptyset)$  either gives a solution to the instance  $\mathcal{I}$  or outputs a vertex which is part of a minimum solution for the instance  $\mathcal{I}$ , so in any case we make progress.

As is the case in algorithms based on the recursive understanding approach, to solve the border problem, we proceed to check whether a good separator  $T'$  exists in the graph. We use the notions of good node separations and good flower separations defined by Chitnis et al.[7] and look for good  $(q, k)$  node separation or  $(q, k)$  flower separation. The definitions are given in the preliminaries section. The running times required to compute such separations (if they exist) are argued by Lemmas 4 and 5. If we succeed in finding such a separation, then the graph gets divided into two large parts using a small separator. The definitions of node separations and flower separation help us argue that one of the parts (containing at most half of the border terminals) is connected. We call the smaller graph  $H$ .

Now we update the set of terminals to include the separator, and solve the border problem  $\mathcal{I}'_b$  recursively on the smaller graph. That is, for every behavior  $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$  of the new border terminals, we get an optimum solution. Let  $Z$  denote the set  $\bigcup_{\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)} \text{sol}_{\mathcal{P}}$  where  $\text{sol}_{\mathcal{P}}$  is the solution for the smaller graph for behavior  $\mathcal{P}$  of the border terminals. Then in Lemma 9, we argue that there exists a solution for the instance  $\mathcal{I}_b$ , which intersects with the smaller graph only in  $Z$ . At this time, we apply certain operations on the graph, such that the total number of the vertices in the graph  $G$  reduces by a sufficient amount. The approach of simply bypassing all the vertices not required by a solution does not quite work, as it could conceivably create spurious odd cycles which could lead to a YES-instance turning into a NO-instance. Hence, we make use of a ‘parity preserving’ bypassing operation to reduce the vertices of the graph, and show in Lemma 11 that this operation results in an equivalent instance. By proving a bound on the size of the set  $\mathbb{P}(\mathcal{I}_b)$  as a function of  $k$  and  $\ell$ , we guarantee that after the application of the recursive step (Step 2 in the algorithm) the number of vertices in the graph undergoes a sufficient reduction.



We then describe the algorithm for the problem in the case when there is no good-separation to recurse upon. Here, we need to solve the BORDER-PSEUDOBI PARTITE DELETION instance  $(G, k, \ell, U, T)$  in the case that Step 2 is not applicable on the graph. For this, for each  $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$ , we solve the instance  $(G_{\mathcal{P}}, k, \ell, U')$ , where the graph  $G_{\mathcal{P}}$  encoded the interaction,  $\mathcal{P}$ , of the border terminals  $T$ . We argue that in the absence of a good node separation or a flower separation of size at most  $k$  in the graph  $G$ , Lemma 6 guarantees an appropriate type of *high connectivity* in the graph  $G$ . In Lemma 12, we exploit this to get a similar high connectivity property for the graph  $G_{\mathcal{P}}$ . Since the graph  $G$  has an oct of bounded size because of Lemma 8, we can show a similar bound on oct of the graph  $G_{\mathcal{P}}$  also. At this stage, we use the algorithm for finding oct given in [19] to find an oct of bounded size and get an instance of the following problem.

OCT-PBD

**Parameters:**  $k, \ell$

**Input:** An instance  $(G, k, \ell, U)$  of PSEUDOBI PARTITE DELETION along with an oct  $O$  of  $G$  of size at most  $g(k, \ell)$  such that for any  $Z \subseteq (V(G) \setminus U)$  of size at most  $k$ , in the graph  $G - Z$ , at most one connected component containing a vertex of  $V(G) \setminus U$  has more than  $h(k, \ell)$  vertices not in  $U$ .

**Output:** A minimum sized  $\ell$ -pseudobipartite deletion set  $X$  of  $G$  of size at most  $k$  such that  $X \cap U = \emptyset$ . Output  $\perp$  if such a set does not exist.

Then we guess the intersection of the oct with the solution. This lets us assume that the solution is disjoint from the oct. Formally, we branch into  $2^{|O|}$  instances of the following problem which we call OCT-PBD(I). Here, the input is an instance  $(G, k, \ell, U, O)$  of OCT-PBD and the objective is to compute a minimum sized  $\ell$ -pseudobipartite deletion set  $X$  of  $G$  of size at most  $k$  such that  $X \cap (O \cup U) = \emptyset$  or return  $\perp$  if such a set does not exist. To solve an instance of OCT-PBD(I), we guess which vertices of the oct are going to be in the same connected component after deleting the solution. This gives us  $g(k, \ell)^{g(k, \ell)}$  instances of following variant of OCT-PBD(I), which we call OCT-PBD(II). Here, the input is an instance  $(G, k, \ell, U, O)$  of OCT-PBD(I) and an equivalence relation  $\S$  on  $O$  with the guarantee that there exists a minimum sized  $\ell$ -pseudobipartite deletion set  $X \subseteq V(G) \setminus (U \cup O)$  of  $G$  such that for all  $u, v \in O$ ,  $u$  and  $v$  belong to the same connected component of  $G - X$  if and only if  $(u, v) \in \S$ . The objective is to compute a minimum sized  $\ell$ -pseudobipartite deletion set  $X$  of  $G$  of size at most  $k$  such that  $X \cap (O \cup U) = \emptyset$ . We output  $\perp$  if such a set does not exist.

To solve an instance  $(G, k, \ell, U, O, \S)$  of OCT-PBD(II), we look at the number of equivalence classes in  $\S$ . If it is more than one, then either we solve the problem via brute force on some connected component (of size at most  $h(k, \ell)$ ) or we give a branching step where we solve at most  $h(k, \ell) + 1$  instances OCT-PBD(II), where the size of the solution decreases by at least one. In the other case, there is only one equivalence class in  $S$ , which means that all the vertices of oct are going to be part of a single connected component resulting by deleting a solution, we also guess how the oct vertices themselves will be bipartitioned eventually upon deleting the  $k$  vertices in the solution and an arbitrary but fixed set of at most  $\ell$  edges which make the connected component containing these vertices bipartite. So for each instance of OCT-PBD(II), we get  $2^{|O|}$  instances of the problem OCT-PBD(III) as defined below.

OCT-PBD(III)	<b>Parameters:</b> $k, \ell$
<b>Input:</b> An instance $(G, k, \ell, U, O, \xi)$ of OCT-PBD(II) such that for all $u, v \in O$ , $(u, v) \in \xi$ and a bipartition $(O_1 \uplus O_2)$ of $O$ with the guarantee that there exists a minimum sized $\ell$ -pseudobipartite deletion set $X' \subseteq V(G) \setminus (U \cup O)$ of $G$ such that all vertices of $O$ belong to the same connected component of $G - X'$ having vertex set $C$ and there exists an edge-oct $F$ of $G[C]$ of size at most $\ell$ and a bipartition $(C_1 \uplus C_2)$ of $C$ such that $G[C_1] - F$ and $G[C_2] - F$ are independent sets and $O_1 \subseteq C_1$ and $O_2 \subseteq C_2$ .	
<b>Output:</b> A minimum sized $\ell$ -pseudobipartite deletion set $X$ of $G$ of size at most $k$ such that $X \cap (O \cup U) = \emptyset$ . Output $\perp$ if such a set does not exist.	

In Lemma 13, we argue that this branching is correct. Finally, to solve an instance  $\mathcal{I}$  of OCT-PBD(III), we reduce it to an instance of MMCU\* – a special instance of MMCU with some undeletable vertices.

MMCU*	<b>Parameters:</b> $k, \ell$
<b>Input:</b> A graph $G$ , integers $k$ and $\ell$ , $T \subseteq V(G)$ , an equivalence relation $\mathcal{R}$ on $T$ having at most two equivalence classes, and set of undeletable vertices $U \subseteq V(G)$ .	
<b>Output:</b> Output a minimal solution $\mathcal{X} = (X, F)$ to MMCU instance $(G, T, \mathcal{R}, k, \ell)$ such that $X \cap U = \emptyset$ and $\perp$ if no such solution exists.	

Finally, to solve MMCU\* we show that it can be reduced to  $(|U| + 2)^{|U|}$  instances of MMCU, which is solved using the algorithm in [25].

## 4 An algorithm for Pseudobipartite Deletion

In this section, we describe the FPT algorithm for PSEUDOBI-PARTITE DELETION. In the first step of the algorithm, we get rid of connected components of the graph which are already  $\ell$ -pseudobipartite.

► **Step 1.** Let  $(G, k, \ell, U)$  be an instance of PSEUDOBI-PARTITE DELETION. Let  $\mathcal{C} := \{C \mid G[C] \text{ is a connected component of } G\}$ . For each  $C \in \mathcal{C}$ , find if the graph  $G[C]$  is  $\ell$ -pseudobipartite. Let  $\mathcal{C}' = \{C \in \mathcal{C} \mid G[C] \text{ is } \ell\text{-pseudobipartite}\}$  and let  $C' = \bigcup_{C \in \mathcal{C}'} C$ . Pass the PSEUDOBI-PARTITE DELETION instance  $(G - C', k, \ell, U \setminus C')$  to the next step.

Now we use the method of iterative compression to reduce an instance of PSEUDOBI-PARTITE DELETION to at most  $n - k$  instances of PBD COMPRESSION (PBD-C) problem, where we are given an instance  $(G, k, \ell, U)$  of PSEUDOBI-PARTITE DELETION along with an  $\ell$ -pseudobipartite deletion set of  $G$  of size at most  $k + 1$  and we are asked whether one exists of size at most  $k$ . We prove the following lemma which helps us in bounding the size of an oct.

► **Lemma 7.** *Let  $(G, k, \ell, U, S)$  be an instance of PBD-C such that none of the connected components of  $G$  are  $\ell$ -pseudobipartite. If  $G$  has an  $\ell$ -pseudobipartite deletion set  $Z$  disjoint from  $S$  of size at most  $k$ , then it has an oct of size at most  $g'(k, \ell) := 2k + k\ell^2 + 1$ .*

The proof of the lemma follows from the argument that  $G - Z$  can have at most  $k + k\ell + 1$  connected components which are not bipartite. Now we prove the lemma which reduces a compression instance to many instances of PSEUDOBI-PARTITE DELETION with a guarantee on the size of an oct for all of them.

► **Lemma 8.** *There is an algorithm, which given an instance  $(G, k, \ell, U, S)$  of PBD-C, runs in time  $2^{\mathcal{O}(k\ell^2)} n^{\mathcal{O}(1)}$  and returns at most  $2^{k+1}$  instances  $\{I_1, I_2, \dots, I_q\}$  of PSEUDOBI-PARTITE DELETION, where  $I_i = (G_i, k_i, \ell, U)$  and  $q \leq 2^{k+1}$  such that the following holds.*

- $(G, k, \ell, U, S)$  is a YES instance of PBD-C if and only if there is an  $1 \leq i \leq q$  such that  $(G_i, k_i, \ell, U)$  is a YES instance of PSEUDOBIPARTITE DELETION.
- For each  $1 \leq i \leq q$ ,  $k_i \leq k$  and  $G_i$  has an oct of size at most  $2k + k\ell^2 + 1$ .

Henceforth, we will assume that the given PSEUDOBIPARTITE DELETION instance is returned by the algorithm of Lemma 8 and hence has an oct of bounded size.

**Borders and Recursive Understanding.** After applying Lemma 8, we still need to solve  $2^{k+1}$  PSEUDOBIPARTITE DELETION instances. We give an algorithm to solve instances which are given by Lemma 8. We first do some preprocessing to assume that the instances of PSEUDOBIPARTITE DELETION which we are going to solve are connected. Now we are ready to define the border problem and describe the recursive phase of the algorithm.

Let  $\mathcal{I} = (G, k, \ell, U)$  be a PSEUDOBIPARTITE DELETION instance. The input to the border problem consists of an instance  $\mathcal{I} = (G, k, \ell, U)$  of PSEUDOBIPARTITE DELETION along with a set  $T$  of at most  $2k$  vertices of  $G$  disjoint from  $U$ . The output to the border problem either consists of *several* solutions, one for each relevant ‘behaviour’ defined on the set of terminals, or it consists of a single *special* vertex, which is part of some minimum solution for every behaviour. In what follows, we will formalize this statement.

Let  $\mathcal{I} = (G, k, \ell, U)$  be an instance of PSEUDOBIPARTITE DELETION and  $T \subseteq V(G)$ . We let  $\mathbb{P}(\mathcal{I}_b)$  denote the set of all tuples  $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L)$ , such that  $X_T \subseteq T$ ,  $\mathcal{R}$  is an equivalence relation on  $T \setminus X_T$ ,  $\mathcal{B}$  is a bipartition of  $T \setminus X_T$  and  $L = \{(R_1, \ell_1), (R_2, \ell_2), \dots, (R_q, \ell_q)\}$  is a set of pairs which associates an integer  $\ell_i \leq \ell$  with each equivalence class  $R_i$  in  $\mathcal{R}$ . For a tuple  $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L)$  with the equivalence classes of  $\mathcal{R}$  being  $R_1, \dots, R_q$ , the bipartition induced on the class  $R_i$  by  $\mathcal{B}$  is denoted by  $(R_{i_1}, R_{i_2})$ .

For each  $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$ , we define a super-graph  $G_{\mathcal{P}}$  of  $G$  with the following additional vertices and edges.

- For each equivalence class  $R_i$  of  $\mathcal{R}$ , we add sets of vertices  $R'_{i_1}$  and  $R'_{i_2}$  such that  $|R'_{i_1}| = |R'_{i_2}| = \ell + 1$ . Then we add all the edges between vertices  $u$  and  $v$  such that  $u \in R_{i_1} \cup R'_{i_1}$  and  $v \in R_{i_2} \cup R'_{i_2}$ . If  $(R_i, \ell_i) \in L$ , then we pick an arbitrary vertex  $u^i \in R_i$  and add  $\ell_i$ -many edge disjoint triangles which only intersect in  $u^i$ . That is, we add  $2\ell_i$  new vertices  $u_1^{i,a}, \dots, u_{\ell_i}^{i,a}, u_1^{i,b}, \dots, u_{\ell_i}^{i,b}$  and edges  $\{(u^i, u_j^{i,a}), (u^i, u_j^{i,b}), (u_j^{i,a}, u_j^{i,b})\}$  for each  $1 \leq j \leq \ell_i$ .
- For each vertex  $u \in X_T$ , we add  $\ell + 1$ -many edge disjoint triangles which only intersect in  $u$ . That is, we add  $2(\ell + 1)$  new vertices  $u_1^a, \dots, u_r^a, u_1^b, \dots, u_r^b$  where  $r = 2\ell + 2$  and edges  $\{(u, u_j^a), (u, u_j^b), (u_j^a, u_j^b)\}$  for each  $1 \leq j \leq r$ .

This completes the description of the graph  $G_{\mathcal{P}}$ . It can be seen that  $|V(G_{\mathcal{P}}) \setminus V(G)| \leq 2k(4\ell + 1)$  and  $|E(G_{\mathcal{P}}) \setminus E(G)| \leq 4k(2k + 3)(\ell + 1)$ . The intuition behind the newly added vertices and edges is the following. Consider an instance of PSEUDOBIPARTITE DELETION and suppose that  $G$  is a subgraph of the input instance with the terminal set  $T$  separating this subgraph from the rest of the input graph. The tuple  $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L)$  essentially captures the interaction of the terminal set with the solution and the rest of the graph. That is,  $X_T$  denotes the intersection of the solution with  $T$ . The partition  $\mathcal{R}$  denotes the way the remaining vertices of  $T$  are partitioned as connected components after deleting a solution. The bipartition  $\mathcal{B}$  denotes the bipartition of  $T$  induced by an arbitrary bipartition of the graph obtained from the input graph by deleting the  $k$  vertices in the solution and then a minimum edge-oct in the rest of the graph. Finally, for each  $R \in \mathcal{R}$ , if  $(R, x) \in L$ , it means that after deleting the  $k$  vertices in a solution, there is a set of at most  $x$  edges in the connected component containing  $R$  such that they are part of a minimum edge-oct of

## 21:10 Strong Parameterized Deletion: Bipartite Graphs

this component *and* they lie outside  $G$ . The newly added vertices essentially simulate this interaction of the terminals with the vertices in the solution.

We denote by  $U_{\mathcal{P}}$  the union of the set  $U$  with the vertices in  $V(G_{\mathcal{P}}) \setminus V(G)$ . Also, for  $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L)$ , for each  $R_i \in \mathcal{R}$ , we denote by  $\mathfrak{H}_{\mathcal{P}}^G(R_i)$  the set of vertices in  $V(G_{\mathcal{P}}) \setminus V(G)$  which are adjacent to a vertex of  $R_i$ . We drop the reference to  $G$  if it is clear from the context.

We say that a set  $X \subseteq V(G) \setminus U$  is a *solution* to  $(\mathcal{I}_b, \mathcal{P})$  where  $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L) \in \mathbb{P}(\mathcal{I}_b)$  if  $X$  is a solution to the PSEUDOBI PARTITE DELETION instance  $(G_{\mathcal{P}}, k, \ell, U'_{\mathcal{P}})$  where  $U'_{\mathcal{P}} = U_{\mathcal{P}} \cup (T \setminus X_T)$ . Recall that when we say that a set  $S$  is a solution to a PSEUDOBI PARTITE DELETION instance  $(G, k, \ell, U)$ , we mean that  $|S| \leq k$ ,  $S \cap U = \emptyset$  and  $G - S$  is  $\ell$ -pseudobipartite.

It can be easily seen that for a PSEUDOBI PARTITE DELETION instance  $\mathcal{I} = (G, k, \ell, U)$ , solving the B-PBD instance  $(G, k, \ell, U, \emptyset)$  either gives a solution to the instance  $\mathcal{I}$  or outputs a vertex which is part of a minimum solution for the instance  $\mathcal{I}$ , so in any case we make progress. In what follows we will show that *given* a correct output for an instance of B-PBD, there is an FPT algorithm which either computes an equivalent instance whose size is bounded by a function of  $k$  and  $\ell$  or it outputs a special vertex as described in the problem definition.

► **Lemma 9.** *Let  $\mathcal{I}_b = (G, k, \ell, U, T)$  be an instance of B-PBD. Let  $T' \subseteq V(G) \setminus U$  and let  $C$  be the vertex set of a connected component of  $G - T'$  such that  $N(C) = T'$ . Let  $H = G[C \cup T']$  and let  $Q = T' \cup (C \cap T)$  and suppose that  $|Q| \leq 2k$ . Let  $\mathcal{I}'_b$  denote the B-PBD instance  $(H, k, \ell, (U \cap V(H)), Q)$ . Let  $Z$  denote the set  $\bigcup_{\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)} \text{sol}_{\mathcal{P}}$ . Then, for each  $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$ , if there is a solution for  $(\mathcal{I}_b, \mathcal{P})$ , then there is one whose intersection with  $C$  is in  $Z$ . Moreover, if there exists a vertex  $v$  such that  $v \in \text{sol}_{\mathcal{P}'}$  for all  $\mathcal{P}' \in \mathbb{P}(\mathcal{I}'_b)$ , then for all  $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$ , there exists a minimum solution to  $(\mathcal{I}_b, \mathcal{P})$  which contains  $v$ .*

Before we state our next lemma, we need to recall the notion of *parity-torso* (see for example [20]).

► **Definition 10.** Let  $G$  be a graph and  $S \subseteq V(G)$ . We denote by  $PT(G, S)$  the graph obtained from  $G - S$  as follows. For every pair of vertices  $u, v$  in  $V(G) \setminus S$ , if there is an odd length  $u$ - $v$  path in  $G$  whose internal vertices all lie in  $S$  then we add an edge  $(u, v)$  and if there is an even length  $u$ - $v$  path in  $G$  whose internal vertices all lie in  $S$  then we add a *subdivided* edge (path of length 2) between  $u$  and  $v$ .

We are now ready to state our next crucial lemma which essentially gives a way to *get rid of* vertices which we know will never be required in our solution.

► **Lemma 11.** *Let  $\mathcal{I}_b = (G, k, \ell, U, T)$  be an instance of B-PBD. Let  $T' \subseteq V(G) \setminus U$  and let  $C$  be the vertex set of a connected component of  $G - T'$  such that  $N(C) = T'$ . Let  $H = G[C \cup T']$  and let  $Q = T' \cup (C \cap T)$  and suppose that  $|Q| \leq 2k$ . Let  $\mathcal{I}'_b$  denote the B-PBD instance  $(H, k, \ell, (U \cap V(H)), Q)$ . Suppose that for every  $v \in V(H)$ , there is a  $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$  such that  $v \notin \text{sol}_{\mathcal{P}}$ . Let  $Z$  denote the set  $\bigcup_{\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)} \text{sol}_{\mathcal{P}}$ . Then,*

- $|Z| = 2^{\mathcal{O}(k \log(k+\ell))}$  and
- there are functions  $\tau, \alpha$  and an algorithm that, given  $\mathcal{I}_b$  and  $Z$ , runs in time  $(\tau(k, \ell)n^{\mathcal{O}(1)})$  and computes a set  $W \subseteq V(H)$  such that  $W \supseteq Q$  and has size at most  $\alpha(k, \ell)$  such that the instance  $\mathcal{I}_b$  is equivalent to the instance  $\mathcal{I}_b^1 = (G', k, \ell, U, T)$  where the graph  $G'$  is defined as  $PT(G, V(H) \setminus W)$ . Here,  $\alpha(k, \ell)$  and  $\tau(k, \ell)$  are both  $2^{\mathcal{O}(k \log(k+\ell))}$ .

Now we describe the recursive step of the algorithm.

► **Step 2.** Assume we are given a B-PBD instance  $\mathcal{I}_b = (G, k, \ell, U, T)$  and let  $q := \alpha(k, \ell) + \binom{\alpha(k, \ell)}{2} + 1$ . Invoke first the algorithm of Lemma 4 in a search for  $(q, k)$ -good node separation

(with  $V^\infty = U$ ). If it returns a good node separation  $(Z, V_1, V_2)$ , let  $j \in \{1, 2\}$  be such that  $|V_j \cap T| \leq k$  and denote  $T' = N(V_j) \subseteq Z$ ,  $C = V_j$ . Otherwise, if it returns that no such good node separation exists in  $G$ , invoke the algorithm of Lemma 5 in a search for  $(q, k)$ -flower separation w.r.t.  $T_b$  (with  $V^\infty = U$  again). If it returns that no such flower separation exists in  $G$ , pass the instance  $\mathcal{I}_b$  to the next step (high connectivity phase). Otherwise, if it returns a flower separation  $(Z, (V_i)_{i=1}^r)$ , denote  $C = \bigcup_{i=1}^r V_i$  and  $T' = N(C) \subseteq Z$ . Let  $H = G[C \cup T']$ . In the case we have obtained  $T'$  and  $C$  (either from Lemma 4 or Lemma 5), invoke the algorithm recursively for the B-PBD instance  $\mathcal{I}'_b$  defined as in the statement of Lemma 9 for sets  $T'$  and set  $C$ , obtaining an output  $\text{sol}_{\mathcal{P}}$  for each  $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$ . If there exists  $v \in V(H)$  such that  $v \in \text{sol}_{\mathcal{P}}$  for every  $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$ , we return  $v$  as the special vertex. Otherwise, compute the set  $Z = \bigcup_{\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)} \text{sol}_{\mathcal{P}}$ . Use the algorithm of Lemma 11 on  $\mathcal{I}'_b$  and  $Z$  to compute the set  $W$ . Generate the graph  $G' = PT(G, V(H) \setminus W)$ . Let  $\mathcal{I}_b^1 = (G', k, \ell, (U \cap V(H)), \mathcal{Q})$ , where  $\mathcal{Q} = T' \cup (C \cap T)$ .

Restart this step on instance  $\mathcal{I}_b^1$ . If it returns a special vertex  $v$ , then return  $v$  as a special vertex for the instance  $\mathcal{I}_b$ . Otherwise, obtain a family of solutions  $(\text{sol}'_{\mathcal{P}})_{\mathcal{P} \in \mathbb{P}}$  and return this family as output to the instance  $\mathcal{I}_b$ .

The correctness of Step 2 follows from Lemmas 9 and 11. Now we do a running time analysis for Step 2. Since  $q = \mathcal{O}(2^{\mathcal{O}(k \log(k+\ell))})$ , finding a good  $(q, k)$ -node separation or flower separation takes time  $\mathcal{O}(2^{\mathcal{O}(\min(q,k) \log(q+k))} n^3 \log n) = \mathcal{O}(2^{\mathcal{O}(k^2 \log(k+\ell))} n^3 \log n)$ . Let  $|V(H)| = n'$ , and hence by definitions of good node separation and flower separation we have that  $q + 1 \leq n' \leq n - q - 1$ . The first recursive call is applied to an instance on  $n'$  vertices. While taking the torso operation, we have that  $|W| = \alpha(k, \ell) = 2^{\mathcal{O}(k \log(k+\ell))}$  and finding the set  $W$  takes  $\tau(k, \ell) n^{\mathcal{O}(1)} = 2^{\mathcal{O}(k \log(k+\ell))} n^{\mathcal{O}(1)}$  time.

Let  $H' = G - V(H)$ . We know that  $T'$  separates  $H'$  from rest of the graph and  $T' \subseteq W$ . So, for any  $u, v \in V(G)$  such that  $u \in V(H')$ , we do not have any path from  $u$  to  $v$  having its internal vertices entirely in  $V(H) \setminus W$ . So if a new vertex  $z$  is added because of an even length path from  $u$  to  $v$ , we have that  $u, v \in V(H)$ . As  $G - (V(H) \setminus W)$  has at most  $n - n' + \alpha(k, \ell)$  vertices and none of the vertices in  $H'$  contribute to the torso operation, we have that  $|V(G')| \leq n - n' + \alpha(k, \ell) + \binom{\alpha(k, \ell)}{2} < |V(G)|$ . The base case for the recursive calls is the high connectivity phase, which we will argue takes time  $2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} n^{\mathcal{O}(1)}$ .

Solving the resulting recurrence gives  $T(n) = 2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} n^{\mathcal{O}(1)}$  in the worst case, which is the running time for Step 2. We remark that we never actually introduce any new undeletable vertices in the graph in this step.

**High Connectivity phase.** Assume we have a B-PBD instance  $\mathcal{I}_b = (G, k, \ell, U, T)$  where Step 2 is not applicable. Let us fix  $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L) \in \mathbb{P}(\mathcal{I}_b)$  and let  $U' := U \cup ((T \setminus X_T) \cup R_{\text{new}})$ , where  $R_{\text{new}} = V(G_{\mathcal{P}}) \setminus V(G)$ . We iterate through all possible values of  $\mathcal{P}$  and try to find a minimum solution to  $(G_{\mathcal{P}}, k, \ell, U')$ . Since  $|\mathbb{P}(\mathcal{I}_b)| = 2^{\mathcal{O}(k \log(k+\ell))}$ , this results in a factor of  $2^{\mathcal{O}(k \log(k+\ell))}$  in the running time. Thus, from now onwards we focus on one such  $\mathcal{P}$ . Furthermore, here we only solve the instances where  $|U'| \leq \mathcal{O}(k\ell)$ , which is sufficient for our purpose as we will argue later. We first prove the following lemma.

► **Lemma 12.** *Let  $\mathcal{I}_b = (G, k, \ell, U, T)$  be an instance of B-PBD where Step 2 is not applicable. Let  $U' := U \cup ((T \setminus X_T) \cup R_{\text{new}})$ , where  $R_{\text{new}} = V(G_{\mathcal{P}}) \setminus V(G)$ . Then for every  $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$ , the graph  $G_{\mathcal{P}}$  satisfies the following.*

- for any  $Z \subseteq (V(G_{\mathcal{P}}) \setminus U')$  of size at most  $k$ , the graph  $G_{\mathcal{P}} - Z$  contains at most  $f(k, \ell) = (2q+2)(2^k-1)+2k+1$  connected components containing a vertex of  $V(G) \setminus U'$ , out of which at most one has more than  $h(k, \ell)$  vertices not in  $U'$  where  $h(k, \ell) := q(4k(2k+3)(\ell+1)+1)$ , and
- $G_{\mathcal{P}}$  has an oct of size at most  $g(k, \ell) := 2k + k\ell^2 + 1 + |U'|$ .

## 21:12 Strong Parameterized Deletion: Bipartite Graphs

So, now we can generate an instance of PSEUDOBIPARTITE DELETION where an oct is also given and we have a bound on the number of vertices from  $V(G) \setminus U$  in all connected components after deleting a solution except one. We recall the formal problem definition from section 3.

<p>OCT-PBD</p> <p><b>Input:</b> An instance <math>(G, k, \ell, U)</math> of PSEUDOBIPARTITE DELETION along with an oct <math>O</math> of <math>G</math> of size at most <math>g(k, \ell)</math> such that for any <math>Z \subseteq (V(G) \setminus U)</math> of size at most <math>k</math>, in the graph <math>G - Z</math>, at most one connected component containing a vertex of <math>V(G) \setminus U</math> has more than <math>h(k, \ell)</math> vertices not in <math>U</math>.</p> <p><b>Output:</b> A minimum sized <math>\ell</math>-pseudobipartite deletion set <math>X</math> of <math>G</math> of size at most <math>k</math> such that <math>X \cap U = \emptyset</math>. Output <math>\perp</math> if such a set does not exist.</p>	<p><b>Parameters:</b> <math>k, \ell</math></p>
---	--

► **Step 3.** Find an oct  $O$  of  $G_{\mathcal{P}}$  of size  $g(k, \ell)$  using algorithm in [19]. Return an instance  $(G, k, \ell, U, O)$  of OCT-PBD.

The correctness of Step 3 is immediate from Lemma 12. After doing some simple guessing as explained in the overview section we arrive at an instance of OCT-PBD(II). Now to solve OCT-PBD(II). We return NO if  $k < 0$ . We first look at the case when the equivalence relation  $\S$  has more than one equivalence class. In this case, we know that there exists a solution  $X$  after deleting which, at least one of the equivalence classes of  $\S$  is in a connected component containing at most  $h(k, \ell)$  vertices not from  $U$ .

We proceed by guessing this equivalence class  $S_i$  in  $\S$ . Then we arbitrarily pick a vertex  $v$  in  $S_i$  and look at a connected subgraph  $H$  of  $G$  containing  $v$  which has  $h(k, \ell) + 1$  vertices not from  $U$ . We know that at least one of the vertices in  $V(H)$  has to be part of the solution  $X$ , because after deleting  $X$ , the connected component containing  $v$  has at most  $h(k, \ell)$  vertices not from  $U$ . Then we pick a vertex of  $V(H)$  and branch on it. Since each branching call decreases solution size we are looking for by at least one, the depth of the recursion tree is bounded by  $k$ .

If such a subgraph  $H$  does not exist, we have that the connected component containing  $v$  has at most  $h(k, \ell)$  vertices not in  $U$ , and then we solve the problem on that connected component using brute force, which takes time  $h(k, \ell)^k n^{\mathcal{O}(1)}$ .

Now we deal with the case when  $\S$  has only one equivalence class. That is, we know that there exists a solution  $X \subseteq V(G) \setminus (U \cup O)$  such that  $G - X$  is  $\ell$ -pseudobipartite and for all  $(u, v) \in O$ ,  $u$  and  $v$  belong to the same connected component of  $G - X$ . In other words, there exists a solution  $X$  of minimum size such that all the vertices of  $O$  lie in the same connected component of  $G - X$ . To solve this problem, we first prove the following.

► **Lemma 13.** *Let  $(G, k, \ell, U, O, \S)$  be an OCT-PBD(II) instance such that for all  $u, v \in O$ ,  $(u, v) \in \S$ . Then there exists a bipartition  $(O_1 \uplus O_2)$  and  $X' \subseteq V(G) \setminus (U \cup O)$  such that  $X'$  is a minimum sized  $\ell$ -pseudobipartite deletion set of  $G$ , all vertices of  $O$  belong to the same connected component of  $G - X'$  having vertex set  $C$  and there exists an edge-oct  $F$  of  $G[C]$  of size at most  $\ell$  and a bipartition  $(C_1 \uplus C_2)$  of  $C$  such that  $G[C_1] - F$  and  $G[C_2] - F = \emptyset$  are independent sets and  $O_1 \subseteq C_1$  and  $O_2 \subseteq C_2$ .*

This lemma helps us reduce an instance of OCT-PBD(II) into  $2^{|O|}$  instances of OCT-PBD(III). We can solve an instance of OCT-PBD(III) by appropriately casting it as an instance of MMCU\* problem.

A careful analysis shows that the total running time for high connectivity phase is  $2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} n^{\mathcal{O}(1)}$ . This finishes the description of the algorithm. Theorem 1 follows from this algorithm and the fact that the set of undeletable vertices is initially empty, and the



only time we actually add undeletable vertices to a graph is while solving the high connectivity phase, in which case, we add at most  $2k(4\ell + 1) = \mathcal{O}(k\ell)$  vertices to the undeletable set.

## 5 Conclusions

We have introduced and studied a stronger version of the classical  $\mathcal{F}$ -DELETION problem where  $\mathcal{F}$  is the class of bipartite graphs and the new objective is to find a set of  $k$  vertices and  $\ell$  edges such that upon deletion of these  $k$  vertices, each component of the resulting graph is in the class  $\mathcal{F} + \ell e$ . We believe that a systematic study of this problem for various well-understood classes  $\mathcal{F}$ , for instance Interval Graphs, Chordal Graphs and so on, will prove to be a fruitful research direction driving the development of new tools and techniques for graph modification problems. We also think that the idea of combining iterative compression with recursive understanding is quite general in its approach and can be useful in getting similar results for other strong editing problems.

---

### References

- 1 Akanksha Agrawal, Daniel Lokshantov, Amer E. Mouawad, and Saket Saurabh. Simultaneous Feedback Vertex Set: A Parameterized Perspective. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2016.7.
- 2 Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.
- 3 Yixin Cao. Unit interval editing is fixed-parameter tractable. In *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6–10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 306–317. Springer, 2015.
- 4 Yixin Cao. Linear recognition of almost interval graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10–12, 2016*, pages 1096–1115, 2016.
- 5 Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Transactions on Algorithms*, 11(3):21:1–21:35, 2015.
- 6 Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016.
- 7 Rajesh Hemant Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20–23, 2012*, pages 460–469, 2012.
- 8 Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *Eur. J. Comb.*, 34(3):541–566, 2013.
- 9 Fedor V. Fomin, Daniel Lokshantov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation, kernelization and optimal FPT algorithms. In *FOCS*, 2012.
- 10 Toshihiro Fujito. A unified approximation algorithm for node-deletion problems. *Discrete Appl. Math.*, 86:213–231, September 1998. doi:10.1016/S0166-218X(98)00035-3.
- 11 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. In *Proceedings of the Twenty-Seventh Annual*

- ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1670–1681, 2016.
- 12 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 479–488, 2011.
  - 13 Bart M. P. Jansen. The power of data reduction: Kernels for fundamental graph problems. *Ph.D. Thesis*, 2013.
  - 14 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1802–1811, 2014.
  - 15 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum  $k$ -way cut of bounded size is fixed-parameter tractable. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 160–169, 2011.
  - 16 Eun Jung Kim and O.-joung Kwon. A polynomial kernel for block graph deletion. In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPICs*, pages 270–281, 2015.
  - 17 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. In *Automata, Languages, and Programming – 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 613–624, 2013.
  - 18 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.
  - 19 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms*, 11(2):15:1–15:31, 2014.
  - 20 Daniel Lokshtanov and M. S. Ramanujan. Parameterized tractability of multiway cut with parity constraints. In *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 750–761, 2012. doi:10.1007/978-3-642-31594-7\_63.
  - 21 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41:960–981, September 1994. doi:10.1145/185675.306789.
  - 22 Dániel Marx, Barry O’Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms*, 9(4):30, 2013.
  - 23 Geevarghese Philip, Ashutosh Rai, and Saket Saurabh. Generalized pseudoforest deletion: Algorithms and uniform kernel. In *Mathematical Foundations of Computer Science 2015 – 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 517–528. Springer, 2015.
  - 24 Marcin Pilipczuk, Michal Pilipczuk, and Marcin Wrochna. Edge bipartization faster than  $2^k$ . In *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark, LIPICs*, 2016.
  - 25 Ashutosh Rai, M. S. Ramanujan, and Saket Saurabh. A parameterized algorithm for mixed-cut. In *LATIN 2016: Theoretical Informatics – 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, pages 672–685, 2016.
  - 26 Mihalis Yannakakis. Node-and edge-deletion NP-complete problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing, STOC’78*, pages 253–264, New York, NY, USA, 1978. ACM. doi:10.1145/800133.804355.



# Parameterized Algorithms for List $K$ -Cycle\*

Fahad Panolan<sup>1</sup> and Meirav Zehavi<sup>†2</sup>

1 Department of Informatics, University of Bergen, Norway  
fahad.panolan@ii.uib.no

2 Department of Informatics, University of Bergen, Norway  
meirav.zehavi@ii.uib.no

---

## Abstract

The classic  $K$ -CYCLE problem asks if a graph  $G$ , with vertex set  $V(G)$ , has a simple cycle containing all vertices of a given set  $K \subseteq V(G)$ . In terms of colored graphs, it can be rephrased as follows: Given a graph  $G$ , a set  $K \subseteq V(G)$  and an injective coloring  $c : K \rightarrow \{1, 2, \dots, |K|\}$ , decide if  $G$  has a simple cycle containing each color in  $\{1, 2, \dots, |K|\}$  (once). Another problem widely known since the introduction of color coding is COLORFUL CYCLE. Given a graph  $G$  and a coloring  $c : V(G) \rightarrow \{1, 2, \dots, k\}$  for some  $k \in \mathbb{N}$ , it asks if  $G$  has a simple cycle of length  $k$  containing each color in  $\{1, 2, \dots, k\}$  (once). We study a generalization of these problems: Given a graph  $G$ , a set  $K \subseteq V(G)$ , a list-coloring  $L : K \rightarrow 2^{\{1, 2, \dots, k^*\}}$  for some  $k^* \in \mathbb{N}$  and a parameter  $k \in \mathbb{N}$ , LIST  $K$ -CYCLE asks if one can assign a color to each vertex in  $K$  so that  $G$  would have a simple cycle (of arbitrary length) containing exactly  $k$  vertices from  $K$  with distinct colors.

We design a randomized algorithm for LIST  $K$ -CYCLE running in time  $2^k n^{\mathcal{O}(1)}$  on an  $n$ -vertex graph, matching the best known running times of algorithms for both  $K$ -CYCLE and COLORFUL CYCLE. Moreover, unless the Set Cover Conjecture is false, our algorithm is essentially optimal. We also study a variant of LIST  $K$ -CYCLE that generalizes the classic HAMILTONICITY problem, where one specifies the size of a solution. Our results integrate three related algebraic approaches, introduced by Björklund, Husfeldt and Taslamán (SODA'12), Björklund, Kaski and Kowalik (STACS'13), and Björklund (FOCS'10).

**1998 ACM Subject Classification** G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

**Keywords and phrases** Parameterized Complexity,  $K$ -Cycle, Colorful Path,  $k$ -Path

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.22

## 1 Introduction

For a graph  $G$ , let  $V(G)$  and  $E(G)$  denote its vertex set and edge set respectively. The input for the classic  $K$ -CYCLE problem consists of an undirected graph  $G$  and a subset  $K \subseteq V(G)$  of size  $k$  for some  $k \in \mathbb{N}$ , and the objective is to decide whether  $G$  has a  $K$ -cycle, that is, a simple cycle that contains all of the vertices in  $K$ . In terms of (partially) colored graphs, it can be rephrased as follows. Given an undirected graph  $G$ , a set of vertices  $K \subseteq V(G)$  of size  $k$  for some  $k \in \mathbb{N}$  and an injective coloring  $c : K \rightarrow \{1, 2, \dots, k\}$ , it asks whether  $G$  has a simple cycle that contains each color in  $\{1, 2, \dots, k\}$  (once). Another problem widely known since the introduction of the color coding method [1] is COLORFUL CYCLE. Given an

---

\* The research leading to these results received funding from the European Research Council (ERC) via grants Rigorous Theory of Preprocessing, reference 267959 and PARAPPROX, reference 306992.

† Part of this work was done while M. Zehavi was visiting the Simons Institute for the Theory of Computing.



© Fahad Panolan and Meirav Zehavi;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 22; pp. 22:1–22:15



Leibniz International Proceedings in Informatics

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

undirected graph  $G$  and a coloring  $c : V \rightarrow \{1, 2, \dots, k\}$  for some  $k \in \mathbb{N}$ , decide whether  $G$  has a simple cycle of length  $k$  that contains each color in  $\{1, 2, \dots, k\}$  (once).

We study the parameterized complexity of a generalization of both  $K$ -CYCLE and COLORFUL CYCLE, called LIST  $K$ -CYCLE. In LIST  $K$ -CYCLE, the input consists of an undirected graph  $G$ , a set of vertices  $K \subseteq V(G)$ , a list-coloring  $L : K \rightarrow 2^{\{1, 2, \dots, k^*\}}$  for some  $k^* \in \mathbb{N}$  and a parameter  $k \in \mathbb{N}$ . We need to decide whether one can find a function  $c : K \rightarrow \{1, \dots, k^*\}$  so that  $c(v) \in L(v)$  for all  $v \in V(G)$  and  $G$  would have a simple cycle containing exactly  $k$  vertices from  $K$  and these vertices have distinct colors. In case  $k = k^*$ , it is simply requested that the cycle contains each color in  $\{1, 2, \dots, k\}$  (once). If  $|K| = k = k^*$  and for all  $v \in K$ ,  $|L(v)| = 1$ , we obtain  $K$ -CYCLE, while if  $K = V$ ,  $k = k^*$  and for all  $v \in K$ ,  $|L(v)| = 1$ , we obtain COLORFUL CYCLE. We also consider a variant of LIST  $K$ -CYCLE, called EXACT LIST  $K$ -CYCLE. Given an undirected graph  $G$ , a set of vertices  $K \subseteq V(G)$ , a list-coloring  $L : K \rightarrow 2^{\{1, 2, \dots, k^*\}}$  for some  $k^* \in \mathbb{N}$  and parameters  $k, \ell \in \mathbb{N}$ , it asks whether one can assign a color to each vertex in  $K$  so that  $G$  would have a simple cycle of length  $\ell$  containing exactly  $k$  vertices from  $K$  and these vertices have distinct colors. EXACT LIST  $K$ -CYCLE generalizes the classic  $\ell$ -PATH and HAMILTONICITY problems.

We study the problem LIST  $K$ -CYCLE in the realm of parameterized complexity. In parameterized complexity algorithms are measured in terms of input length and a parameter, which is expected to be small. More precisely, a problem is *fixed-parameter tractable (FPT)* with respect to a parameter  $k$  if an instance of size  $n$  can be solved in time  $\mathcal{O}^*(f(k)) = \mathcal{O}(f(k) \cdot \text{poly}(n))$  for some function  $f$ . For more details we refer to monographs [14, 11]

**Related Work.** As noted by Björklund et al. [4], the  $K$ -CYCLE problem has been a central topic of graph theory since the 1960's (see [20] for some references). The special cases where  $k = 1$  and  $k = 2$  can be solved by breadth-first search and finding a flow of size 2 between two vertices, respectively. The special case where  $k = 3$  has also long been known to be solvable in linear time [15, 24]. By the work of Robertson and Seymour on DISJOINT PATHS [28], for any constant  $k$ ,  $K$ -CYCLE is solvable in polynomial time. Kawarabayashi [20] showed that  $K$ -CYCLE is solvable in polynomial time also when  $k = \mathcal{O}((\log \log n)^{1/10})$ , where  $n = |V(G)|$ . The best known randomized algorithm for  $K$ -CYCLE was given by Björklund et al. [4], and runs in time  $\mathcal{O}^*(2^k)$ . Recently, Wahlström [31] gave an alternative algorithm that solves  $K$ -CYCLE in time  $\mathcal{O}^*(2^k)$  and showed that the problem admits a polynomial compression. We note that for directed graphs,  $K$ -CYCLE is NP-hard already when  $k = 2$  [18], and that other variants of  $K$ -CYCLE have also been considered in the literature. For example, Kawarabayashi et al. [21] gave an algorithm for detecting a  $K$ -cycle whose length has a given parity, and Kobayashi et al. [33] gave an algorithm for detecting an induced  $K$ -cycle in a planar graph.

The COLORFUL CYCLE problem was introduced in the breakthrough work on color coding by Alon et al. [1]. In that paper, the authors proposed an  $\mathcal{O}^*(2^k)$ -time deterministic algorithm for COLORFUL CYCLE. Although the problem became widely known, faster algorithms have not been obtained. Recently, Kowalik et al. [23] explained the source of difficulty: they showed that unless the Set Cover Conjecture [10] is false, there does not exist a constant  $\epsilon > 0$  such that COLORFUL CYCLE can be solved in time  $\mathcal{O}^*((2 - \epsilon)^k)$ . The conjecture states that there does not exist a constant  $\epsilon > 0$  such that SET COVER can be solved in time  $\mathcal{O}^*((2 - \epsilon)^n)$ , where  $n$  is the size of the universe. Finally, we note that the  $\ell$ -PATH problem has been extensively studied in the field of parameterized complexity, and there has been a race towards obtaining the fastest algorithm that solves it [25, 8, 1, 13, 19, 9, 22, 32, 2, 3, 17, 16, 30, 35]. Currently, the fastest algorithm (randomized) for  $\ell$ -PATH runs in time  $\mathcal{O}^*(1.66^\ell)$  [2, 3].

**Our Contribution.** Our main result is a randomized algorithm solving LIST  $K$ -CYCLE in time  $\mathcal{O}^*(2^k)$ , matching the best known running time in which one can solve  $K$ -CYCLE [4] as well as COLORFUL CYCLE [1]. We note that our algorithm can also be used to find a shortest cycle among solutions. Moreover, since LIST  $K$ -CYCLE generalizes COLORFUL CYCLE, we conclude that unless the Set Cover Conjecture is false, our algorithm is essentially optimal. To complement our main result, we also show that EXACT LIST  $K$ -CYCLE is solvable in time  $\mathcal{O}^*(2^k 1.66^{\ell-k})$ . Our results integrate three related algebraic approaches: the “unlabel inessential elements” used to solve  $K$ -CYCLE [4], the “double-labeling” used to solve a problem called GRAPH MOTIF [6], and the “partitioned single-labeling” used to solve  $\ell$ -PATH and HAMILTONICITY [2, 3]. More precisely, our main result integrates the first two approaches, while our EXACT LIST  $K$ -CYCLE algorithm integrates the latter two.

We remark that the algebraic technique developed in [2, 3] became a standard tool to develop parameterized algorithms (see, e.g., [11, 5, 27, 26]), yet the other two techniques not well known. To the best of our knowledge, the specific technique of [4] has no known additional applications, and the only additional applications of [6] are given in [7, 34].

## 2 Preliminaries

For  $q \in \mathbb{N}$ , let  $[q] = \{1, 2, \dots, q\}$  and  $[q]_0 = \{0, 1, \dots, q\}$ . For a function  $g : A \rightarrow [q]_0$ , let  $\text{count}(g) = |\{a \in A : g(a) \neq 0\}|$ . For  $(i, j), (i', j') \in \mathbb{N} \times \mathbb{N}$ , we say that  $(i, j)$  is smaller than  $(i', j')$ , if either (a)  $i < i'$  or (b)  $i = i'$  and  $j < j'$ . A *fixed-point-free involution* is a permutation that is its own inverse and has no fixed points. For a function  $f : A \rightarrow B$  and  $S \subseteq A$ , let  $f|_S : S \rightarrow B$  be the function such that for all  $s \in S$ ,  $f|_S(s) = f(s)$ .

For a graph  $G$ , let  $V(G)$  and  $E(G)$  denote its vertex-set and edge-set, respectively. For  $v \in V(G)$ , let  $N(v)$  denote its neighbor-set. A *walk* in a graph  $G$  is a sequence of vertices  $v_1 \dots v_\ell$  such that  $\{v_i, v_{i+1}\} \in E(G)$  for all  $i \in [\ell - 1]$ . For a graph  $G$  and  $e \in E(G)$ , we use  $G - e$  to denote the graph with vertex set  $V(G)$  and edge set  $E(G) \setminus \{e\}$ . For a graph  $G$  and  $V' \subseteq V(G)$ , we use  $E(V')$  to denote the set  $\{\{u, v\} \in E(G) \mid u, v \in V'\}$ . For a walk  $W$ , we use  $V(W)$  and  $E(W)$  to denote the set of vertices and edges, respectively, contained in  $W$ .

## 3 An Algorithm for List $K$ -Cycle

In this section we show that albeit the “list” requirement, the time in which one can solve LIST  $K$ -CYCLE matches the best known time in which one can solve both  $K$ -CYCLE and COLORFUL CYCLE. That is, we develop an algorithm solving LIST  $K$ -CYCLE in time  $\mathcal{O}^*(2^k)$ .

Our algorithm is based on the algebraic technique which underlies the algorithm for  $K$ -CYCLE by Björklund et al. [4], where one associates a polynomial over a finite field with the structure that should be found. Yet, there are essential differences between our algorithm and the one in [4]. To cancel potential solutions that “visit” vertices in  $V(G) \setminus K$  more than once, Björklund et al. [4] rely on a pairing argument whose correctness is based on the requirement that computations are performed over a field of characteristic 2 – our algorithm also relies on this pairing argument. However, to ensure that each vertex in  $K$  is encountered exactly once while attempting to construct a solution, Björklund et al. [4] explicitly store the set of vertices from  $K$  that have been encountered so far. This approach cannot be taken by our algorithm, since while constructing a solution, we would have stored each vertex in  $K$  that has been encountered so far as well as its color. This could lead to an algorithm with running time  $\mathcal{O}^*(\binom{|K|}{k} \binom{k^*}{k})$ . This solution does not even show that LIST  $K$ -CYCLE is FPT when parameterized by  $k$  (since  $|K|$  and  $k^*$  could be significantly larger than  $k$ ).

An  $\mathcal{O}^*(4^k)$ -time algorithm for LIST  $K$ -CYCLE can be obtained as follows. We mark each color once it is visited to ensure that it is not visited again via the labeling-based approach of Björklund et al. [2, 3] (since  $k^*$  can be significantly larger than  $k$ ) at the cost of a factor of  $\mathcal{O}^*(2^k)$  in the running time. Here, we also use this labeling-based approach on top of the algorithm for  $K$ -CYCLE by Björklund et al. [4] to ensure that the vertices that marked colors are not visited more than once (at the cost of a factor of  $\mathcal{O}^*(2^k)$  in the running time). The remaining vertices in  $V(G)$  are treated in the same manner as the algorithm for  $K$ -CYCLE by Björklund et al. [4] does (at the cost of a polynomial factor). However, this is still not sufficient, since our purpose is to match the best known running times in which one can solve  $K$ -CYCLE and COLORFUL CYCLE. By using the double-labeling-based approach introduced by Björklund et al. [6] to solve the GRAPH MOTIF problem, we are able to circumvent the need to keep track of colors and vertices that mark these colors separately.

**Initialization.** Instead of solving LIST  $K$ -CYCLE, we will solve the following problem: Given an  $n$ -vertex undirected graph  $G$ , a subset  $K \subseteq V(G)$ , a list-coloring  $L : K \rightarrow 2^{\{1,2,\dots,k^*\}}$  for some  $k^* \in \mathbb{N}$ , a parameter  $k \in \mathbb{N}$  and two vertices  $s, t \in V(G)$ , LIST  $K$ -PATH asks whether one can assign a color to each vertex in  $K$  so that  $G$  would have a simple path between  $s$  and  $t$  containing exactly  $k$  vertices from  $K$  and these vertices have distinct colors.

To solve LIST  $K$ -CYCLE in time  $\mathcal{O}^*(2^k)$ , it is sufficient to show that one can solve LIST  $K$ -PATH in time  $\mathcal{O}^*(2^k)$ . Indeed, an instance  $(G, K, L, k^*, k)$  is a YES-instance of LIST  $K$ -CYCLE if and only if there is an edge  $e = \{s, t\} \in E(G)$  such that  $(G - e, K, L, k^*, k, s, t)$  is a YES-instance of LIST  $K$ -PATH.

**Structures.** We define some notations which are essential for our algorithm. Let the input instance of LIST  $K$ -PATH be  $(G, K, L, k^*, k, s, t)$ . For  $q \in [n] \setminus \{1\}$  and  $u, v \in V(G)$ , a function  $f : [q] \rightarrow V(G)$  is called a  $(q, u, v)$ -function if  $f$  satisfies the following properties: (i)  $f(1) = s$ ,  $f(q-1) = u$  and  $f(q) = v$ , (ii) for all  $i \in [q-1]$ ,  $\{f(i), f(i+1)\} \in E(G)$ , and (iii) for all  $i \in [q-2]$  such that  $f(i) = f(i+2)$  it holds that  $f(i+1) \notin K$ . Observe that the function  $f$  defines a walk in  $G$ , and that if  $f$  is injective, it defines a simple path in  $G$ . We will use  $(q, u, v)$ -functions to construct a simple path that is a solution. We need to consider walks rather than simple paths as during the construction, we do not want to keep information regarding vertices that have already been visited – later we will use pairing arguments (in Lemma 6) to cancel such incorrect constructions. To make the pairing argument work, we will need property (iii) of  $f$ , which is also the reason why we explicitly mention  $u$ , one vertex before the last vertex we have visited, in the definition of  $f$ .

In addition to  $f$ , to know whether we are constructing a solution, we need to know which colors in  $[k^*]$  have been visited. For this purpose, we need the following term. For  $k' \in [k]_0$ , a  $(q, u, v, k')$ -structure is a pair  $(f, g)$  such that (a)  $f$  is a  $(q, u, v)$ -function, (b)  $g : [q] \rightarrow [k^*]_0$  is a function such that  $\text{count}(g) = k'$ , and (c) for every  $i \in [q]$ , if  $f(i) \in K$  then  $g(i) \in L(f(i))$  and otherwise  $g(i) = 0$ . In other words, for each occurrence of a vertex in  $K$  which we have visited,  $g$  specifies a color, and overall, it specifies exactly  $k'$  occurrences of colors. To simplify the presentation, given a  $(q, u, v, k')$ -structure  $(f, g)$ , we define  $\text{col}(g) = \{i \in [q] : g(i) \neq 0\}$ . That is,  $\text{col}(g)$  is the set of indices associated with vertices in  $K$  (which receive colors). When  $v = t$  and  $k' = k$ , for any  $u \in V(G)$ , a  $(q, u, v, k')$ -structure is also called a  $q$ -structure.

► **Definition 1.** A  $q$ -structure  $(f, g)$  is called *good  $q$ -structure* if (i)  $f$  is injective, and (ii) for all  $i, j \in \text{col}(g)$ ,  $g(i) \neq g(j)$  (i.e.  $g|_{\text{col}(g)}$  is injective).

► **Observation 2.** *The input instance of LIST  $K$ -PATH is a YES-instance if and only if there exists a good  $q$ -structure.*

**Labels.** We are now going to assign labels to some of the occurrences of vertices of a  $(q, u, v, k')$ -structure. Later we will use these labels to ensure that potential structures that are not good get cancelled when computations are performed over a field of characteristic 2. More precisely, we will swap distinct labels assigned to occurrences of vertices that obtained the same color, an ability that will be used by one of our pairing arguments (see Lemma 6).

We use labels for each position in a walk which corresponds to a vertex from  $K$ . For a  $(q, u, v, k')$ -structure  $(f, g)$  and  $h : \text{col}(g) \rightarrow [k]$ , we say that a triple  $(f, g, h)$  is a  $(q, u, v, k')$ -labeling. When  $(f, g)$  is a  $q$ -structure or a good  $q$ -structure, we say that  $(f, g, h)$  is a  $q$ -labeling or good  $q$ -labeling, respectively. Finally, if  $h$  is bijective, we say that the entire triple  $(f, g, h)$  is bijective. We give special consideration to bijective functions  $h$  since such functions ensure that all good  $q$ -labelings survive and all the  $q$ -labelings which are not good cancel each other in the polynomial we construct later.

Next, we define two sets of  $q$ -labelings. First, we let  $B_1^q$  denote the set of all bijective  $q$ -labelings. Now, we let  $B_2^q \subseteq B_1^q$  denotes the set of all bijective good  $q$ -labelings. Given a good  $q$ -structure  $(f, g)$ , we can arbitrarily order the elements in  $\text{col}(g)$ , and let  $h$  assign to each element in  $\text{col}(g)$  its location in this order. This results in a bijection  $h$ , and overall, in a bijective  $q$ -labeling  $(f, g, h)$ . Thus, by Observation 2, we get the following observation.

► **Observation 3.** *The input instance of LIST  $K$ -PATH is a YES-instance if and only if there exists  $q \in [n] \setminus \{1\}$  such that  $B_2^q \neq \emptyset$ .*

**Monomials.** We will now associate a monomial with each  $(q, u, v, k')$ -labeling. Towards this, we introduce the following variables. First, for each  $e \in E(G)$ , we introduce a variable  $x_e$ . Now, for each  $v \in K$  and color  $a \in L(v)$ , we introduce a variable  $y_{v,a}$ . Finally, for each color  $a \in [k^*]$  and label  $i \in [k]$ , we introduce a variable  $z_{a,i}$ . Let  $N$  be the total number of variables created. Notice that  $N = n^{O(1)}$ . We are now ready to define monomial for each  $(q, u, v, k')$ -labeling.

► **Definition 4.** For a  $(q, u, v, k')$ -labeling  $(f, g, h)$ , the *monomial of  $(f, g, h)$* , denoted by  $m(f, g, h)$ , is defined as follows.

$$m(f, g, h) = \left( \prod_{i \in [q-1]} x_{\{f(i), f(i+1)\}} \right) \cdot \left( \prod_{i \in \text{col}(g)} y_{f(i), g(i)} \cdot z_{g(i), h(i)} \right).$$

On the one hand, observe that if  $(f, g, h)$  is a bijective good  $q$ -labeling, then it can be uniquely recovered from its monomial. That is, we have the following observation.

► **Observation 5.** *For any  $q \in [n] \setminus \{1\}$  and  $(f, g, h) \in B_2^q$ , there does not exist  $(f', g', h') \in B_1^q \setminus \{(f, g, h)\}$  such that  $m(f, g, h) = m(f', g', h')$ .*

When the input instance  $(G, K, L, k^*, k, s, t)$  is a NO-instance of LIST  $K$ -PATH, then we can get a fixed-point free involution on  $B_1^q$  with some properties:

► **Lemma 6.** *Let  $(G, K, L, k^*, k, s, t)$  be a NO-instance of LIST  $K$ -PATH and  $B_1^q$  is the set of all bijective  $q$ -labeling for any  $q \in [n] \setminus \{1\}$ . Then there exists a fixed-point-free involution  $\phi : B_1^q \rightarrow B_1^q$  such that for all  $(f, g, h) \in B_1^q$ ,  $m(f, g, h) = m(\phi(f, g, h))$ .*

**Proof.** Since  $(G, K, L, k^*, k, s, t)$  is a NO-instance, by Observation 3, we have that  $B_2^q = \emptyset$ . Let  $(f, g, h) \in B_1^q$ . We choose a pair  $(i, j) \in [q] \times [q]$  and by carefully modifying  $f, g$  and  $h$  between  $i$  and  $j$  we get a  $q$ -labeling that can be mapped to  $(f, g, h)$  by  $\phi$ . We will consider several cases and in each case, we assume that the conditions of the previous cases are

false, define  $\phi(f, g, h)$ , and prove that  $\phi(f, g, h) \neq (f, g, h)$ ,  $m(f, g, h) = m(\phi(f, g, h))$  and  $\phi(\phi(f, g, h)) = (f, g, h)$ . Since  $(f, g, h) \notin B_2^q$ , we know that either (a) there exist  $i, j \in \text{col}(g)$ ,  $i < j$ , such that  $g(i) = g(j)$  or (b) there exist  $i, j \in [q]$ ,  $i < j$  such that  $f(i) = f(j)$ .

**Case 1: There exist  $i, j \in \text{col}(g)$ ,  $i < j$ , such that  $g(i) = g(j)$ .** Among all such pairs, that we call bad pairs of type 1, let  $(i, j)$  be the smallest one. We define  $(f', g', h') = \phi(f, g, h)$  as follows. First, we simply let  $f' = f$  and  $g' = g$ . Now, we let  $h'(i) = h(j)$  and  $h'(j) = h(i)$ , and for all  $r \in \text{col}(g) \setminus \{i, j\}$ , we let  $h'(r) = h(r)$ . Since  $(f', g') = (f, g)$  is a  $q$ -structure and  $h'$  is bijective, we have that  $(f, g, h') \in B_1^q$ . Since  $h$  is bijective, we have that  $h' \neq h$ , and therefore  $\phi(f, g, h) \neq (f, g, h)$ . Moreover,  $z_{g(i), h(i)} = z_{g'(j), h'(j)}$  and  $z_{g(j), h(j)} = z_{g'(i), h'(i)}$ , and therefore  $m(f, g, h) = m(\phi(f, g, h))$ . Finally, because  $(f, g) = (f', g')$ , the sets of bad pairs of type 1 of  $(f, g, h)$  and  $(f', g', h')$  are same. Also, by swapping the values assigned to  $i$  and  $j$  in  $h'$ , we obtain  $h$ , and so we have that  $\phi(\phi(f, g, h)) = (f, g, h)$ .

**Case 2: There exist two indices  $i, j \in \text{col}(g)$ ,  $i < j$ , such that  $f(i) = f(j)$ .** Among all such pairs, that we call bad pairs of type 2, let  $(i, j)$  be the smallest one. Since  $i \in \text{col}(g)$ ,  $f(i) \in K$ . We define  $(f', g', h') = \phi(f, g, h)$  as follows. First, we simply let  $f' = f$ . Now, we let  $g'(i) = g(j)$  and  $g'(j) = g(i)$ , and for all  $r \in [q] \setminus \{i, j\}$ , we let  $g'(r) = g(r)$ . Finally, we let  $h'(i) = h(j)$ ,  $h'(j) = h(i)$ , and for all  $r \in \text{col}(g) \setminus \{i, j\}$ , we let  $h'(r) = h(r)$ . Again, it is clear that  $\phi(f, g, h) \in B_1^q$ . Because Case 1 is false, we have that  $g(i) \neq g(j)$ , and therefore  $\phi(f, g, h) \neq (f, g, h)$ . Moreover, since  $f(i) = f(j)$ , it holds that  $y_{f(i), g(i)} = y_{f'(j), g'(j)}$  and  $y_{f(j), g(j)} = y_{f'(i), g'(i)}$ , and it also holds that  $z_{g(i), h(i)} = z_{g'(j), h'(j)}$  and  $z_{g(j), h(j)} = z_{g'(i), h'(i)}$ . Therefore,  $m(f, g, h) = m(\phi(f, g, h))$ . By our definition of  $g'$ , we have that there are no bad pairs of type 1. Moreover, our definitions of  $f'$  and  $g'$  ensures that the sets of bad pairs of type 2 of  $(f, g, h)$  and  $(f', g', h')$  are the same. Also, by swapping the values assigned to  $i$  and  $j$  by  $g'$  as well as  $h'$ , we get  $g$  and  $h$  respectively, and so we have that  $\phi(\phi(f, g, h)) = (f, g, h)$ .

**Case 3: Cases 1 and 2 are false.** Let  $I = [q]$ . If there exists a pair  $(i', j') \in I \times I$ ,  $i' < j'$ , such that  $f(i') = f(j')$ , then we choose such a pair with the following inductive priorities in the order: (1)  $i'$  is minimized and (2)  $j'$  is maximized. If  $f(i')f(i'+1) \dots f(j')$  is not a palindrome, then we set  $(i, j) = (i', j')$ . Otherwise, we set  $I := I \setminus [j']$  and continue the process of choosing a pair as above from  $I \times I$ . Observe that since Cases 1 and 2 are false and yet the input instance is a NO-instance, at least one pair  $(i', j')$  will be chosen.

First we show that we succeed in finding a pair  $(i, j)$ . Towards this we first show that for any pair  $(i', j')$  considered by the above process but not considered as the pair  $(i, j)$ , we have that  $f(i'), f(i'+1), \dots, f(j') \notin K$ . Suppose there is a vertex  $w \in K$  appearing in the sequence  $f(i')f(i'+1) \dots f(j')$ . We know that  $f(i')f(i'+1) \dots f(j')$  is a palindrome and Case 2 is not applicable. This implies that  $w$  appears only once and it is in the middle of the palindrome  $f(i')f(i'+1) \dots f(j')$ . But then this will contradict property (iii) of the  $(q, u, t)$ -function  $f$  (since  $(f, g)$  is a  $q$ -structure,  $f$  is a  $(q, u, t)$ -function for some  $u \in V(G)$ ). Let  $(i_1, j_1), \dots, (i_\ell, j_\ell)$  be the pairs considered in the above process in the given order. Suppose  $(i_\ell, j_\ell) \neq (i, j)$  (that is, the process terminated before we set  $(i, j)$ ). Then,  $f|_I$  is injective, where  $I = [q] \setminus (\bigcup_{r \in [q]} \{i_r, i_r + 1, \dots, j_r\})$ . Also we know that for  $i, j \in \text{col}(g)$ ,  $i < j$ ,  $g(i) \neq g(j)$ , because Case 1 is not applicable. Then this implies that  $(f|_I, g)$  is a good  $q'$ -structure for some  $q' < q$ , which is a contradiction because  $(G, K, L, k^*, k, s, t)$  is a NO-instance. Thus we have shown that the above process succeeds in finding  $(i, j)$  such that  $f(i) \dots f(j)$  is not a palindrome.

We define  $(f', g', h') = \phi(f, g, h)$  as follows. For every index  $r \in [q]$  such that  $r < i$  or  $r > j$ , we let  $f'(r) = f(r)$ ,  $g'(r) = g(r)$  and  $h'(r) = h(r)$ . Next, we are going to redirect the

subwalk from  $i$  to  $j$  to be from  $j$  to  $i$ , adjusting the values assigned by  $f, g$  and  $h$  accordingly. Formally, for each index  $i \leq r \leq i + \lfloor (j-i)/2 \rfloor$ , define  $s(r) = j - (r-i)$  and  $s(j - (r-i)) = r$ . Now, for each index  $i \leq r \leq j$ , let  $f'(r) = f(s(r))$ ,  $g'(r) = g(s(r))$  and  $h'(t) = h(s(t))$ .

Because  $f(i)f(i+1) \cdots f(j)$  is not a palindrome, we have that  $\phi(f, g, h) \neq (f, g, h)$ . Notice that  $f$  and  $f'$  corresponds to two walks in the graph  $G$ . Observe that because  $f(i) = f(j)$ , redirecting the subwalk between  $i$  and  $j$  does not change the edges which the walk contains (only the order in which we visit them changes), and since upon changing the location of an occurrence of a vertex, we update all of the three functions  $f, g$  and  $h$  accordingly, we have that  $m(f, g, h) = m(\phi(f, g, h))$ . We show that  $f'$  is a  $(q, u', t)$ -function for some  $u'$ . Properties (i) and (ii) of  $(q, u', t)$ -function trivially hold. The only occurrences of vertices whose adjacent occurrences of vertices might change are  $f'(i)$  and  $f'(j)$ , and because  $f'(i) \notin K$  (since Case 2 not applicable), it is still true that for all  $r \in [q-2]$  such that  $f'(r) = f'(r+2)$  it holds that  $f'(r+1) \notin K$ . Hence  $(f', g', h') = \phi(f, g, h) \in B_1^q$ .

Finally we prove that  $\phi(f', g', h') = (f, g, h)$ . Notice that for  $(f', g', h')$ , Cases 1 and 2 are not applicable. Looking at  $(f', g', h')$  and applying the process above in Case 3 will result in the same pair  $(i, j)$ , since the updates that are performed with respect to  $I$  only concern indices before  $i$  (which were not change), and the only occurrences of vertices whose location has changed lie between  $i$  and  $j$ , and therefore they do not affect the minimality of  $i$  and maximality of  $j$  relevant to the choice of  $(i, j)$ . Thus, since by redirecting the subwalk between  $i$  and  $j$  yet again we obtain the original walk, and hence we conclude that  $\phi(\phi(f, g, h)) = (f, g, h)$ .  $\blacktriangleleft$

Now, for all  $q \in [n]$ , we define a polynomial that is evaluated over the finite field  $\mathbb{F}_p$ , where  $p = 2^{\lceil \log(3(n+2k)) \rceil}$ .

► **Definition 7.**  $P^q = \sum_{(f,g,h) \in B_1^q} m(f, g, h)$ .

► **Lemma 8.**  $(G, K, L, k^*, k, s, t)$  is a YES-instance of LIST  $K$ -PATH if and only if there is  $q \in [n] \setminus \{1\}$  such that  $P^q$  is not identically 0.

**Proof.** Suppose  $(G, K, L, k^*, k, s, t)$  is a YES-instance and  $n = |V(G)|$ . Then Observation 3, there is a  $q \in [n] \setminus \{1\}$  such that  $B_2^q \neq \emptyset$ . Then by Observation 5  $P^q$  contains a unique monomial corresponding to  $(f, g, h)$  where  $(f, g, h) \in B_2^q$ . This implies that  $P^q \neq 0$ .

Suppose  $(G, K, L, k^*, k, s, t)$  is a NO-instance, then by Lemma 6 and the fact that  $\mathbb{F}_p$  has characteristic 2, we can conclude that for all  $q \in [n] \setminus \{1\}$ ,  $P^q \equiv 0$ .  $\blacktriangleleft$

Thus, it remains to determine whether at least one polynomial  $P^q$  is not identically 0.

**Evaluation.** Given  $I \subseteq [k]$ , a  $(q, u, v, k', I)$ -labeling  $(f, g, h)$  is a  $(q, u, v, k')$ -labeling whose set of assigned labels belongs to  $I$  (i.e., the image of  $h$  is a subset of  $I$ ). Now, we define polynomials whose evaluations, which will be done below, can be considered as “intermediate” steps towards evaluating  $P^q$ . To this end, we let  $S(q, u, v, k', I)$  denote the set of all  $(q, u, v, k', I)$ -labelings.

► **Definition 9.**  $P^q(u, v, k', I) = \sum_{(f,g,h) \in S(q,u,v,k',I)} m(f, g, h)$ .

By the inclusion-exclusion principle and because  $\mathbb{F}_p$ , the field over which we evaluate polynomials, has characteristic 2, we have the following observation.

► **Observation 10.**  $P^q = \sum_{I \subseteq [k]} \sum_{u^* \in N(t)} P^q(u^*, t, k, I)$ .



It is easier to compute polynomials of the form  $P^q(u^*, t, k, I)$  rather than the polynomial  $P^q$  since then we do not need to ensure that labels are not repeated, but only need to ensure that certain labels are not used at all. We next show that a polynomial of the form  $P^q(u^*, t, k, I)$  can be computed in polynomial time by using a procedure based on dynamic programming.

► **Lemma 11** ( $\star^1$ ). *Let  $I \subseteq [k]$ ,  $u^* \in N(t)$  and let  $\{x_1, \dots, x_N\}$  be the set of variables in  $P^q(u^*, t, k, I)$ . Let  $a_1, a_2, \dots, a_N \in \mathbb{F}_p$  denote the chosen values for the  $N$  variables. Then, the polynomial  $P^q(u^*, t, k, I)$  can be evaluated at  $(a_1, a_2, \dots, a_N)$  in time  $N^{\mathcal{O}(1)}$ .*

Combining Observation 10 with Lemma 11, and since there are  $2^k$  subsets  $I$  of  $[k]$ , we have, the following result.

► **Lemma 12.** *Given an assignment of values from  $\mathbb{F}_p$  to the variables of  $P^q$ , the polynomial  $P^q$  can be evaluated in time  $2^k N^{\mathcal{O}(1)}$  and in space  $N^{\mathcal{O}(1)}$ .*

**The Algorithm.** To calculate the number of evaluations we should perform, we rely on the following well-known result, proved in [12, 29, 36].

► **Lemma 13.** *Let  $p(x_1, x_2, \dots, x_m)$  be a nonzero polynomial of  $n$  variables and total degree at most  $d$  over the finite field  $\mathbb{F}$ . Then, for  $a_1, a_2, \dots, a_n \in \mathbb{F}$  selected independently and uniformly at random:  $\Pr(p(a_1, a_2, \dots, a_n) \neq 0) \geq 1 - d/|\mathbb{F}|$ .*

We are now ready to conclude this section with our main result.

► **Theorem 14.** *There is a polynomial space randomized algorithm for LIST  $K$ -CYCLE running in time  $\mathcal{O}^*(2^k)$  with one sided constant error probability.*

**Proof.** As explained earlier, we actually solve LIST  $K$ -PATH. Our algorithm is defined as follows. Let  $(G, K, L, k^*, k, s, t)$  be the input instance and  $n = |V(G)|$ . It examines every length  $q \in [n]$ . Now, consider a specific length  $q$ . First, uniformly at random, it chooses  $N$  values from  $\mathbb{F}_p$  to be assigned to the variables of  $P^q$  (recall that  $N$  is the number of variables which is bounded by  $n^{\mathcal{O}(1)}$ ). Then, it evaluates  $P^q$  accordingly using the computation presented in Lemma 12. Finally, if the result is not zero, it accepts. Eventually, if for every length  $q$  the result is zero, it rejects.

By Lemma 8, if the input instance of LIST  $K$ -PATH is a NO -instance, for every length  $q$  the result of evaluating  $P^q$  is zero, and therefore the algorithm will reject. Now, suppose that the input instance of LIST  $K$ -PATH is a YES-instance. By Lemma 8, there exists  $q \in [n]$  such that  $P^q$  is not identically zero. Observe that  $P^q$  is a polynomial of total degree at most  $n + 2k$ . Thus, by Lemma 13, the result of evaluating  $P^q$  is not zero with probability at least  $2/3$ . Therefore, the algorithm accepts with probability at least  $2/3$ . Clearly, the probability of success can be increased via multiple runs, or, in a more direct manner, by choosing a larger field (choosing a field of size  $p = 2^{\lceil \log(c'(n+2k)) \rceil}$  for some constant  $c'$  will result in a success probability of at least  $1 - 1/c'$ ). ◀

## 4 An Algorithm for Exact List $K$ -Cycle

In this section we show that EXACT LIST  $K$ -CYCLE can be solved in time  $\mathcal{O}^*(2^k 1.66^{\ell-k})$ . On the one hand, EXACT LIST  $K$ -CYCLE generalizes COLORFUL CYCLE – indeed, COLORFUL

<sup>1</sup> Due to space constraints, proofs of results marked with  $\star$  are omitted.



CYCLE is the special case of EXACT LIST  $K$ -CYCLE where  $K = V(G)$ ,  $k = k^* = \ell$  and for all  $v \in K$  it holds that  $|L(v)| = 1$ . Therefore, unless the Set Cover Conjecture is false, we cannot obtain an algorithm that runs in time  $\mathcal{O}^*((2 - \epsilon)^k)$  even if  $k = \ell$ . On the other hand, EXACT LIST  $K$ -CYCLE generalizes  $\ell$ -PATH and HAMILTONICITY (which is a special case of  $\ell$ -PATH where  $\ell = |V(G)|$ ), and there are randomized algorithms for  $\ell$ -PATH and HAMILTONICITY running in times  $\mathcal{O}^*(1.66^\ell)$  and  $\mathcal{O}^*(1.66^{|V(G)|})$ , respectively [3, 2]. We present a multivariate algorithm for EXACT LIST  $K$ -CYCLE that can be viewed as a compromise between the dependency on  $k$  and the dependency on  $\ell$ . Our approach employs the double-labeling-based approach of Björklund et al. [6], used to solve the GRAPH MOTIF problem, on top of the technique underlying the  $\ell$ -PATH algorithm of Björklund et al. [3].

Due to similarities between this algorithm and the one given in the previous section, we present it in a more concise manner. Again, by the explanation given in Section 3, it is sufficient to solve the following problem, called EXACT LIST  $K$ -PATH, in time  $\mathcal{O}^*(2^k 1.66^{\ell-k})$ . Given an  $n$ -vertex undirected graph  $G$ , a set of vertices  $K \subseteq V(G)$ , a list-coloring  $L : K \rightarrow 2^{\{1,2,\dots,k^*\}}$  for some  $k^* \in \mathbb{N}$ , parameters  $k, \ell \in \mathbb{N}$  and two vertices  $s, t \in V(G)$ , and this problem asks whether one can assign a color to each vertex in  $K$  so that  $G$  would have a simple path of length  $\ell$  between  $s$  and  $t$  containing exactly  $k$  vertices from  $K$  and these vertices have distinct colors.

In most of this section, we will assume that we are given a partition  $(V_1, V_2)$  of  $V(G) \setminus K$ , which will be computed later. Accordingly, we define the PARTITIONED EXACT LIST  $K$ -PATH ( $K$ -PEL PATH) problem as the EXACT LIST  $K$ -PATH problem where one is also given parameters  $\ell_1, \ell_2 \in [\ell]$ , and the solution is also required to contain exactly  $\ell_1$  vertices from  $V_1$  and  $\ell_2$  edges from  $E(V_2)$ .

**Structures and Labels.** Given  $q \in [\ell] \setminus \{1\}$ ,  $q_1 \in [\ell_1]_0$ ,  $q_2 \in [\ell_2]_0$ , vertices  $u, v \in V(G)$  and  $k' \in [k]$ , a function  $f : [q] \rightarrow V(G)$  is called a  $(q, q_1, q_2, u, v, k')$ -function if the following properties hold:

- (i)  $f(1) = s$ ,  $f(q-1) = u$  and  $f(q) = v$ ,
- (ii) for all  $i \in [q-1]$ ,  $\{f(i), f(i+1)\} \in E(G)$ ,
- (iii) for all  $i \in [q-2]$  such that  $f(i) = f(i+2) \in V_2$  it holds that  $f(i+1) \in V_2$ ,
- (iv)  $q_1 = |V_1(f)|$ , where  $V_1(f) = \{i \in [q] : f(i) \in V_1\}$ ,
- (v)  $q_2 = |E_2(f)|$ , where  $E_2(f) = \{(i, i+1) : i \in [q-1], \{f(i), f(i+1)\} \subseteq V_2\}$ ,
- (vi)  $k' = |\{i \in [q] : f(i) \in K\}|$ .

Observe that  $V_1(f)$  and  $E_2(f)$  are sets of indices. In addition to  $f$ , to know whether we are constructing a solution, we need to know which colors in  $[k^*]$  have been used. For this purpose, we need the following term. For a  $(q, q_1, q_2, u, v, k')$ -function  $f$  and a function  $g : [q] \rightarrow [k^*]_0$ , the pair  $(f, g)$  is called  $(q, q_1, q_2, u, v, k')$ -structure if the following condition holds: for any  $i \in [q]$ , if  $f(i) \in K$  then  $g(i) \in L(f(i))$  and otherwise  $g(i) = 0$ . In other words, for each occurrence of a vertex in  $K$  which we have visited,  $g$  specifies a color, and overall, since  $f$  is a  $(q, q_1, q_2, u, v, k')$ -function,  $g$  specifies exactly  $k'$  occurrences of colors. To simplify the presentation, for a  $(q, q_1, q_2, u, v, k')$ -structure  $(f, g)$ , we define  $\text{col}(g) = \{i \in [q] : g(i) \neq 0\}$ . That is,  $\text{col}(g)$  is the set of indices associated with vertices in  $K$  (which receive colors). For any vertex  $u \in V(G)$ , a  $(\ell, \ell_1, \ell_2, u, t, k)$ -structure is also called a *final structure* ( $f$ -structure). When both  $f$  and  $g|_{\text{col}(g)}$  are also injective, we say that it is an *excellent final structure* ( $ef$ -structure). By the definition of  $ef$ -structures, we have the following observation.

► **Observation 15.** *The input instance of  $K$ -PEL PATH is a YES-instance if and only if there exists an  $ef$ -structure.*

As before, the main idea is to use labels for specific elements in  $(q, q_1, q_2, u, v, k')$ -structures. Here we use labels on  $\text{col}(g) \cup V_1(f) \cup E_2(f)$ . For a  $(q, q_1, q_2, u, v, k')$ -structure  $(f, g)$  and a function  $h : \text{col}(g) \cup V_1(f) \cup E_2(f) \rightarrow [k + \ell_1 + \ell_2]$ , we say that the triple  $(f, g, h)$  is a  $(q, q_1, q_2, u, v, k')$ -labeling. When  $(f, g)$  is an  $f$ -structure or an  $ef$ -structure, we say that  $(f, g, h)$  is an  $f$ -labeling or an  $ef$ -labeling, respectively. Moreover, if  $g$  is bijective, we say that the triple  $(f, g, h)$  is bijective. Next, we define two sets of  $f$ -labelings. First, we let  $B_1$  denote the set of all bijective  $f$ -labelings. Now, we let  $B_2 \subseteq B_1$  denote the set of all bijective  $ef$ -labelings.

Observe that for any  $ef$ -structure  $(f, g)$  (which corresponds to a solution),  $|\text{col}(g) \cup V_1(f) \cup E_2(f)| = k + \ell_1 + \ell_2$ , and therefore the sizes of the domain and codomain of the function  $h$  are equal. Thus, given an  $ef$ -structure  $(f, g)$ , we can arbitrarily order the elements in  $\text{col}(g) \cup V_1(f) \cup E_2(f)$ , and let  $h$  assign to each element in  $\text{col}(g) \cup V_1(f) \cup E_2(f)$  its location in this order. This results in a bijective function  $h$ , and overall, in a bijective  $ef$ -labeling  $(f, g, h)$ . Thus, by Observation 15, we have the following observation.

► **Observation 16.** *The input instance of  $K$ -PEL PATH is a YES-instance if and only if  $B_2 \neq \emptyset$ .*

**Monomials.** Now we explain how to assign monomial for each  $(q, q_1, q_2, u, v, k')$ -labeling and show that the polynomial which is sum of monomials corresponding to  $f$ -labeling will give us the answer for  $K$ -PEL PATH through Polynomial Identity Testing (PIT). First we introduce the following variables. For each edge  $e \in E(G)$ , introduce a variable  $x_e$ . Now, for each vertex  $v \in V_1$  and label  $i \in [k + \ell_1 + \ell_2]$ , introduce a variable  $y_{v,i}$ , and for each edge  $e \in E(V_2)$  and label  $i \in [k + \ell_1 + \ell_2]$ , introduce a variable  $y_{e,i}$ . For each vertex  $v \in K$  and color  $a \in L(v)$ , introduce the variable  $z_{v,a}$ . Finally, for each color  $a \in [k^*]$  and label  $i \in [k + \ell_1 + \ell_2]$ , introduce the variable  $z_{a,i}$ . Let  $N$  be the total number of variables created.

Now, we associate a monomial with each  $(q, q_1, q_2, u, v, k')$ -labeling.

► **Definition 17.** Given a  $(q, q_1, q_2, u, v, w, k')$ -labeling  $(f, g, h)$ , the *monomial of  $(f, g, h)$* , denoted by  $m(f, g, h)$ , is defined as follows.

$$m(f, g, h) = \left( \prod_{i \in [q-1]} x_{\{f(i), f(i+1)\}} \right) \left( \prod_{i \in V_1(f)} y_{f(i), h(i)} \right) \left( \prod_{(i, i+1) \in E_2(f)} y_{\{f(i), f(i+1)\}, h((i, i+1))} \right) \\ \cdot \left( \prod_{i \in \text{col}(g)} z_{f(i), g(i)} \cdot z_{g(i), h(i)} \right).$$

If  $(f, g, h)$  is a bijective  $ef$ -labeling, then it can be uniquely recovered from its monomial. That is, we have the following observation.

► **Observation 18.** *For all  $(f, g, h) \in B_2$ , there does not exist  $(f', g', h') \in B_1 \setminus \{(f, g, h)\}$  such that  $m(f, g, h) = m(f', g', h')$ .*

For bijective  $f$ -labelings that are not  $ef$ -labelings, we have the following lemma.

► **Lemma 19.** *Let  $(G, K, L, k^*, k, s, t, \ell, V_1, V_2, \ell_1, \ell_2)$  be a NO-instance of  $K$ -PEL PATH and  $B_1$  is the set of all bijective  $f$ -labeling. Then there is a fixed-point-free involution  $\phi : B_1 \rightarrow B_1$  such that for all  $(f, g, h) \in B_1$ ,  $m(f, g, h) = m(\phi(f, g, h))$ .*

**Proof.** Since  $(G, K, L, k^*, k, s, t, \ell, V_1, V_2, \ell_1, \ell_2)$  is a NO-instance of  $K$ -PEL PATH, by Observation 16,  $B_2 = \emptyset$ . Let  $(f, g, h) \in B_1$ . We will consider several cases, and in each case, we assume that previous cases are false, define  $\phi(f, g, h)$ , and prove that  $\phi(f, g, h) \neq (f, g, h)$ ,  $m(f, g, h) = m(\phi(f, g, h))$  and  $\phi(\phi(f, g, h)) = (f, g, h)$ . Since  $B_2 = \emptyset$  and  $(f, g, h) \in B_1$ , we have that  $(f, g)$  is a  $f$ -structure but not an ef-structure. This implies that either  $f$  is not injective or  $g|_{\text{col}(g)}$  is not injective. This implies that there exist  $i, j \in [\ell], i < j$  such that either (a)  $f(i) = f(j)$  or (b)  $i, j \in \text{col}(g)$  and  $g(i) = g(j)$ .

**Case 1: There exists  $i, j \in [\ell], i < j$  such that  $f(i) = f(j) \in V_1$ .** Among all such pairs, called bad pairs of type 1, let  $(i, j)$  be the smallest one. We define  $(f', g', h') = \phi(f, g, h)$  as follows. First, let  $f' = f$  and  $g' = g$ . Now, let  $h'(i) = h(j)$  and  $h'(j) = h(i)$ , and for all  $r \in (\text{col}(g) \cup V_1(f) \cup E_2(f)) \setminus \{i, j\}$ , let  $h'(r) = h(r)$ . Since  $h$  is bijective, we have that  $h' \neq h$ , and therefore  $\phi(f, g, h) = (f', g', h') \neq (f, g, h)$ . Moreover,  $y_{f(i), h(i)} = y_{f'(j), h'(j)}$  and  $y_{f(j), h(j)} = y_{f'(i), h'(i)}$ , and therefore  $m(f, g, h) = m(\phi(f, g, h))$ . Notice that the smallest bad pair of type 1 in  $(f', g', h')$  is  $(i, j)$ . So by swapping the values assigned to  $i$  and  $j$  in  $h'$ , we obtain  $h$ , and hence we have that  $\phi(\phi(f, g, h)) = (f, g, h)$ .

**Case 2: There exists  $i, j \in [\ell], i < j$  such that  $f(i) = f(j) \in K$ .** Among all such pairs, that we call bad pairs of type 2, let  $(i, j)$  be the smallest one. We define  $(f', g', h') = \phi(f, g, h)$  as follows. First, let  $f' = f$ . Now, let  $g'(i) = g(j)$  and  $g'(j) = g(i)$ , and for all  $r \in [q] \setminus \{i, j\}$ , let  $g'(r) = g(r)$ . Finally, let  $h'(i) = h(j)$ ,  $h'(j) = h(i)$ , and for all  $r \in (\text{col}(g) \cup V_1(f) \cup E_2(f)) \setminus \{i, j\}$ , we let  $h'(r) = h(r)$ . Since  $h$  is bijective,  $h \neq h'$  and  $\phi(f, g, h) \neq (f, g, h)$ . Moreover, since  $f(i) = f(j)$ , it holds that  $z_{f(i), g(i)} = z_{f'(j), g'(j)}$  and  $z_{f(j), g(j)} = z_{f'(i), g'(i)}$ . It also holds that  $z_{g(i), h(i)} = z_{g'(j), h'(j)}$  and  $z_{g(j), h(j)} = z_{g'(i), h'(i)}$ . Therefore,  $m(f, g, h) = m(\phi(f, g, h))$ . Since there is no bad pair of type 1 in  $(f, g, h)$ , there is no bad pairs of type 1 in  $(f', g', h')$ . Notice that the is the smallest bad pair of type 2 in  $(f', g', h')$  is  $(i, j)$ . By swapping the values assigned to  $i$  and  $j$  by  $g'$  and  $h'$  we get  $g$  and  $h$ , and hence we have that  $\phi(\phi(f, g, h)) = (f, g, h)$ .

**Case 3: There exists  $i, j \in [\ell], i < j$  such that  $f(i) = f(j) \in V_2$ .** Among all such pairs, that we call bad pairs of type 3, let  $(i, j)$  be the smallest one. We have two sub-cases based  $f(i) \dots f(j)$  is a palindrome or not.

**Case 3(a):  $f(i) \dots f(j)$  is a palindrome.** Since  $f(i) \dots f(j)$  is a walk in a simple graph  $G$  and  $f(i) \dots f(j)$  is a palindrome, the length of the sequence  $f(i) \dots f(j)$  is odd and  $j - i$  is even. Let  $r = i + \frac{j-i}{2}$ . Notice that  $r$  is the middle index in the sequence  $i, i+1, \dots, j$ . Since Cases 1 and 2 are not applicable, we have that for all  $r' \in \{i, \dots, j\} \setminus \{r\}$   $f(r') \in V_2$ . Now consider the sequence  $f(r-1)f(r)f(r+1)$ . We know that  $f(r-1) = f(r+1) \in V_2$ . Thus by property (iii) of  $(q, q_1, q_2, u, v, k')$ -function, we have that  $f(r) \in V_2$ . Hence, we have that  $f(i), \dots, f(j) \in V_2$ .

Now, we define  $(f', g', h') = \phi(f, g, h)$  as follows. First, let  $f' = f$  and  $g' = g$ . Now, let  $h'((i, i+1)) = h((j-1, j))$ ,  $h'((j-1, j)) = h((i, i+1))$  and for all  $r' \in (\text{col}(g) \cup V_1(f) \cup E_2(f)) \setminus \{(i, i+1), (j-1, j)\}$ , let  $h'(r) = h(r)$ . Since  $h$  is bijective, we have that  $h' \neq h$ , and therefore  $\phi(f, g, h) \neq (f, g, h)$ . Since  $\{f(i), f(i+1)\} = \{f(j-1), f(j)\}$ , we have that  $y_{\{f(i), f(i+1)\}, h((i, i+1))} = y_{\{f(j-1), f(j)\}, h((j-1, j))}$  and  $y_{\{f(j-1), f(j)\}, h((j-1, j))} = y_{\{f'(i), f'(i+1)\}, h'((i, i+1))}$ , and therefore  $m(f, g, h) = m(\phi(f, g, h))$ . Finally, because  $f = f'$ , it still holds that there are no bad pairs of types 1 and 2 in  $(f', g', h')$ , and the sets of bad pairs

of type 3 of  $(f, g, h)$  and  $(f', g', h')$  are same. Since by swapping the values assigned to  $i$  and  $j$  by  $h'$ , we obtain  $h$ , and hence we have that  $\phi(\phi(f, g, h)) = (f, g, h)$ .

**Case 3(b):  $f(i) \dots f(j)$  is not a palindrome.** We define  $(f', g', h') = \phi(f, g, h)$  as follows. For any  $r \in [q]$  such that  $r < i$  or  $r > j$ , we let  $f'(r) = f(r)$ ,  $g'(r) = g(r)$ . For any  $r \in \text{col}(g) \cup V_1(f)$ ,  $r < i$  or  $r > j$ ,  $h'(r) = h(r)$ . (In this context, recall that the domain of  $h$  is a set of indices.) For all  $(i', i' + 1)$ ,  $i' < i$  or  $i' \geq j$  such that  $(i', i' + 1) \in E_2(f)$ , let  $h'((i', i' + 1)) = h((i', i' + 1))$ . Next, we are going to redirect the subwalk from  $i$  to  $j$  to be from  $j$  to  $i$ , adjusting  $f, g$  and  $h$  accordingly. Formally, for each index  $i \leq r \leq i + \lfloor \frac{i-j}{2} \rfloor$ , define  $s(r) = j - (r - i)$  and  $s(j - (r - i)) = r$ . Now, for each index  $i \leq r \leq j$ , let  $f'(r) = f(s(r))$ ,  $g'(r) = g(s(r))$  and, if  $r \in \text{col}(g) \cup V_1(f)$ ,  $h'(r) = h(s(r))$ , and if  $(r, r + 1) \in E_2(f)$  and  $r + 1 \leq j$   $h'((r, r + 1)) = h(s(r), s(r + 1))$ . Since  $f(i) \dots f(j)$  is not a palindrome, we have that  $f \neq f'$  and hence,  $\phi(f, g, h) \neq (f, g, h)$ . Notice that  $f$  and  $f'$  defines two walks in the graph  $G$ . Observe that because  $f(i) = f(j)$ , redirecting the subwalk between  $i$  and  $j$  does not change the edges which the walk contains (but only the order between them changes), and since upon changing the location of an occurrence of a vertex, we update  $f, g$  and  $h$  accordingly to get  $f', g'$  and  $h'$ , we have that  $m(f, g, h) = m(f', g', h') = m(\phi(f, g, h))$ . Finally,  $(f', g', h')$  does not have bad pairs of types 1 and 2, and it has the same bad pairs of type 3 as  $(f, g, h)$ . Also, since by redirecting the subwalk between  $i$  and  $j$  yet again we obtain the original walk, and so we conclude that  $\phi(\phi(f, g, h)) = (f, g, h)$ .

We also need to show that  $(f', g', h') \in B_1$ . Towards that it is enough to show that  $f'$  is a  $(\ell, \ell_1, \ell_2, u, t, k)$ -function for some  $u \in V(G)$ . Since  $f$  is such a function, all the properties except the property (iii) of  $(\ell, \ell_1, \ell_2, u, t, k)$ -function hold trivially. Now we show that in fact property (iii) also true for  $f'$ . That is for all  $i' \in [q - 2]$ ,  $f'(i') = f'(i' + 2) \in V_2$  implies that  $f'(i' + 1) \in V_2$ . Since  $f$  satisfies property (iii), it holds that for all  $i' \notin \{i - 1, i, j - 1, j\}$ , if  $f'(i') = f'(i' + 2) \in V_2$  implies that  $f'(i' + 1) \in V_2$ . Now consider the case of  $i - 1$  and  $f'(i - 1), f'(i), f'(i + 1)$ . Since  $f'(i) \in V_2$ , statement mentioned in property (iii) holds for  $i - 1$ . Now consider the case  $i$  and  $f'(i), f'(i + 1), f'(i + 2)$ . Notice that  $f'(i) = f(i)$  and  $f'(i + 2) = f(j - 2)$ . Since  $(i, j)$  is a smallest pair with  $f(i) = f(j) \in V_2$ , we have that  $f(i) \neq f(j - 2)$  and hence  $f'(i) \neq f'(i + 2)$ . Hence statement mentioned in property (iii) holds for  $i$ . Now consider the case of  $j - 1$  and  $f'(j - 1), f'(j), f'(j + 1)$ . Since  $f'(j) \in V_2$ , statement mentioned in property (iii) holds for  $j - 1$ . Now consider the case of  $j$  and  $f'(j), f'(j + 1), f'(j + 2)$ . Since  $f'(j) = f(j), f'(j + 1) = f(j + 1), f'(j + 2) = f(j + 2)$  and property (iii) holds for  $f$ , we have that property (iii) holds for  $j$ . Hence  $(f', g', h') \in B_1$ .

**Case 4:  $i, j \in \text{col}(g)$  and  $g(i) = g(j)$ .** Among all such pairs, that we call bad pairs of type 4, let  $(i, j)$  be the smallest one. We define  $(f', g', h') = \phi(f, g, h)$  as follows. First, let  $f' = f$  and  $g' = g$ . Now, let  $h'(i) = h(j)$  and  $h'(j) = h(i)$ , and for all  $r \in (\text{col}(g) \cup V_1(f) \cup E_2(f)) \setminus \{i, j\}$ , we let  $h'(r) = h(r)$ . Since  $h$  is bijective, we have that  $h' \neq h$ , and therefore  $\phi(f, g, h) \neq (f, g, h)$ . Moreover,  $z_{g(i), h(i)} = z_{g'(j), h'(j)}$  and  $z_{g(j), h(j)} = z_{g'(i), h'(i)}$ , and therefore  $m(f, g, h) = m(\phi(f, g, h))$ . Finally, because  $(f, g) = (f', g')$ , there are still no bad pairs of types 1, 2 and 3, and the sets of bad pairs of type 4 of  $(f, g, h)$  and  $(f', g', h')$  are same. Also, by swapping the values assigned to  $i$  and  $j$  in  $h'$ , we obtain  $h$ , and so we have that  $\phi(\phi(f, g, h)) = (f, g, h)$ . ◀

Now, we define a polynomial  $Q = \sum_{(f, g, h) \in B_1} m(f, g, h)$  which will be evaluated over the finite field  $\mathbb{F}_p$ , where  $p = 2^{\lceil \log(3(\ell + \ell_1 + \ell_2 + 2k)) \rceil}$ . Since  $\mathbb{F}_p$  has characteristic 2, by Observa-

tions 16 and 18, and Lemma 19, we get the following lemma (its proof is identical to the proof of Lemma 8).

► **Lemma 20.** *The input instance of  $K$ -PEL PATH is a YES-instance if and only if  $P \neq 0$ .*

**Evaluation** For  $I \subseteq [k + \ell_1 + \ell_2]$ , a  $(q, q_1, q_2, u, v, k', I)$ -labeling  $(f, g, h)$  is a  $(q, q_1, q_2, u, v, k')$ -labeling such that the image of  $h$  is a subset of  $I$ . Let  $S(q, q_1, q_2, u, v, k', I)$  denote the set of all  $(q, q_1, q_2, u, v, k', I)$ -labelings. Let  $P(q, q_1, q_2, u, v, k', I) = \sum_{(f,g,h) \in S(q,q_1,q_2,u,v,k',I)} m(f, g, h)$ .

By the inclusion-exclusion principle and because  $\mathbb{F}_p$ , the field over which we evaluate polynomials, has characteristic 2, we have the following observation.

► **Observation 21.**  $Q = \sum_{I \subseteq [\ell_1 + \ell_2 + k]} \sum_{u^* \in N(t)} P(\ell, \ell_1, \ell_2, u^*, t, k, I)$ .

We next show that a polynomial of the form  $P(\ell, \ell_1, \ell_2, u^*, t, k, I)$  can be evaluated at any point in time polynomial in  $N$  using dynamic programming, where  $N$  is an upper bound on number of variables.

► **Lemma 22** ( $\star$ ). *Let  $I \subseteq [k]$ ,  $u^* \in N(t)$ . Let  $x_1, \dots, x_N$  be the variables in  $P(\ell, \ell_1, \ell_2, u^*, t, k, I)$ . Let  $a_1, a_2, \dots, a_N \in \mathbb{F}_p$  denote the chosen values for the variables. Then, the polynomial  $P(\ell, \ell_1, \ell_2, u^*, t, k, I)$  can be evaluated at  $(a_1, a_2, \dots, a_N)$  in time  $N^{\mathcal{O}(1)}$  using space  $N^{\mathcal{O}(1)}$ .*

Combining Observation 21 with Lemma 22, and since there are  $2^{\ell_1 + \ell_2 + k}$  subsets  $I$  of  $[\ell_1 + \ell_2 + k]$ , we have the following result.

► **Lemma 23.** *Given an assignment of values from  $\mathbb{F}_p$  to the variables of  $P$ , the polynomial  $P$  can be evaluated in time  $2^{\ell_1 + \ell_2 + k} N^{\mathcal{O}(1)}$  using space polynomial in  $N$ .*

**The Algorithm.** By Lemmata 13, 20 and 23, we get the following lemma (its proof is identical to Theorem 14).

► **Lemma 24.** *There is a polynomial space randomized algorithm for  $K$ -PEL PATH running in time  $\mathcal{O}^*(2^{\ell_1 + \ell_2 + k})$  with one sided constant error probability.*

► **Lemma 25** ([3]). *Let  $W$  be a simple path of length  $\ell$  in a graph  $G$ . Then, for a partition  $(V_1, V_2)$  of  $V(G)$  chosen uniformly at random, let  $p(\ell, \ell_1, \ell_2)$  be the probability that  $|V(W) \cap V_1| = \ell_1$  and  $|E(W) \cap E(V_2)| = \ell_2$ . Then for any  $\ell$ , we can find  $\ell'_1, \ell'_2 \in \mathbb{N}$  such that  $\ell'_1 + \ell'_2 \leq \ell$  and  $\frac{2^{\ell'_1 + \ell'_2}}{p(\ell, \ell'_1, \ell'_2)} \leq 1.6569^\ell |V(G)|^c$  for some constant  $c$ .*

We use Lemma 25 to prove the main theorem of the section.

► **Theorem 26.** *There is a polynomial space randomized algorithm for EXACT LIST  $K$ -CYCLE running in time  $\mathcal{O}^*(2^k 1.6569^{\ell - k})$  with one sided constant error probability.*

**Proof.** Let  $(G, K, L, k^*, k, \ell)$  be the input instance and  $n = |V(G)|$ . We describe an algorithm  $\mathcal{A}$  for EXACT LIST  $K$ -CYCLE. Let  $\ell', \ell'_2$  be the integers guaranteed by the Lemma 25 and let  $p(\ell, \ell'_1, \ell'_2)$  be the probability mentioned in Lemma 25. Now for each  $\ell_1 \leq \ell'_1$  and  $\ell_2 \leq \ell'_2$ , we randomly choose a partition  $(V_1, V_2)$  of  $V(G) \setminus K$  and run algorithm  $\mathcal{B}$  mentioned in Lemma 24 for  $K$ -PEL PATH. Algorithm  $\mathcal{A}$  repeats algorithm  $\mathcal{B}$   $\frac{1}{p(\ell, \ell'_1, \ell'_2)}$  many times. If at least once algorithm  $\mathcal{B}$  outputs YES, then  $\mathcal{A}$  outputs YES and otherwise outputs No.

Clearly if  $(G, K, L, k^*, k, \ell)$  is a NO instance, then our algorithm will output No. Now suppose  $(G, K, L, k^*, k, \ell)$  is a YES instance. Then there is a simple path  $W$  of length  $\ell$

with required property. We know that for a partition  $(V'_1, V'_2)$  of  $V(G)$  chosen uniformly at random, the probability that  $|V(W) \cap V_1| = \ell'_1$  and  $|E(W) \cap E(V_2)| = \ell'_2$  is  $p(\ell, \ell'_1, \ell'_2)$ . This implies that there exist  $\ell_1 \leq \ell'_1$  and  $\ell_2 \leq \ell'_2$  such that for a partition  $(V_1, V_2)$  of  $V(G) \setminus K$  chosen uniformly at random, the probability that  $|V(W) \cap V_1| = \ell_1$  and  $|E(W) \cap E(V_2)| = \ell_2$  is  $p(\ell, \ell'_1, \ell'_2)$ . For that choice of  $\ell_1$  and  $\ell_2$  the probability that algorithm  $\mathcal{B}$  outputs YES is  $p(\ell, \ell'_1, \ell'_2)$ . Since algorithm  $\mathcal{B}$  runs  $\frac{1}{p(\ell, \ell'_1, \ell'_2)}$  many times with parameters  $\ell_1$  and  $\ell_2$ , algorithm  $\mathcal{A}$  outputs YES with constant probability.

The running time of algorithm  $\mathcal{A}$  is upper bounded by  $\frac{2^{\ell'_1 + \ell'_2 + k} n^{c'}}{p(\ell, \ell'_1, \ell'_2)}$  for some constant  $c'$ . By Lemma 25, we have that  $\frac{2^{\ell'_1 + \ell'_2}}{p(\ell, \ell'_1, \ell'_2)} \leq 1.6569^\ell n^c$ , where  $c$  is some constant. This implies that the running time of algorithm  $\mathcal{A}$  is upper bounded by  $\mathcal{O}^*(2^k 1.6569^{\ell-k})$ . Since algorithm  $\mathcal{B}$  uses only polynomial space,  $\mathcal{A}$  is a polynomial space algorithm. This completes the proof of the theorem.  $\blacktriangleleft$

---

## References

- 1 N. Alon, R. Yuster, and U. Zwick. Color coding. *J. ACM*, 42(4):844–856, 1995.
- 2 A. Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014.
- 3 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Narrow sieves for parameterized paths and packings. *CoRR abs/1007.1161*, 2010.
- 4 A. Björklund, T. Husfeldt, and N. Taslaman. Shortest cycle through specified elements. In *SODA*, pages 1747–1753, 2012.
- 5 A. Björklund, V. Kamat, L. Kowalik, and M. Zehavi. Spotting trees with few leaves. In *ICALP*, pages 243–255, 2015.
- 6 A. Björklund, P. Kaski, and L. Kowalik. Constrained multilinear detection and generalized graph motifs. *Algorithmica*, 74(2):947–967, 2016.
- 7 A. Björklund, P. Kaski, J. Lauri, and L. Kowalik. Engineering motif search for large graphs. In *ALENEX*, pages 104–118, 2016.
- 8 H. L. Bodlaender. On linear time minor tests with depth-first search. *J. Algorithms*, 14(1):1–23, 1993.
- 9 J. Chen, J. Kneis, S. Lu, D. Mölle, S. Richter, P. Rossmanith, S. H. Sze, and F. Zhang. Randomized divide-and-conquer: Improved path, matching, and packing algorithms. *SIAM J. on Computing*, 38(6):2526–2547, 2009.
- 10 M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, P. Paturi, S. Saurabh, and M. Wahlström. On problems as hard as CNF-SAT. In *CCC*, pages 74–84, 2012.
- 11 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, 2015.
- 12 R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inform. Process Lett.*, 7(4):193–195, 1978.
- 13 P. Deshpande, R. Barzilay, and D. R. Karger. Randomized decoding for selection-and-ordering problems. In *HLT-NAACL*, pages 444–451, 2007.
- 14 R. Downey and M. Fellows. *Fundamentals of parameterized complexity*. Springer, 2013.
- 15 H. Fleischner and G. H. Woeginger. Detecting cycles through three fixed vertices in a graph. *Inform. Process Lett.*, 41:29–33, 1992.
- 16 F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh. Representative sets of product families. In *ESA*, pages 443–454, 2014.
- 17 F. V. Fomin, D. Lokshtanov, and S. Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *SODA*, pages 142–151, 2014.



- 18 S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980.
- 19 F. Hüffner, S. Wernicke, and T. Zichner. Algorithm engineering for color-coding with applications to signaling pathway detection. *Algorithmica*, 52(2):114–132, 2008.
- 20 K. Kawarabayashi. An improved algorithm for finding cycles through elements. In *IPCO*, pages 374–384, 2008.
- 21 K. Kawarabayashi, Z. Li, and B. A. Reed. Recognizing a totally odd  $k_4$ -subdivision, parity 2-disjoint rooted paths and a parity cycle through specified elements. In *SODA*, pages 318–328, 2010.
- 22 I. Koutis. Faster algebraic algorithms for path and packing problems. In *ICALP*, pages 575–586, 2008.
- 23 L. Kowalik and J. Lauri. On finding rainbow and colorful paths. *Theor. Comput. Sci.*, 2016.
- 24 A. S. LaPaugh and R. L. Rivest. The subgraph homomorphism problem. *J. Comput. Sys. Sci.*, 20:133–149, 1980.
- 25 B. Monien. How to find long paths efficiently. *Annals Disc. Math.*, 25:239–254, 1985.
- 26 R. Y. Pinter, H. Shachnai, and M. Zehavi. Improved parameterized algorithms for network query problems. In *IPEC*, pages 294–306, 2014.
- 27 R. Y. Pinter and M. Zehavi. Algorithms for topology-free and alignment network queries. *JDA*, 27:29–53, 2014.
- 28 N. Robertson and P. D. Seymour. Graph minors XIII: The disjoint paths problem. *J. Combin. Theory Ser. B*, 63:65–110, 1995.
- 29 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. Assoc. Comput. Mach.*, 27(4):701–717, 1980.
- 30 H. Shachnai and M. Zehavi. Representative families: A unified tradeoff-based approach. In *ESA*, pages 786–797, 2014.
- 31 M. Wahlström. Abusing the Tutte matrix: an algebraic instance compression for the  $k$ -set-cycle problem. In *STACS*, pages 341–352, 2013.
- 32 R. Williams. Finding paths of length  $k$  in  $O^*(2^k)$  time. *Inf. Process. Lett.*, 109(6):315–318, 2009.
- 33 Kobayashi Y. and K. Kawarabayashi. Algorithms for finding an induced cycle in planar graphs and bounded genus graphs. In *SODA*, pages 1146–1155, 2009.
- 34 M. Zehavi. Parameterized algorithms for the module motif problem. In *MFCS*, pages 825–836, 2013.
- 35 M. Zehavi. Mixing color coding-related techniques. In *ESA*, pages 1037–1049, 2015.
- 36 R. Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, pages 216–226, 1979.





# Lossy Kernels for Graph Contraction Problems

R. Krithika<sup>1</sup>, Pranabendu Misra<sup>2</sup>, Ashutosh Rai<sup>3</sup>, and  
Prafullkumar Tale<sup>4</sup>

1 The Institute of Mathematical Sciences, HBNI, Chennai, India  
rkkrithika@imsc.res.in

2 The Institute of Mathematical Sciences, HBNI, Chennai, India  
pranabendu@imsc.res.in

3 The Institute of Mathematical Sciences, HBNI, Chennai, India  
ashutosh@imsc.res.in

4 The Institute of Mathematical Sciences, HBNI, Chennai, India  
pptale@imsc.res.in

---

## Abstract

We study some well-known graph contraction problems in the recently introduced framework of *lossy kernelization*. In classical kernelization, given an instance  $(I, k)$  of a parameterized problem, we are interested in obtaining (in polynomial time) an equivalent instance  $(I', k')$  of the same problem whose size is bounded by a function in  $k$ . This notion however has a major limitation. Given an approximate solution to the instance  $(I', k')$ , we can say nothing about the original instance  $(I, k)$ . To handle this issue, among others, the framework of lossy kernelization was introduced. In this framework, for a constant  $\alpha$ , given an instance  $(I, k)$  we obtain an instance  $(I', k')$  of the same problem such that, for every  $c > 1$ , any  $c$ -approximate solution to  $(I', k')$  can be turned into a  $(c\alpha)$ -approximate solution to the original instance  $(I, k)$  in polynomial time. Naturally, we are interested in a polynomial time algorithm for this task, and further require that  $|I'| + k' = k^{\mathcal{O}(1)}$ . Akin to the notion of polynomial time approximation schemes in approximation algorithms, a parameterized problem is said to admit a polynomial size approximate kernelization scheme (PSAKS) if it admits a polynomial size  $\alpha$ -approximate kernel for every approximation parameter  $\alpha > 1$ . In this work, we design PSAKSs for TREE CONTRACTION, STAR CONTRACTION, OUT-TREE CONTRACTION and CACTUS CONTRACTION problems. These problems do not admit polynomial kernels, and we show that each of them admit a PSAKS with running time  $k^{f(\alpha)}|I|^{\mathcal{O}(1)}$  that returns an instance of size  $k^{g(\alpha)}$  where  $f(\alpha)$  and  $g(\alpha)$  are constants depending on  $\alpha$ .

**1998 ACM Subject Classification** F.2 Analysis of Algorithms and Problem Complexity

**Keywords and phrases** parameterized complexity, lossy kernelization, graph theory, edge contraction problems

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.23

## 1 Introduction

Many computational problems arising from real-world problems are NP-hard, and we do not expect any efficient algorithms for solving them optimally. Preprocessing heuristics, or data reduction rules, are widely applied to reduce large instances of these problems to a smaller size before attempting to solve them. Such algorithms are often extremely effective, and provide a significant boost to the subsequent step of computing a solution to the instance. Kernelization, under the aegis of Parameterized Complexity, has been developed as a mathematical framework to study these algorithms and quantify their efficacy.



© R. Krithika, Pranabendu Misra, Ashutosh Rai, and Prafullkumar Tale;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 23; pp. 23:1–23:14



Leibniz International Proceedings in Informatics



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

In *Parameterized Complexity*, we consider instances  $(I, k)$  of a parameterized problem  $\Pi \subseteq \Sigma^* \times \mathbb{N}$ , where  $\Sigma$  is a finite alphabet. Typically,  $I$  is an instance of some computational problem, and  $k$  denotes the *parameter* which reflects some structural property of the instance. A common parameter is a bound on the size of an optimum solution to the problem instance. A data reduction algorithm, formally called a *Kernelization algorithm*, runs in polynomial time and reduces a given instance  $(I, k)$  of the problem to an *equivalent* instance  $(I', k')$  such that  $|I'| + k' = k^{\mathcal{O}(1)}$ . The instance  $(I', k')$  is called a *polynomial kernel*, and we say that the problem  $\Pi$  admits a polynomial kernelization (also called classical kernelization). Designing kernelization algorithms for various computational problems, and investigating the associated lower bounds, is an active area of research in Computer Science. We refer the reader to [6, 9, 10] for an introduction to Parameterized Complexity and Kernelization.

The notion of polynomial kernels turns out to be a bit stringent, and it has been discovered that many problems do not admit a polynomial kernel under well-known complexity theoretic conjectures. On the other hand this notion turns out to be too lax as the instances  $(I, k)$  and  $(I', k')$  are not as tightly-coupled as one would like. For example, it is not possible to translate an approximate solution to the instance  $(I', k')$ , into an approximate solution to the original instance  $(I, k)$ . Indeed, given anything but an optimal solution (or a solution of size  $k'$ ) to  $(I', k')$ , it is impossible to conclude anything about the original instance  $(I, k)$ . These issues, among others, have led to the development of a framework for “approximation preserving kernelization” or *Lossy Kernelization*. Informally, an  $\alpha$ -approximate kernelization algorithm ensures that given any  $c$ -approximate solution to the kernel  $(I', k')$ , it can be converted into a  $c \cdot \alpha$ -approximate solution to the original instance  $(I, k)$  in polynomial time. This notion was formally introduced, very recently, in [18] which shows that there are many problems without classical polynomial kernels that admit lossy polynomial kernels. Furthermore, it is likely that this notion will be very useful in practice. Many state of the art approximation algorithms are extremely sophisticated and it is infeasible to apply them to large problem instances. It is far more practical to reduce a large instance to a small kernel, then obtain a good approximate solution to this kernel, and finally transform it into an approximate solution to the original instance. In other words, lossy kernelization provides a mathematical framework for designing and analyzing preprocessing heuristics for approximation algorithms.

Let us state these notions formally. We first define a *parameterized optimization (maximization / minimization) problem*, which is the parameterized analogue of an optimization problem in the theory of approximation algorithms. A *parameterized minimization problem* is a computable function  $\Pi : \Sigma^* \times \mathbb{N} \times \Sigma^* \mapsto \mathbb{R} \cup \{\pm\infty\}$ . The instances of  $\Pi$  are pairs  $(I, k) \in \Sigma^* \times \mathbb{N}$  and a solution to  $(I, k)$  is simply a string  $S \in \Sigma^*$  such that  $|S| \leq |I| + k$ . The *value* of a solution  $S$  is  $\Pi(I, k, S)$ . The *optimum value* of  $(I, k)$  is  $\text{OPT}_\Pi(I, k) = \min_{S \in \Sigma^*, |S| \leq |I| + k} \Pi(I, k, S)$ , and an *optimum solution* for  $(I, k)$  is a solution  $S$  such that  $\Pi(I, k, S) = \text{OPT}_\Pi(I, k)$ . A *parameterized maximization problem* is defined in a similar way. We will omit the subscript  $\Pi$  in the notation for optimum value if the problem under consideration is clear from context. Next we come to the notion of an  $\alpha$ -approximate polynomial-time preprocessing algorithm for a parameterized optimization problem  $\Pi$ . It is defined as a pair of polynomial-time algorithms, called the *reduction algorithm* and the *solution lifting algorithm*, that satisfy the following properties.

- Given an instance  $(I, k)$  of  $\Pi$ , the reduction algorithm computes an instance  $(I', k')$  of  $\Pi$ .
- Given the instances  $(I, k)$  and  $(I', k')$  of  $\Pi$ , and a solution  $S'$  to  $(I', k')$ , the solution lifting algorithm computes a solution  $S$  to  $(I, k)$  such that  $\frac{\Pi(I, k, S)}{\text{OPT}_\Pi(I, k)} \leq \alpha \cdot \frac{\Pi(I', k', S')}{\text{OPT}_\Pi(I', k')}$ .

A *reduction rule* is the execution of the reduction algorithm on an instance, and it is applicable on an instance if the output instance is different from the input instance. An

$\alpha$ -approximate kernelization (or  $\alpha$ -approximate kernel) for  $\Pi$  is an  $\alpha$ -approximate polynomial-time preprocessing algorithm such that the size of the output instance is upper bounded by a computable function  $g : \mathbb{N} \times \mathbb{N}$  of  $k$ . In classical kernelization, often we apply reduction rules several times to reduce the given instance. This however breaks down in lossy kernelization, since each application of a reduction rule increases the “gap” between the approximation quality of the solution to the kernel, and the approximation quality of solution to the original instance that is computed by the solution lifting algorithm. To remedy this shortcoming, we require the notion of  $\alpha$ -strict kernelization and  $\alpha$ -safe reduction rules. An  $\alpha$ -approximate kernelization is said to be *strict* if  $\frac{\Pi(I,k,s)}{\text{OPT}(I,k)} \leq \max\{\frac{\Pi(I',k',s')}{\text{OPT}(I',k')}, \alpha\}$ . A reduction rule is said to be  $\alpha$ -safe for  $\Pi$  if there is a solution lifting algorithm, such that the rule together with this algorithm constitute a strict  $\alpha$ -approximate polynomial-time preprocessing algorithm for  $\Pi$ . A reduction rule is *safe* if it is 1-safe, and note this this is more strict than the usual definition of safeness in classical kernelization. A *polynomial-size approximate kernelization scheme (PSAKS)* for  $\Pi$  is a family of  $\alpha$ -approximate polynomial kernelization algorithms for each  $\alpha > 1$ . Note that, the size of an output instance of a PSAKS, when run on  $(I, k)$  with approximation parameter  $\alpha$ , must be upper bounded by  $f(\alpha)k^{g(\alpha)}$  for some functions  $f$  and  $g$  independent of  $|I|$  and  $k$ . And finally, let us discuss the importance of the parameter  $k$  in this framework. In a classical kernelization, given an instance  $(I, k)$  the algorithm either returns another instance (i.e. a kernel) or decides that given instance has no solution of value at most  $k$  (i.e. a NO instance). In lossy kernelization the output is always an instance of the optimization problem, and we must ensure that our kernelization algorithm must be safe on all instances. This may seem like a difficult goal, but recall that we are only interested in solutions of value at most  $k$ , and therefore we may define our parameterized minimization problem to reflect this fact.

$$\Pi(I, k, S) = \begin{cases} \infty & \text{if } S \text{ is not a solution} \\ \min\{|S|, k + 1\} & \text{otherwise} \end{cases}$$

This definition allows us to design the reduction rules without regard to the solutions of value more than  $k$ . In particular, if the solution lifting algorithm is given a solution of value  $k + 1$  or more, it simply returns an a trivial feasible solution to the instance, and this is safe as per the above definitions. We encourage the reader to see [18] for a more comprehensive discussion of these ideas and definitions.

In [18], the authors exhibit lossy kernels for several problems which do not admit a classical kernelization, such as CONNECTED VERTEX COVER, DISJOINT CYCLE PACKING and DISJOINT FACTORS. They also develop a lower bound framework for lossy kernels, by extending the lower bound framework of classical kernelization. They then show that LONGEST PATH does not admit a lossy kernel of polynomial size unless  $\text{NP} \subseteq \text{coNP}/\text{poly}$ . In this paper, we investigate several other problems in the framework of lossy kernelization. In particular, we design lossy polynomial kernels for several *graph contraction* problems which do not admit classical polynomial kernels under well known complexity theoretic conjectures. These problems are defined as follows. For a graph class  $\mathcal{G}$ , the  $\mathcal{G}$ -CONTRACTION problem is to determine if an input graph  $G$  can be contracted to some graph  $H \in \mathcal{G}$  using at most  $k$  edge contractions. These problems are well studied and  $\mathcal{G}$ -CONTRACTION has been proven to be NP-complete for several classes  $\mathcal{G}$  [1, 4, 20, 21]. They have also received a lot of attention in Parameterized Complexity [2, 5, 12, 13, 14, 15, 16, 17, 19]. In this work, we give lossy polynomial kernels for the following problems. In the following  $G/F$  denotes the graph obtained from  $G$  by contracting the edges in  $F$ .

TREE CONTRACTION

Parameter:  $k$ **Input:** A graph  $G$  and an integer  $k$ **Question:** Does there exist  $F \subseteq E(G)$  of size at most  $k$  such that  $G/F$  is a tree?

STAR CONTRACTION

Parameter:  $k$ **Input:** A graph  $G$  and an integer  $k$ **Question:** Does there exist  $F \subseteq E(G)$  of size at most  $k$  such that  $G/F$  is a star?

OUT-TREE CONTRACTION

Parameter:  $k$ **Input:** A digraph  $D$  and an integer  $k$ **Question:** Does there exist  $F \subseteq A(D)$  of size at most  $k$  such that  $D/A$  is an out-tree?

CACTUS CONTRACTION

Parameter:  $k$ **Input:** A graph  $G$  and an integer  $k$ **Question:** Does there exist  $F \subseteq E(G)$  of size at most  $k$  such that  $G/F$  is a cactus?

It can be shown that these problems do not admit polynomial kernels, via a parameter preserving reduction from the RED BLUE DOMINATING SET problem. Let us define these terms formally. A *polynomial-time parameter preserving reduction* from a parameterized problem  $\Pi_1$  to a parameterized problem  $\Pi_2$  is a polynomial-time function that maps an instance  $(I_1, k_1)$  of  $\Pi_1$  to an instance  $(I_2, k_2)$  of  $\Pi_2$  such that  $k_2 = k_1^{\mathcal{O}(1)}$ , and  $(I_1, k_1)$  is a YES instance of  $\Pi_1$  if and only if  $(I_2, k_2)$  is a YES instance of  $\Pi_2$ . It is known that if  $\Pi_1$  does not admit a polynomial kernel, then neither does  $\Pi_2$  [3]. Next, let us define the RED BLUE DOMINATING SET problem. The input is a bipartite graph  $G$  with bipartition  $(A, B)$  and an integer  $t$ , this problem asks if  $B$  has a subset of at most  $t$  vertices that dominates  $A$ . This problem is NP-complete [11] and it does not have a polynomial kernel when parameterized by  $|A|$  [8]. It was shown that TREE CONTRACTION and STAR CONTRACTION do not admit a polynomial kernel, by a polynomial parameter preserving reduction from this problem [16]. We build upon the reductions in [16] to show that the two remaining problems also do not admit a polynomial kernel. The following theorem is the main result of this paper.

► **Theorem 1.1.** *Given a graph (digraph)  $G$  on  $n$  vertices, an integer  $k$  and an approximation parameter  $\alpha > 1$ , there is an algorithm that runs in  $k^{f(\alpha)}n^{\mathcal{O}(1)}$  time and outputs a graph (digraph)  $G'$  on  $k^{g(\alpha)}$  vertices and an integer  $k'$  such that for every  $c > 1$ , a  $c$ -approximate (tree/star/cactus/out-tree contraction) solution for  $(G', k')$  can be turned into a  $(c\alpha)$ -approximate (tree/star/cactus/out-tree contraction) solution for  $(G, k)$  in  $n^{\mathcal{O}(1)}$ . Here  $f(\alpha)$  and  $g(\alpha)$  are constants depending on  $\alpha$ .*

## 2 Preliminaries

An undirected graph is a pair consisting of a set  $V$  of vertices and a set  $E$  of edges where  $E \subseteq V \times V$ . An edge is specified as an unordered pair of vertices. For a graph  $G$ ,  $V(G)$  and  $E(G)$  denote the set of vertices and edges respectively. Two vertices  $u, v$  are said to be *adjacent* if there is an edge  $uv$  in the graph. The neighbourhood of a vertex  $v$ , denoted by  $N_G(v)$ , is the set of vertices adjacent to  $v$  and its degree  $d_G(v)$  is  $|N_G(v)|$ . The subscript in the notation for neighbourhood and degree is omitted if the graph under consideration is clear. For a set of edges  $F$ ,  $V(F)$  denotes the set of endpoints of edges in  $F$ . For a set  $S \subseteq V(G)$ ,  $G - S$  denotes the graph obtained by deleting  $S$  from  $G$  and  $G[S]$  denotes the subgraph of  $G$  induced on set  $S$ . For graph theoretic terms and notation which are not explicitly defined here, we refer the reader to the book by Diestel [7].

Two non-adjacent vertices  $u$  and  $v$  are called as *false twins* of each other if  $N(u) = N(v)$ . A *path*  $P = (v_1, \dots, v_l)$  is a sequence of distinct vertices where every consecutive pair of vertices is adjacent. The vertices of  $P$  is the set  $\{v_1, \dots, v_l\}$  and is denoted by  $V(P)$ . The *length* of a path is  $|V(P)| - 1$ . A *cycle* is a sequence  $(v_1, \dots, v_l, v_1)$  of vertices such that  $(v_1, \dots, v_l)$  is a path and  $v_l v_1$  is an edge. A *leaf* is a vertex of degree 1. A graph is called *connected* if there is a path between any pair of its vertices and it is called *disconnected* otherwise. A *cut vertex* of a connected graph  $G$  is a vertex  $v$  such that  $G - \{v\}$  is disconnected. A graph that has no cut vertex is called *2-connected*. A *component* of a disconnected graph is a maximal connected subgraph. A set  $S \subseteq V(G)$  is called a *vertex cover* if for every edge  $uv$ , either  $u \in S$  or  $v \in S$ . Further,  $S$  is called a *connected vertex cover* if  $G[S]$  is connected. A set  $I \subseteq V(G)$  of pairwise non-adjacent vertices is called as an *independent set*. A set  $S$  of vertices is said to *dominate* another set  $S'$  of vertices if for every vertex  $v$  in  $S'$ ,  $N(v) \cap S \neq \emptyset$ . A *tree* is a connected acyclic graph. A *star* is a tree in which there is a path of length at most 2 between any 2 vertices. A graph is called a *cactus* if every edge is a part of at most one cycle.

The *contraction* operation of an edge  $e = uv$  in  $G$  results in the deletion of  $u$  and  $v$  and the addition of a new vertex  $w$  adjacent to vertices that were adjacent to either  $u$  or  $v$ . Any parallel edges added in the process are deleted so that the graph remains simple. The resulting graph is denoted by  $G/e$ . Formally,  $V(G/e) = V(G) \cup \{w\} \setminus \{u, v\}$  and  $E(G/e) = \{xy \mid x, y \in V(G) \setminus \{u, v\}, xy \in E(G)\} \cup \{wx \mid x \in N_G(u) \cup N_G(v)\}$ . For a set of edges  $F \subseteq E(G)$ ,  $G/F$  denotes the graph obtained from  $G$  by contracting the edges in  $F$  in an arbitrary order. It is easy to see that  $G/F$  is oblivious to the contraction sequence. A graph  $G$  is *contractible* to a graph  $T$ , if  $T$  can be obtained from  $G$  by a sequence of edge contractions. For graphs  $G$  and  $T$  with  $V(T) = \{t_1, \dots, t_l\}$ ,  $G$  is said to have a  *$T$ -witness structure*  $\mathcal{W}$  if  $\mathcal{W}$  is a partition of  $V(G)$  into  $l$  sets and there is a bijection  $W : V(T) \mapsto \mathcal{W}$  such that the following properties hold.

- For each  $t_i \in V(T)$ ,  $G[W(t_i)]$  is connected.
- For a pair  $t_i, t_j \in V(T)$ ,  $t_i t_j \in E(T)$  if and only if there is an edge between a vertex in  $W(t_i)$  and a vertex in  $W(t_j)$  in  $G$ .

The sets  $W(t_1), \dots, W(t_l)$  in  $\mathcal{W}$  are called *witness sets*. It is easy to observe the following (also see [16]).

► **Observation 1.**  $G$  is contractible to  $T$  if and only if  $G$  has a  $T$ -witness structure.

We associate a  $T$ -witness structure  $\mathcal{W}$  of  $G$  with a set  $F \subseteq E(G)$  whose contraction in  $G$  results in  $T$ , by defining  $F$  to be the union of the set of edges of a spanning tree of  $G[W]$ , for each  $W \in \mathcal{W}$ . Note that there is a unique  $T$ -witness structure of  $G$  corresponding to a set  $F$ .

► **Observation 2.**  $|F| = \sum_{W \in \mathcal{W}} (|W| - 1)$ .

We say that,  $G$  is said to be  $|F|$ -contractible to  $T$  and it is easy to verify the following. For every  $W \in \mathcal{W}$ ,  $|W| \leq |F| + 1$ , and further,  $|\{W \in \mathcal{W} \mid |W| > 1\}| \leq |F|$ . Finally, we have the following observation on the neighbors of vertices in  $W(t)$ , when  $t$  is a leaf in  $T$ .

► **Observation 3.** Let  $t$  be a leaf in  $T$  and  $t'$  be its unique neighbour. Then,  $\bigcup_{v \in W(t)} N_G(v) \subseteq W(t') \cup W(t)$ .

**Proof.** Consider a leaf  $t$  in  $T$ . Assume on the contrary that there exists  $t'$  and  $t''$  (distinct from  $t$ ) such that  $N(u) \cap W(t') \neq \emptyset$  and  $N(v) \cap W(t'') \neq \emptyset$  for some  $u$  and  $v$  (not necessarily distinct) in  $W(t)$ . Then,  $t$  has degree at least 2 contradicting the fact that it is a leaf. ◀

We denote the set of integers from 1 to  $n$  by  $[n]$ . We also use the bound,  $\frac{x+p}{y+q} \leq \max\{\frac{x}{y}, \frac{p}{q}\}$  for any positive real numbers  $x, y, p, q$ , to prove that the reduction rules we define are strict  $\alpha$ -approximate for some real number  $\alpha$ . In this paper, use  $\text{TC}(\cdot)$ ,  $\text{OTC}(\cdot)$  and  $\text{CC}(\cdot)$  to denote the parameterized minimization version of TREE CONTRACTION, OUT-TREE CONTRACTION and CACTUS CONTRACTION, respectively. Recall that these functions assign a value of  $k+1$  to any solution of value  $k+1$  or more.

### 3 Tree Contraction

We begin with the TREE CONTRACTION problem, which admits a  $4^k n^{\mathcal{O}(1)}$  algorithm (where  $n$  is the number of vertices of the input graph) by using a FPT algorithm for CONNECTED VERTEX COVER as a subroutine, and further it does not admit a polynomial kernel unless  $\text{NP} \subseteq \text{coNP}/\text{poly}$  [16]. This lower-bound also holds for STAR CONTRACTION. Before we proceed to describe a PSAKS for these problems, we mention the following simplifying assumption known from [16] which states that, the tree witness structure of a graph can be constructed from the tree witness structures of its 2-connected components.

► **Lemma 3.1** ([16]). *A connected graph is  $k$ -contractible to a tree if and only if each of its 2-connected components is contractible to a tree using at most  $k$  edge contractions in total.*

Observe that there can be at most  $k$  non-trivial 2-connected components in the input graph, and we can consider each such component separately. The output of our kernelization algorithm will be a disjoint union of the kernels for each 2-connected component. So from now onwards we assume that the input graph is 2-connected. Next, we make some observations on the tree witness structure of a graph.

► **Lemma 3.2.** *Let  $F$  be a minimal set of edges of a 2-connected graph  $G$  such that  $G/F$  is a tree  $T$  with  $V(T) = \{t_1, t_2, \dots, t_l\}$  and  $l \geq 3$ . Let  $\mathcal{W}$  denote the corresponding  $T$ -witness structure of  $G$ . Then there exists a set  $F'$  of at most  $|F|$  edges of  $G$  such that,  $G/F'$  is a tree  $T'$  and the corresponding  $T'$ -witness structure  $\mathcal{W}'$  of  $G$  satisfies the following property:  $W'(t') \in \mathcal{W}'$  is a singleton set, if and only if  $t'$  is a leaf in  $T'$ .*

**Proof.** First, we show that every vertex  $t \in V(T)$  such that  $|W(t)| = 1$  is a leaf in  $T$ . Suppose there is a non-leaf  $t$  in  $T$  such that  $W(t) = \{u\}$  for some  $u \in V(G)$ . Then,  $T - \{t\}$  has at least two non-empty subtrees, say  $T_1$  and  $T_2$ . Consider  $U_1 = \bigcup_{t \in V(T_1)} W(t)$  and  $U_2 = \bigcup_{t \in V(T_2)} W(t)$ . As  $\mathcal{W}$  is the corresponding  $T$ -witness structure of  $G$ , it follows that there is no edge between a vertex in  $U_1$  and a vertex in  $U_2$  in  $G - \{u\}$ . This contradicts the fact that  $G$  is 2-connected.

Now, consider a leaf  $t_i$  in  $T$  such that  $|W(t_i)| > 1$ . Let  $t_j$  be the unique neighbour of  $t_i$ , and note that  $t_j$  is not a leaf in  $T$ . As  $t_i t_j \in E(T)$ , there exists an edge in  $G$  between a vertex in  $W(t_i)$  and a vertex in  $W(t_j)$ . Therefore,  $G[W(t_i) \cup W(t_j)]$  is connected. We claim that  $G[W(t_i) \cup W(t_j)]$  has a spanning tree which has a leaf from  $W(t_i)$ . Observe that as  $|W(t_i)| > 1$ , any spanning tree of  $G[W(t_i)]$  has at least 2 leaves. If there is a spanning tree of  $G[W(t_i)]$  that has a leaf  $u$  which is not adjacent to any vertex in  $W(t_j)$ , then  $G[(W(t_i) \cup W(t_j)) \setminus \{u\}]$  is connected too and  $u$  is the required vertex. Otherwise, every leaf in every spanning tree of  $G[W(t_i)]$  is adjacent to some vertex in  $W(t_j)$  and hence  $G[(W(t_i) \cup W(t_j)) \setminus \{u\}]$  is connected for each vertex  $u \in W(t_i)$ . Therefore, as claimed,  $G[W(t_i) \cup W(t_j)]$  has a spanning tree which has a leaf  $v$  from  $W(t_i)$ . Consider the partition  $\mathcal{W}' = (\mathcal{W} \cup \{W_v, W_{ij}\}) \setminus \{W(t_i), W(t_j)\}$  of  $G$  where  $W_v = \{v\}$  and  $W_{ij} = (W(t_j) \cup W(t_i)) \setminus \{v\}$ . Then, as  $N(v) \subseteq W(t_i) \cup W(t_j)$  by Observation 3, it follows that  $\mathcal{W}'$  is a  $T'$ -witness structure of  $G$ , where  $T'$  is a tree. Further,



$T'$  is the tree obtained from  $T$  by adding a new vertex  $t_{ij}$  adjacent to  $N(t_j)$  and a new vertex  $t_v$  adjacent to  $t_{ij}$  and then deleting  $t_i, t_j$ . This leads to a set  $F'$  of at most  $|F|$  edges of  $G$  such that  $T' = G/F'$  is a tree. Further note that,  $T$  and  $T'$  has the same number of vertices. Now recall that  $T$  has at least 3 vertices and hence it has at least one non-leaf vertex. Further, every leaf vertex is adjacent to some non-leaf vertex in  $T$  (and  $T'$ ). Therefore we repeat the above process for every non-singleton leaf, which proves the lemma. ◀

Let us remark that, it is safe to assume that  $T$  has at least 3 vertices. Otherwise, we can bound the number of vertices in  $G$  by  $|F| + 2$ , and since we are concerned with solutions of value  $k$  or smaller, this gives us a trivial kernel. Indeed, the main challenge lies in bounding the set of singleton witness sets. Subsequently, we assume that all tree witness structures have this property. Lemma 3.2 immediately leads to the following equivalence of STAR CONTRACTION and CONNECTED VERTEX COVER.

► **Lemma 3.3** ( $\star^1$ ).  *$G$  has a set  $F \subseteq E(G)$  such that  $G/F$  is a star if and only if  $G$  has a connected vertex cover of size  $|F| + 1$ .*

As CONNECTED VERTEX COVER has a PSAKS [18], we have the following result.

► **Theorem 3.4.** STAR CONTRACTION parameterized by the solution size admits a PSAKS.

Lemma 3.2 also leads to the following relationship between TREE CONTRACTION and CONNECTED VERTEX COVER.

► **Lemma 3.5.** *If  $G$  is  $k$ -contractible to a tree, then  $G$  has a connected vertex cover of size at most  $2k$ .*

**Proof.** As  $G$  is  $k$ -contractible to a tree, there exists a (minimal) set of edges  $F$  such that  $|F| \leq k$  and  $T = G/F$  is a tree. Let  $\mathcal{W}$  be the corresponding  $T$ -witness structure of  $G$  and  $\mathcal{W}'$  denote the set of non-singleton sets in  $\mathcal{W}$ . Let  $X$  denote the set of vertices of  $G$  which are in a set in  $\mathcal{W}'$ . By Lemma 3.2, we can assume that every leaf of  $T$  corresponds to a singleton witness set, and vice versa. Let  $L$  be the set of leaves of  $T$ . Then,  $I = \{v \in V(G) \mid v \in W(t), t \in L\}$  is an independent set in  $G$ . Thus,  $X$  is a vertex cover of  $G$ . As  $|F| \leq k$ , we have  $|X| \leq 2k$  as every vertex in  $X$  has an edge incident on it that is in  $F$ . Finally, since the set of non-leaves of a tree induces a subtree, it follows that  $G[X]$  is connected. ◀

Now, we move on to describe a PSAKS for TREE CONTRACTION. We define a partition of vertices of  $G$  into the following three parts:

$$\begin{aligned} H &= \{u \in V(G) \mid d(u) \geq 2k + 1\}, \\ I &= \{v \in V(G) \setminus H \mid N(v) \subseteq H\}, \\ R &= V(G) \setminus (H \cup I). \end{aligned}$$

We define the first reduction rule as follows.

► **Reduction Rule 3.1.** If there is a vertex  $v \in I$  that has at least  $2k + 1$  false twins, then delete  $v$ . That is, the resultant instance is  $(G - \{v\}, k)$ .

► **Lemma 3.6.** *Reduction Rule 3.1 is safe.*

<sup>1</sup> Proofs of results marked  $\star$  have been omitted due to the lack of space. They will appear in the full version of the paper.

**Proof.** Consider a solution  $F'$  of the reduced instance  $(G', k')$ . If  $|F'| \geq k' + 1$ , then the solution lifting algorithm returns  $E(G)$ , otherwise it returns  $F = F'$ . We show that this solution lifting algorithm with the reduction rule constitutes a strict 1-approximate polynomial time preprocessing algorithm. If  $|F'| \geq k' + 1$  then  $\text{TC}(G, k, F) \leq k + 1 = \text{TC}(G', k', F')$ . Otherwise,  $|F'| \leq k$  and let  $T'$  be the tree  $G'/F'$  and  $\mathcal{W}'$  denote the corresponding  $T'$ -witness structure of  $G'$ . Then, as  $v$  has at least  $2k + 1$  false twins, one of these twins, say  $u$ , is not in  $V(F')$ . In other words, there is a vertex  $t$  in  $T'$  such that  $W'(t) = \{u\}$ . By Lemma 3.2,  $t$  is a leaf. Let  $t'$  denote the unique neighbour of  $t$  in  $T'$ . Then, from Observation 3,  $N_{G'}(u) \subseteq W'(t')$ . Let  $T$  be the tree obtained from  $T'$  by adding a new vertex  $t_v$  as a leaf adjacent to  $t'$ . Since  $N_{G'}(u) = N_G(u) = N_G(v)$ , all the vertices in  $N_G(v)$  are in  $W'(t')$ . Define the partition  $\mathcal{W}$  of  $V(G)$  obtained from  $\mathcal{W}'$  by adding a new set  $\{v\}$ . Then,  $G/F$  is  $T$  and  $\mathcal{W}$  is the corresponding  $T$ -witness structure of  $G$ . Hence,  $\text{TC}(G, k, F) \leq \text{TC}(G', k', F')$ .

Next, consider an optimum solution  $F^*$  for  $(G, k)$ . If  $|F^*| \geq k + 1$  then  $\text{OPT}(G, k) = k + 1 \geq \text{OPT}(G', k')$ . Otherwise,  $|F^*| \leq k$  and let  $T = G/F^*$ . Let  $\mathcal{W}^*$  denote the corresponding  $T$ -witness structure of  $G$ . If there is a leaf  $t$  in  $T$  such that  $W^*(t) = \{v\}$ , then  $F^*$  is also a solution for  $(G', k')$  and  $\text{OPT}(G', k') \leq \text{OPT}(G, k)$ . Otherwise, as  $v$  has at least  $2k + 1$  false twins, one of these twins, say  $u$ , is not in  $V(F^*)$ . That is, there is a leaf  $t$  in  $T$  such that  $W^*(t) = \{u\}$ . Define the partition  $\mathcal{W}'$  of  $V(G)$  obtained from  $\mathcal{W}^*$  by replacing  $u$  by  $v$  and  $v$  by  $u$ . Then, the set  $F'$  of edges of  $G$  obtained from  $F$  by replacing the edge  $xv$  with the edge  $xu$  for each  $x$  is also an optimum solution for  $(G, k)$ . Further, it is a solution for  $(G', k')$  and therefore,  $\text{OPT}(G', k') \leq \text{OPT}(G, k)$ . Hence,  $\frac{\text{TC}(G, k, F)}{\text{OPT}(G, k)} \leq \frac{\text{TC}(G', k', F')}{\text{OPT}(G', k')}$ . ◀

Given  $\alpha > 1$ , let  $d$  be the minimum integer such that  $\alpha \geq \frac{d}{d-1}$ . That is,  $d = \lceil \frac{\alpha}{\alpha-1} \rceil$ . The second reduction rule is the following.

► **Reduction Rule 3.2.** If there are vertices  $v_1, v_2, \dots, v_{2k+1} \in I$  and  $h_1, h_2, \dots, h_d \in H$  such that  $\{h_1, \dots, h_d\} \subseteq N(v_i)$  for each  $i \in [2k+1]$  then contract all edges in  $\tilde{E} = \{v_1 h_i \mid i \in [d]\}$  and reduce the parameter by  $d - 1$ . The resulting instance is  $(G/\tilde{E}, k - d + 1)$ .

► **Lemma 3.7.** *Reduction Rule 3.2 is  $\alpha$ -safe.*

**Proof.** Consider a solution  $F'$  of the reduced instance  $(G', k')$ . If  $|F'| \geq k' + 1$ , then the solution lifting algorithm returns  $E(G)$ , otherwise it returns  $F = F' \cup \tilde{E}$ . We will show that this solution lifting algorithm with the reduction rule constitutes a strict  $\alpha$ -approximate polynomial time preprocessing algorithm. First, we prove that  $\text{TC}(G, k, F) \leq \text{TC}(G', k', F') + d$ . If  $|F'| \geq k' + 1$  then  $\text{TC}(G', k', F') = k' + 1$ . In this case,  $F = E(G)$  and  $\text{TC}(G, k, F) \leq k + 1 = k' + d = \text{TC}(G', k', F') + d - 1$ . Consider the case when  $|F'| \leq k'$  and let  $\mathcal{W}' = \{W'(t_1), W'(t_2), \dots, W'(t_l)\}$  be the corresponding  $G'/F'$ -witness structure of  $G$ . Let  $w$  denote the vertex in  $V(G') \setminus V(G)$  obtained by contracting  $\tilde{E}$ . Without loss of generality, assume that  $w \in W'(t_1)$ . Define  $\mathcal{W} = (\mathcal{W}' \cup \{W_1\}) \setminus \{W'(t_1)\}$  where  $W_1 = (W'(t_1) \cup \{v_1, h_1, h_2, \dots, h_d\}) \setminus \{w\}$ . Note that  $V(G) \setminus \{v_1, h_1, h_2, \dots, h_d\} = V(G') \setminus \{w\}$  and hence  $\mathcal{W}$  is partition of  $V(G)$ . Further,  $G[W_1]$  is connected as  $G'[W'(t_1)]$  is connected. A spanning tree of  $G'[W'(t_1)]$  along with  $\tilde{E}$  is a spanning tree of  $G[W_1]$ . Also,  $|W_1| = |W'(t_1)| + d$  and any vertex which is adjacent to  $w$  in  $G'$  is adjacent to at least one vertex in  $\{v_1, h_1, h_2, \dots, h_d\}$  in  $G$ . Thus,  $\mathcal{W}$  is a  $G/F$ -witness structure of  $G$  where  $G/F$  is a tree isomorphic to  $G'/F'$ . Therefore,  $\text{TC}(G, k, F) \leq \text{TC}(G', k', F') + d$ .

We now show that  $\text{OPT}(G', k') \leq \text{OPT}(G, k) - (d - 1)$ . Let  $F^*$  be an optimum solution for  $(G, k)$  and  $\mathcal{W}$  be the corresponding  $G/F^*$ -witness structure of  $G$ . Let  $T$  be  $G/F^*$ . If  $|F^*| \geq k + 1$ , then  $\text{OPT}(G, k) = k + 1 = k' + d \geq \text{OPT}(G', k') + d - 1$ . Otherwise,  $|F^*| \leq k$  and there is at least one vertex, say  $v_q$  in  $\{v_1, v_2, \dots, v_{2k+1}\}$  which is not in



$V(F^*)$ . By Observation 3,  $N(v_q)$  and hence  $\{h_1, h_2, \dots, h_d\}$  are in the same witness set, say  $W(t_i)$  where  $t_i \in V(T)$ . If  $v_1 \in W(t_i)$  then  $F' = F^* \setminus \tilde{E}$  is solution to  $(G', k')$  and so  $\text{OPT}(G', k') \leq |F'| = |F^*| - d = \text{OPT}(G, k) - d$ . Otherwise,  $v_1 \notin W(t_i)$  and let  $t_j \in V(T)$  be the vertex such that  $v_1 \in W(t_j)$ . Then,  $t_i$  and  $t_j$  are adjacent in  $T$ . Define another partition  $\mathcal{W}' = \mathcal{W} \cup \{W(t_{ij})\} \setminus \{W(t_i), W(t_j)\}$  of  $V(G)$  where  $W(t_{ij}) = W(t_i) \cup W(t_j)$ . Clearly,  $G[W(t_{ij})]$  is connected. Thus,  $\mathcal{W}'$  is a  $G/F$ -witness structure of  $G$  where  $|F| = |F^*| + 1$  as  $|W(t_i)| - 1 + |W(t_j)| - 1 = (|W(t_{ij})| - 1) - 1$ . In particular,  $G/F$  is the tree obtained from  $G/F^*$  by contracting the edge  $t_i t_j$ . Finally, without loss of generality  $\tilde{E} \subseteq F$  and thus  $F' = F \setminus \tilde{E}$  is a solution to  $(G', k')$ . Therefore,  $\text{OPT}(G', k') \leq |F'| = |F^*| + 1 - d = \text{OPT}(G, k) - d + 1$ . Combining these bounds, we have  $\frac{\text{TC}(G, k, F)}{\text{OPT}(G, k)} \leq \frac{\text{TC}(G', k', F') + d}{\text{OPT}(G', k') + (d-1)} \leq \max \left\{ \frac{\text{TC}(G', k', F')}{\text{OPT}(G', k')}, \alpha \right\}$ . ◀

It is easy to see that the above rule can be applied in  $\mathcal{O}((2k)^d \cdot n^c)$  time, by considering each subset of  $H$  of cardinality  $d$ , where  $c$  is a constant independent of  $\alpha$  and  $n$  is the number of vertices in the graph. This leads to the following bound.

► **Lemma 3.8.** *Suppose  $G$  is  $k$ -contractible to a tree and neither of the Reduction rules 3.1 and 3.2 are applicable on the instance  $(G, k)$ . Then,  $|V(G)|$  is  $\mathcal{O}((2k)^{d+1} + k^2)$ .*

**Proof.** We will bound  $H$ ,  $I$  and  $R$  separately in order to bound  $V(G)$ . By Lemma 3.5,  $G$  has a connected vertex cover  $S$  of size at most  $2k$ . As  $H$  is the set of vertices of degree at least  $2k + 1$ ,  $H \subseteq S$  and so  $|H| \leq 2k$ . Every vertex in  $R$  has degree at most  $2k$ . Therefore, as  $S \cap R$  is a vertex cover of  $G[R]$ ,  $|E(G[R])|$  is  $\mathcal{O}(k^2)$ . Also, by the definition of  $I$ , every vertex in  $R$  has a neighbour in  $I$  and hence there are no isolated vertices in  $G[R]$ . Thus,  $|R|$  is  $\mathcal{O}(k^2)$ . Finally, we bound the size of  $I$ . For every set  $H' \subseteq H$  of cardinality less than  $d$ , there are at most  $2k + 1$  vertices in  $I$  which have  $H'$  as their neighbourhood. Otherwise, Reduction Rule 3.1 would have been applied. Hence, there are at most  $(2k + 1) \cdot \binom{2k}{d-1}$  vertices in  $I$  which have degree less than  $d$ . Further, for a  $d$ -size subset  $H'$  of  $H$ , there are at most  $2k + 1$  vertices in  $I$  which contain  $H'$  in their neighbourhood. Otherwise, Reduction Rule 3.2 would have been applied. As a vertex in  $I$  of degree at least  $d$  is adjacent to all vertices in at least one such subset of  $H$ , there are at most  $(2k + 1) \binom{2k}{d}$  vertices of  $I$  of degree at least  $d$ . Therefore,  $|I|$  is  $\mathcal{O}((2k)^{d+1})$ . ◀

Now, we have a PSAKS for the problem.

► **Theorem 3.9** (\*). *TREE CONTRACTION admits a strict PSAKS with  $\mathcal{O}((2k)^{\lceil \frac{\alpha-1}{\alpha-1} \rceil + 2} + k^3)$  vertices.*

## 4 Out-Tree Contraction

In this section, we describe a PSAKS for an analogue of TREE CONTRACTION in directed graphs. We first require some terminology on directed graphs. A *directed graph* (or *digraph*) is a pair consisting of a set  $V$  of vertices and a set  $A$  of directed edges (arcs) where  $A \subseteq V \times V$ . An arc is specified as an ordered pair of vertices  $uv$  and we say that the arc  $uv$  is directed from  $u$  to  $v$ . Let  $V(D)$  and  $A(D)$  denote the sets of vertices and arcs of a digraph  $D$ . For a vertex  $v \in V(D)$ ,  $N^-(v)$  denotes the set  $\{u \in V(D) \mid uv \in A(D)\}$  of its in-neighbors and  $N^+(v)$  denotes the set  $\{u \in V(D) \mid vu \in A(D)\}$  of its out-neighbors. The neighborhood of a vertex  $v$  is the set  $N(v) = N^+(v) \cup N^-(v)$ . The in-degree of a vertex  $v$ , denoted by  $d^-(v)$ , is  $|N^-(v)|$ . Similarly, its out-degree is  $|N^+(v)|$  which is denoted by  $d^+(v)$ . The (total) degree of  $v$ , denoted by  $d(v)$ , is the sum of its in-degree and out-degree. A sequence  $P = (v_1, \dots, v_l)$  of distinct vertices of  $D$  is called a directed path in  $D$  if  $v_1 v_2, \dots, v_{l-1} v_l \in A(D)$ .

For a digraph  $D$ , its underlying undirected graph  $G_D$  is the undirected graph on the vertex set  $V(D)$  with the edge set  $\{uv \mid uv \in A(D)\}$ . An out-tree  $T$  is a digraph where each vertex has in-degree at most 1 such that  $G_T$  is a tree. A vertex  $v$  of an out-tree is called a *leaf* if  $d^-(v) = 1$  and  $d^+(v) = 0$ . The *root* of an out-tree is the unique vertex that has no in-neighbor. The contraction of an arc  $e = uv$  in  $D$  results in the digraph, denoted by  $D/e$ , on the vertex set  $V' = V(D) \setminus \{u, v\} \cup \{x\}$  with  $A(D/e) = \{pq \mid pq \in A(D) \text{ and } p, q \in V'\} \cup \{xz \mid vz \in A(D)\} \cup \{zx \mid zu \in A(D)\} \cup \{xz \mid uz \in A(D)\} \cup \{zx \mid zv \in A(D)\}$ . The notion of witness structures and witness sets are extended to digraphs as follows. For digraphs  $D$  and  $T$  with  $V(T) = \{t_1, \dots, t_l\}$ ,  $D$  is said to have a  $T$ -witness structure  $\mathcal{W}$  if  $\mathcal{W}$  is a partition of  $V(D)$  into  $l$  sets (called witness sets) and there is a bijection  $W : V(T) \mapsto \mathcal{W}$  such that the following properties hold.

- For each  $t_i \in V(T)$ ,  $G_D[W(t_i)]$  is connected.
- For a pair  $t_i, t_j \in V(T)$ ,  $t_i t_j \in A(T)$  if and only if there is an arc from a vertex in  $W(t_i)$  to a vertex in  $W(t_j)$  in  $D$ .

Analogous to undirected graphs, we associate a  $T$ -witness structure  $\mathcal{W}$  of  $G$  with a set  $F \subseteq A(D)$  whose contraction in  $D$  results in  $T$  by defining  $F$  to be the set of the arcs corresponding to the edges of a spanning tree of  $G_D[W]$  for each  $W \in \mathcal{W}$ . Now, we show that similar to TREE CONTRACTION, OUT-TREE CONTRACTION also does not admit a polynomial kernel. We modify the reduction known for TREE CONTRACTION to show this hardness.

► **Lemma 4.1** ( $\star$ ). OUT-TREE CONTRACTION *does not have a polynomial kernel unless*  $\text{NP} \subseteq \text{coNP}/\text{poly}$ .

Now, we describe a PSAKS for OUT-TREE CONTRACTION. We note that the simplifying assumptions in TREE CONTRACTION, such as ignoring cut vertices and requiring that the leaves of the resultant tree correspond to singleton witness sets, do not hold anymore. Our first reduction rule is based on the observation that the digraph obtained from an out-tree, by adding a new vertex as an out-neighbor of any vertex, is once again an out-tree.

► **Reduction Rule 4.1.** If there is a vertex  $v \in V(D)$  with  $d^-(v) = 1$  and  $d^+(v) = 0$  then delete  $v$ . The resulting instance is  $(D', k')$  where  $D' = D - \{v\}$  and  $k' = k$ .

► **Lemma 4.2** ( $\star$ ). *Reduction Rule 4.1 is safe.*

The operation of subdividing an arc  $uv$  in  $D$  results in the deletion of the arc  $uv$  and the addition of a new vertex  $w$  as an out-neighbor of  $u$  and an in-neighbor of  $v$ . The next reduction rule is based on the observation that subdividing an arc of an out-tree results in another out-tree. To exploit this observation, we need the following lemma.

► **Lemma 4.3** ( $\star$ ). *Suppose  $D$  has a directed path  $P = (v_0, v_1, \dots, v_l, v_{l+1})$  with  $l > k + 1$  and  $d^-(v) = d^+(v) = 1$  for each  $v \in V(P)$ . Then, no minimal out-tree contraction solution  $F$  of  $D$  with  $|F| \leq k$  contains an arc incident on  $V(P) \setminus \{v_0, v_{l+1}\}$ .*

► **Reduction Rule 4.2.** If there is a directed path  $P = (v_0, v_1, \dots, v_l, v_{l+1})$  with  $l > k + 2$  and  $d^-(v) = d^+(v) = 1$  for each  $v \in V(P)$ , then replace  $P$  by the path  $P' = (v_0, v_1, \dots, v_{k+2}, v_{l+1})$ . Specifically, the resulting instance is  $(D', k' = k)$  where  $D'$  is the digraph obtained from  $D$  by deleting  $\{v_{k+3}, \dots, v_l\}$  and adding the arc  $v_{k+2}v_{l+1}$ .

We note that this rule can be applied in polynomial time by searching for such a path in the subgraph induced on the vertices of degree 2.

► **Lemma 4.4** ( $\star$ ). *Reduction Rule 4.2 is safe.*

Before we describe the next reduction rule, we define the following partition of  $V(D)$ :

$$I = \{v \in V(D) \mid d^+(v) = 0\},$$

$$H = V(D) \setminus I.$$

Now, we apply the following reduction rule on  $I$ .

► **Reduction Rule 4.3.** If there are vertices  $v, v_1, v_2, \dots, v_{2k+1} \in I$  such that  $N^-(v) = N^-(v_1) = \dots = N^-(v_{2k+1})$ , then delete  $v$ . The resulting instance is  $(D', k')$  where  $D' = D - \{v\}$  and  $k' = k$ .

► **Lemma 4.5** ( $\star$ ). *Reduction Rule 4.3 is safe.*

Now, we describe the final reduction rule. Given  $\alpha > 1$ , let  $d$  be the minimum integer such that  $\alpha \geq \frac{d}{d-1}$ .

► **Reduction Rule 4.4.** If there are vertices  $v_1, v_2, \dots, v_{2k+1} \in I$  and  $h_1, h_2, \dots, h_d \in H$  such that  $\{h_1, \dots, h_d\} \subseteq N(v_i)$  for each  $i \in [2k+1]$ , then contract arcs in  $\tilde{A} = \{(h_i v_1) \mid i \in [d]\}$  and reduce the parameter by  $d - 1$ . That is, the resulting instance is  $(D/\tilde{A}, k - (d - 1))$ .

► **Lemma 4.6** ( $\star$ ). *Reduction Rule 4.4 is  $\alpha$ -safe.*

Now, we prove that the reduction rules described lead to a lossy kernel of polynomial size.

► **Lemma 4.7** ( $\star$ ). *Suppose  $D$  is  $k$ -contractible to an out-tree and none of Reduction Rules 4.1, 4.2, 4.3 and 4.4 are applicable on the instance  $(D, k)$ . Then,  $|V(D)|$  is  $\mathcal{O}((2k)^{d+1} + k^2)$ .*

Now, we have the following result.

► **Theorem 4.8** ( $\star$ ). *OUT-TREE CONTRACTION admits a PSAKS with  $\mathcal{O}(k^{2\lceil \frac{\alpha}{\alpha-1} \rceil + 1} + k^2)$  vertices.*

## 5 Cactus Contraction

As mentioned earlier, TREE CONTRACTION has been shown not to admit a polynomial kernel unless  $\text{NP} \subseteq \text{coNP}/\text{poly}$  by a reduction from RED BLUE DOMINATING SET [16]. We modify this reduction to show similar hardness for CACTUS CONTRACTION.

► **Lemma 5.1** ( $\star$ ). *CACTUS CONTRACTION does not have a polynomial kernel unless  $\text{NP} \subseteq \text{coNP}/\text{poly}$ .*

Next, we proceed to describe a PSAKS for CACTUS CONTRACTION. We first list the following simplifying assumption.

► **Lemma 5.2** ( $\star$ ). *A connected graph is  $k$ -contractible to a cactus if and only if each of its 2-connected components is contractible to a cactus using at most  $k$  edge contractions in total.*

So, without loss of generality, we assume that the input graph  $G$  is 2-connected. Before we proceed to describe the reduction rules, we need to define some additional terminology. The operation of *subdividing* an edge  $uv$  results in the graph obtained by deleting  $uv$  and adding a new vertex  $w$  adjacent to both  $u$  and  $v$ . The operation of *short-circuiting* a degree 2 vertex  $v$  with neighbors  $u$  and  $w$  results in the graph obtained by deleting  $v$  and then adding the edge  $uw$  if it is not already present.

► **Observation 4.** The following statements hold for a cactus  $T$ .

1. Every vertex of degree at least 3 in  $T$  is a cut vertex.
2. The graph obtained by subdividing an edge of  $T$  is a cactus.
3. The graph obtained by short-circuiting a degree 2 vertex  $v$  in  $T$  is a cactus.

Next, we will make some observations on a cactus witness structure of a graph. We define the following terms, which are useful for the forthcoming results. For a path  $P$  in  $G$ , let  $N(P)$  denote the neighborhood of  $P$ , i.e. the set of vertices in  $V(G) \setminus V(P)$  that are adjacent to a vertex in  $P$ .

► **Definition 5.3 (Simple Path).** A path  $P$  in a graph  $G$  is called simple path of  $G$ , if every internal vertex of  $P$  has degree exactly equal to two in  $G$ . Observe that, the endpoints of the path  $P$  are the only vertices with a neighbor in  $G \setminus P$ .

We say that a simple path  $P$  is neighbor to a set of vertices  $W$  disjoint from  $V(P)$ , if each vertex in  $W$  is neighbor of at least one of the two endpoints of  $P$ . We denote the set of all neighbors of a simple path  $P$  in  $G$  by  $N(P)$ .

► **Definition 5.4 (Pendent Cycle).** For a cactus  $T$ , a cycle  $C$  is called an pendent cycle if it contains exactly one cut vertex.

Let  $T$  be a cactus obtained by contracting a set of edges in the graph  $G$  with a witness structure  $\mathcal{W}$ . Now, consider a pendent cycle  $(uPu)$  in cactus  $T$  where  $u$  is the cut vertex, and for every  $t \in P$ ,  $|W(t)| = 1$ . Then observe that  $P$  corresponds to a simple path in  $G$ , and with a slight abuse of notation let us use  $P$  to denote the path in  $G$  as well. Observe the set of internal vertices of any two simple paths are disjoint. We say that a simple path  $P$  in  $G$  forms a pendent cycle in  $T$  if there is a cut vertex  $u$  in  $T$  such that  $uPu$  is a pendent cycle in  $T$ . The following lemma establishes a few more properties of the cactus witness structure.

► **Lemma 5.5 ( $\star$ ).** Let  $F$  be a minimal set of edges of a 2-connected graph  $G$  such that  $G/F$  is a cactus  $T$  with  $V(T) = \{t_1, t_2, \dots, t_l\}$  and  $l \geq 3$ . Let  $\mathcal{W}$  be the corresponding  $T$ -witness structure of  $G$ . Then the following properties hold.

1. There exists a set  $F'$  of at most  $|F|$  edges of  $G$  such that  $G/F'$  is a cactus and the corresponding  $G/F'$ -witness structure  $\mathcal{W}'$  of  $G$  satisfies the property that for every leaf  $t$  in  $G/F'$ ,  $W'(t) \in \mathcal{W}'$  is a singleton set.
2. For any three vertices  $u_1, u_2$  and  $u_3$  such that  $W(t_1) = \{u_1\}, W(t_2) = \{u_2\}, W(t_3) = \{u_3\}$ , there is a vertex  $t \in V(T)$  such that  $(N(u_1) \cap N(u_2) \cap N(u_3)) \subseteq W(t)$ . Similarly, for any three simple paths  $P_1, P_2$  and  $P_3$  in  $G$ , such that each of them form a pendent cycle in  $T$ , there is a vertex  $t \in V(T)$  such that  $N_G(P_1) \cap N_G(P_2) \cap N_G(P_3) \subseteq W(t)$ .
3. If  $t$  is cut vertex in  $T$  then  $|W(t)| > 1$ .
4. If  $|F| \leq k$  and  $d(v) \geq k + 3$ , then there is a vertex  $t \in V(T)$  such that  $v \in W(t)$  and  $|W(t)| > 1$ .

Let us remark that, it is safe to assume that  $T$  has at least 3 vertices, as otherwise  $T$  is just an edge. Therefore, we can bound the number of vertices in  $G$  by  $|F| + 2$ , and since we are concerned with solutions of value  $k$  or smaller, this gives us a trivial kernel. Indeed, the main challenge lies in bounding the set of singleton witness sets. Subsequently, we assume that all cactus witness structures have these properties.

► **Lemma 5.6 ( $\star$ ).** Suppose  $G$  has a path  $P = (u_0, u_1, \dots, u_l, u_{l+1})$  with  $l > k + 1$  consisting of vertices of degree 2. Then, no minimal cactus contraction solution  $F$  of  $G$  with  $|F| \leq k$  contains an edge incident on  $V(P) \setminus \{u_0, u_{l+1}\}$ .

Now, we are ready to state the first reduction rule.

► **Reduction Rule 5.1.** If  $G$  has a path  $P = (u_0, u_1, \dots, u_l, u_{l+1})$  such that  $l > k + 2$  consisting of vertices of degree 2, then replace  $P$  by the path  $P' = (u_0, u_1, \dots, u_{k+2}, u_{l+1})$ . In other words, the resulting instance is  $(G', k' = k)$  where  $G'$  is the graph obtained from  $G$  by deleting  $\{u_{k+3}, \dots, u_l\}$  and adding the edge  $u_{k+2}u_{l+1}$ .

We observe that this rule can be applied in polynomial time by considering each simple path in the graph of length more than  $k + 1$ .

► **Lemma 5.7** ( $\star$ ). *Reduction Rule 5.1 is safe.*

We apply Reduction Rule 5.1 exhaustively to the graph  $G$ , and observe that any simple path contains at most  $k + 4$  vertices. Now, we partition  $V(G)$  into the following four parts:

$$\begin{aligned} H &= \{u \in V(G) \mid d(u) \geq k + 3\}, \\ I_v &= \{v \in V(G) \setminus H \mid N_G(v) \subseteq H\}, \\ I_p &= \{V(P) \mid P \text{ is a simple path in } G \text{ and } N_G(P) \subseteq H\}, \\ R &= V(G) \setminus (H \cup I_v \cup I_p). \end{aligned}$$

With a slight abuse of notation, we say that a path  $P$  is contained in  $I_p$  (i.e.  $P \in I_p$ ) if  $V(P) \subseteq I_p$ . Let us make the following observation.

► **Observation 5.** For any two paths  $P_1, P_2 \in I_p$ , we have  $V(P_1) \cap V(P_2) = \emptyset$ .

► **Lemma 5.8** ( $\star$ ). *Let  $G$  be  $k$ -contractible to a cactus such that Reduction Rule 5.1 is not applicable on  $(G, k)$ . If  $H, R$  are the partitions defined above, then  $|H \cup R|$  is at most  $\mathcal{O}(k^4)$ .*

For our next reduction rule, we extend the notion of false twins to simple paths. We call two paths,  $P_1$  and  $P_2$  in  $I_p$ , *false twins* if  $N(P_1) = N(P_2)$ .

► **Reduction Rule 5.2.** If there is a vertex  $v \in I_v$  that has at least  $2k + 3$  false twins, then delete  $v$ . That is, the resultant instance is  $(G - \{v\}, k)$ . Similarly, if there is a path  $P$  in  $I_p$  that has at least  $2k + 3$  false twins, then delete  $P$ .

► **Lemma 5.9** ( $\star$ ). *Reduction Rule 5.2 is safe.*

Given  $\alpha > 1$ , let  $d$  be  $\lceil \frac{\alpha}{\alpha-1} \rceil$ . For every simple path  $P \in I_p$ , such that  $N(P)$  contains at least  $2d$  vertices of  $H$ , pick one of its endpoints, that is adjacent to at least  $d$  vertices of  $H$ , into the set  $\tilde{I}_p$ . We apply the following reduction rule to the set  $I = I_v \cup \tilde{I}_p$ .

► **Reduction Rule 5.3.** If there are vertices  $v_1, v_2, \dots, v_{2k+3} \in I$  and  $h_1, h_2, \dots, h_d \in H$  such that  $\{h_1, \dots, h_d\} \subseteq N(v_i)$  for all  $i \in [2k + 3]$  then contract all edges in  $\tilde{E} = \{v_1 h_i \mid i \in [d]\}$  and reduce the parameter by  $d - 1$ . The resulting instance is  $(G/\tilde{E}, k - d + 1)$ .

► **Lemma 5.10** ( $\star$ ). *Reduction Rule 5.3 is  $\alpha$ -safe.*

This leads to the following result.

► **Lemma 5.11** ( $\star$ ). *Suppose graph  $G$  is  $k$ -contractible to a cactus and none of the Reduction Rules 5.1, 5.2 and 5.3 are applicable on the instance  $(G, k)$ . Then,  $|V(G)|$  is  $\mathcal{O}((2k)^{2d} + k^4)$ .*

► **Theorem 5.12** ( $\star$ ). **CACTUS CONTRACTION** admits a strict PSAKS with  $\mathcal{O}((2k)^{2\lceil \frac{\alpha}{\alpha-1} \rceil + 1} + k^5)$  vertices.

## References

- 1 T. Asano and T. Hirata. Edge-contraction problems. *Journal of Computer and System Sciences*, 26(2):197–208, 1983. doi:10.1016/0022-0000(83)90012-0.
- 2 R. Belmonte, P. A. Golovach, P. van 't Hof, and D. Paulusma. Parameterized complexity of three edge contraction problems with degree constraints. *Acta Informatica*, 51(7):473–497, 2014. doi:10.1007/s00236-014-0204-z.
- 3 H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theoretical Computer Science*, 412(35):4570–4578, 2011. doi:10.1016/j.tcs.2011.04.039.
- 4 A. E. Brouwer and H. J. Veldman. Contractibility and NP-completeness. *Journal of Graph Theory*, 11(1):71–79, 1987.
- 5 L. Cai and C. Guo. *Contracting Few Edges to Remove Forbidden Induced Subgraphs*, pages 97–109. Springer International Publishing, 2013. doi:10.1007/978-3-319-03898-8\_10.
- 6 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer-Verlag, 2015.
- 7 R. Diestel. *Graph Theory*. Springer-Verlag Berlin Heidelberg, 2000.
- 8 M. Dom, D. Lokshtanov, and S. Saurabh. Kernelization lower bounds through colors and IDs. *ACM Transactions on Algorithms (TALG)*, 11(2):13, 2014.
- 9 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer-Verlag London, 2013.
- 10 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
- 11 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- 12 P. A. Golovach, M. Kamiński, D. Paulusma, and D. M. Thilikos. Increasing the minimum degree of a graph by contractions. *Theoretical Computer Science*, 481:74–84, 2013. doi:10.1016/j.tcs.2013.02.030.
- 13 P. A. Golovach, P. van 't Hof, and D. Paulusma. Obtaining planarity by contracting few edges. *Theoretical Computer Science*, 476:38–46, 2013. doi:10.1016/j.tcs.2012.12.041.
- 14 S. Guillemot and D. Marx. A faster FPT algorithm for Bipartite Contraction. *Information Processing Letters*, 113(22–24):906–912, 2013. doi:10.1016/j.ipl.2013.09.004.
- 15 P. Heggernes, P. van 't Hof, D. Lokshtanov, and C. Paul. Obtaining a bipartite graph by contracting few edges. *SIAM Journal on Discrete Mathematics*, 27(4):2143–2156, 2013. doi:10.1137/130907392.
- 16 P. Heggernes, P. van't Hof, B. Lévêque, D. Lokshtanov, and C. Paul. Contracting Graphs to Paths and Trees. *Algorithmica*, 68(1):109–132, 2014.
- 17 D. Lokshtanov, N. Misra, and S. Saurabh. *On the Hardness of Eliminating Small Induced Subgraphs by Contracting Edges*, pages 243–254. Springer International Publishing, 2013. doi:10.1007/978-3-319-03898-8\_21.
- 18 D. Lokshtanov, F. Panolan, M. S. Ramanujan, and S. Saurabh. Lossy kernelization. *CoRR*, abs/1604.04111, 2016.
- 19 B. Martin and D. Paulusma. The computational complexity of disconnected cut and  $2K_2$ -partition. *Journal of Combinatorial Theory, Series B*, 111:17–37, 2015. doi:10.1016/j.jctb.2014.09.002.
- 20 T. Watanabe, T. Ae, and A. Nakamura. On the removal of forbidden graphs by edge-deletion or by edge-contraction. *Discrete Applied Mathematics*, 3(2):151–153, 1981. doi:10.1016/0166-218X(81)90039-1.
- 21 T. Watanabe, T. Ae, and A. Nakamura. On the NP-hardness of edge-deletion and -contraction problems. *Discrete Applied Mathematics*, 6(1):63–78, 1983. doi:10.1016/0166-218X(83)90101-4.



# Faster Exact and Parameterized Algorithm for Feedback Vertex Set in Bipartite Tournaments\*

Mithilesh Kumar<sup>1</sup> and Daniel Lokshtanov<sup>2</sup>

- 1 Department of Informatics, University of Bergen, Norway  
Mithilesh.Kumar@ii.uib.no
- 2 Department of Informatics, University of Bergen, Norway  
daniello@ii.uib.no

---

## Abstract

A *bipartite tournament* is a directed graph  $T := (A \cup B, E)$  such that every pair of vertices  $(a, b), a \in A, b \in B$  are connected by an arc, and no arc connects two vertices of  $A$  or two vertices of  $B$ . A *feedback vertex set* is a set  $S$  of vertices in  $T$  such that  $T - S$  is acyclic. In this article we consider the FEEDBACK VERTEX SET problem in bipartite tournaments. Here the input is a bipartite tournament  $T$  on  $n$  vertices together with an integer  $k$ , and the task is to determine whether  $T$  has a feedback vertex set of size at most  $k$ . We give a new algorithm for FEEDBACK VERTEX SET IN BIPARTITE TOURNAMENTS. The running time of our algorithm is upper-bounded by  $O(1.6181^k + n^{O(1)})$ , improving over the previously best known algorithm with running time  $2^k k^{O(1)} + n^{O(1)}$  [Hsiao, ISAAC 2011]. As a by-product, we also obtain the fastest currently known exact exponential-time algorithm for the problem, with running time  $O(1.3820^n)$ .

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Parameterized algorithms, Exact algorithms, Feedback vertex set, Tournaments, Bipartite tournaments

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.24

## 1 Introduction

A *feedback vertex set* in a graph  $G$  is a vertex set whose removal makes the graph acyclic. The FEEDBACK VERTEX SET problem is a well-studied graph problem where input is a graph  $G$  (directed or undirected) and the task is to find a smallest possible feedback vertex set. Finding such an optimal feedback vertex set turns out to be NP-complete [21], indeed the problem is one of the very first to be shown NP-complete in the influential paper of Karp [25]. Since, polynomial time algorithms are highly unlikely, FEEDBACK VERTEX SET on general directed and undirected graphs has been extensively studied from the perspective of approximation algorithms [2, 14], parameterized algorithms [6, 9, 26], exact exponential-time algorithms [28, 34] as well as graph theory [13, 29].

This paper belongs to a long line of work studying the complexity of FEEDBACK VERTEX SET on restricted classes of graphs. On one hand FEEDBACK VERTEX SET remains NP-complete on tournaments and bipartite tournaments [5], planar undirected graphs [21], planar directed graphs with in-degree and out-degree at most 3 [21] as well as directed graphs with

---

\* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. 306992 and the Beating Hardness by Pre-processing grant funded by the Bergen Research Foundation.



© Mithilesh Kumar and Daniel Lokshtanov;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 24; pp. 24:1–24:15



Leibniz International Proceedings in Informatics

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

in-degree and out-degree at most 2 [21]. On the other hand the problem is polynomial time solvable on undirected graphs of maximum degree 3 [32], chordal graphs [15] and weakly chordal graphs [19], indeed on any class of graphs with polynomially many potential maximal cliques [19]. Being a problem of fundamental importance, FEEDBACK VERTEX SET has been approached algorithmically even on the classes of graphs where it remains NP-complete. For example the problem admits (efficient) polynomial time approximation schemes [8, 11, 17], sub-exponential time parameterized algorithms [10] and linear kernels [18] on classes of graphs excluding a fixed graph  $H$  as a minor. In this paper we study the problem on *bipartite tournaments*.

A *tournament* is a subclass of directed graphs where every pair of vertices are connected by an arc. A *bipartite tournament* is a directed graph where the vertices are partitioned into two sets  $A$  and  $B$ , there is an arc connecting every vertex in  $A$  with every vertex in  $B$ , and there are no edges between vertices of  $A$  and vertices of  $B$ . Tournaments arise naturally from round-robin competitions whereas bipartite tournaments model a two-team competition in which every player in one team plays against every player of the other team. Here arcs are drawn from the winning to the losing player, and often one seeks to rank the players from “best” to “worst” such that players that appear higher in the ranking beat all lower ranked players they played against. Such an absolute ranking possible only if there are no cycles in the tournament. The size of the smallest feedback vertex set then becomes a measure of how far the tournament is from admitting a consistent ranking. For this reason the structure of cycles and feedback vertex sets in (bipartite) tournaments has been studied both from the perspective of graph theory [3, 7, 20] and algorithms.

For bipartite tournaments, finding a feedback vertex set reduces to hitting all cycles of length 4. For this reason the FEEDBACK VERTEX SET problem is more computationally tractable on bipartite tournaments than on general directed graphs. Specifically the best known approximation algorithm for FEEDBACK VERTEX SET on directed graphs has an approximation factor of  $O(\log n \cdot \log \log n)$  [14], and the problem does *not* admit a constant factor approximation assuming the Unique Games Conjecture [22]. On bipartite tournaments it is easy to obtain a 4-approximation (see Lemma 2). Further, an improved approximation algorithm with ratio 2 was obtained by Zuylen. [33].

Similarly, it was open for a long time whether FEEDBACK VERTEX SET on general directed graphs admits an FPT algorithm, that is an algorithm that determines whether there exists a solution of size at most  $k$  in time  $f(k)n^{O(1)}$ . In 2008, Chen et al. [6] gave an algorithm with running time  $O(4^k k^{O(1)} k! nm)$ , and it is an outstanding open problem whether there exists an algorithm with running time  $2^{O(k)} n^{O(1)}$ . For bipartite tournaments, the realization that it is necessary and sufficient to hit all cycles of length 4 yields a simple  $4^k n^{O(1)}$  time parameterized algorithm: recursively branch on vertices of a cycle of length 4. Truß [31] gave an improved algorithm with running time  $3.12^k n^{O(1)}$ , Sasatte [30] further improved the running time to  $3^k n^{O(1)}$ , while Hsiao [24] gave an algorithm with running time  $2^k n^{O(1)}$ . Prior to this work, this was the fastest known parameterized algorithm for FEEDBACK VERTEX SET on bipartite tournaments. Our main result is an algorithm with running time  $O(1.6181^k + n^{O(1)})$ . Using the recent black-box reduction from parameterized to exact exponential time algorithms of Fomin et al. [16] we also obtain an exponential-time algorithm running in  $O(1.3820^n)$  time.

**Methods.** Our algorithm is based on the recent parameterized algorithm with running time  $O(1.6181^k + n^{O(1)})$  by the authors [27] for FEEDBACK VERTEX SET in tournaments. The main idea of this algorithm is that tournaments are very *rigid*. Given as input a tournament



$T$ , by obtaining a large set  $M$  of vertices that is *disjoint* from the feedback vertex set  $H$  sought for, we can get a rough sketch of the rigid structure of  $T - H$ . This structure is then very useful for recovering the solution  $H$ . Indeed, the only way that vertices that are “far apart” in the approximate sketch of the structure of  $T - H$  can interact with each other is by being “in conflict”. Out of two vertices that are in conflict, one of them has to be deleted. Thus, dealing with conflicts can be done in a similar fashion as with edges in the VERTEX COVER problem. For any vertex  $v$  appearing in at least two conflicts, branch into two sub-problems. In the first sub-problem  $v$  is deleted, in the second all vertices in conflict with  $v$  are deleted. If there are *no* conflicts it is sufficient to solve the FEEDBACK VERTEX SET problem “locally”. If every vertex appears in at most one conflict a divide and conquer approach can be taken.

Because bipartite tournaments are also quite “rigid”, we expected that the same approach would easily give an algorithm for FEEDBACK VERTEX SET on bipartite tournaments with the same running time. Our expectations were both wrong and correct; indeed we *do* obtain an algorithm for FEEDBACK VERTEX SET on bipartite tournaments with the same template and the same running time as the algorithm for tournaments [27], yet the adaptation turned out to be anything but easy. Specifically, in virtually every step of the algorithm, the lack of a unique topological sort of acyclic bipartite tournaments presented significant challenges.

The fact that these challenges still could be overcome by sub-exponential time cleaning procedures gives hope that the same template could be applicable in several situations where one seeks a “small” set of vertices or edges to delete in order to modify the input graph to a “rigid” structure; such as CLUSTER VERTEX DELETION, COGRAPH VERTEX DELETION and FEEDBACK VERTEX SET in the more general setting when the input graph is a multi-partite tournament [23].

**Organization of the paper.** In Section 2 we set up definitions and notation, and state a few useful preliminary results. The standard graph notation and parameterized complexity terminology is set up in the appendix. In Section 3 we define and prove some properties of  $M$ -sequence. In Section 4 we define and give an algorithm for Constrained Feedback Vertex Set problem.

## 2 Preliminaries

**Preliminary Results.** If a bipartite tournament is acyclic then it does not contain any squares. It is a well-known and basic fact that the converse is also true, see e.g. [12].

► **Lemma 1** ([12]). *A bipartite tournament is acyclic if and only if it contains no squares.*

Lemma 1 immediately gives rise to a folklore greedy 4-approximation algorithm for BTFVS: as long as  $T$  contains a square, delete all the vertices in this square.

► **Lemma 2** (folklore). *There is a polynomial time algorithm that given as input a bipartite tournament  $T$  and integer  $k$ , either correctly concludes that  $T$  has no feedback vertex set of size at most  $k$  or outputs a feedback vertex set of size at most  $4k$ .*

In fact, BTFVS has a polynomial time factor 3.5-approximation, due to Cai et al. [4]. However, the simpler algorithm from Lemma 2 is already suitable to our needs. The preliminary phase of our algorithm for BTFVS is the kernel of Dom et al. [12]. We will need some additional properties of this kernel that we state here. Essentially, Lemma 3 allows us to focus on the case when the number of vertices in the input bipartite tournament is  $O(k^3)$ .

► **Lemma 3** ([12]). *There is a polynomial time algorithm that given as input a bipartite tournament  $T$  and integer  $k$ , runs in polynomial time and outputs a bipartite tournament  $T'$  and integer  $k'$  such that  $|V(T')| \leq |V(T)|$ ,  $|V(T')| = O(k^3)$ ,  $k' \leq k$ , and  $T'$  has a feedback vertex set of size at most  $k'$  if and only if  $T$  has a feedback vertex set of size at most  $k$ .*

For any sequence  $\sigma$ , let  $|\sigma|$  denote the length of  $\sigma$ . For each  $i = 1, 2, \dots, |\sigma|$ , let  $V_i$  be the  $i$ -th element of  $\sigma$ . Let  $T$  be an  $n$ -vertex acyclic bipartite tournament. The *canonical sequence* for  $T$  is the sequence  $\sigma$  of vertex sets that can be obtained from  $T$  in  $O(n^2)$  time as follows: For each  $i \geq 1$ , let  $V_i$  consist of the vertices without incoming edges in  $T \setminus \bigcup_{j=1}^{i-1} V_j$ .

► **Lemma 4** ([24]). *Let  $T$  be an  $n$ -node acyclic bipartite tournament. Let  $\sigma$  be the canonical sequence for  $T$ . The following statements hold. (i)  $V_1, V_2, \dots, V_{|\sigma|}$  form a partition of  $V(T)$ . (ii) For each directed edge  $(u, v)$  of  $T$ , the vertex set  $V_i$  containing  $u$  precedes the vertex set  $V_j$  containing  $v$  in the sequence (i.e.  $i < j$ ). (iii)  $A = \bigcup_{i \equiv 1 \pmod 2} V_i$  and  $B = \bigcup_{i \equiv 0 \pmod 2} V_i$  are the partite sets of  $T$ .*

► **Definition 5** (*t-wise independent*). A family  $H_{n,t,q}$  of functions from  $[n]$  to  $[q]$  is called a *t-wise independent sample space* if, for every  $t$  positions  $1 < i_1 < i_2 < \dots < i_t \leq n$ , and every tuple  $\alpha \in [q]^t$ , we have  $\Pr(f(i_1), f(i_2), \dots, f(i_t)) = \alpha = q^{-t}$  where the function  $f \in H_{n,t,q}$  is chosen uniformly at random.

► **Theorem 6** ([1]). *There exists a t-wise independent sample space  $H_{n,t,q}$  of size  $O(n^t)$  and it can be constructed efficiently in time linear in the output size.*

### 3 M-Sequence

First we extend the notion of the canonical sequence to general bipartite tournaments relative to a set  $M$  of vertices.

► **Definition 7** (*M-equivalent*). Given a directed graph  $T$  and a subset  $M \subseteq V(T)$ , two vertices  $u, v \in V(T)$  are said to be *M-equivalent* if  $N^+(u) \cap M = N^+(v) \cap M$  and  $N^-(u) \cap M = N^-(v) \cap M$ .

► **Definition 8** (*(M, X)-equivalent*). Let  $T$  be a bipartite tournament and a subset  $M \subseteq V(T)$  such that  $T[M]$  is acyclic. Let  $(X_1, X_2, \dots)$  be the canonical sequence of  $T[M]$ . For any set  $X_i$  in the canonical sequence of  $T[M]$  and any vertex  $v \in V(T)$ ,  $v$  is called *(M, X<sub>i</sub>)-equivalent* if  $v$  is *M-equivalent* to a vertex in  $X_i$ .

► **Definition 9** (*(M, X)-conflicting*). Let  $T$  be a bipartite tournament and a subset  $M \subseteq V(T)$  such that  $T[M]$  is acyclic. Let  $(X_1, X_2, \dots)$  be the canonical sequence of  $T[M]$ . For any set  $X_i$  in  $(X_1, X_2, \dots)$  and for any vertex  $v \in V(T)$ ,  $v$  is called *(M, X<sub>i</sub>)-conflicting* if

- $N^+(v) \cap X_i \neq \emptyset$  and  $N^-(v) \cap X_i \neq \emptyset$ ,
- for every  $j < i$ ,  $N^+(v) \cap X_j = \emptyset$  and for every  $j > i$ ,  $N^-(v) \cap X_j = \emptyset$ .

► **Definition 10** (*M-consistent*). Let  $T$  be a directed graph and  $M \subseteq V(T)$ .  $T$  is called *M-consistent* if for every vertex  $v \in V(T)$   $T[M \cup \{v\}]$  is acyclic.

As a direct consequence of the above definitions, we have the following lemma.

► **Lemma 11**. *Let  $T$  be an M-consistent bipartite tournament for some subset  $M \subseteq V(T)$ . Let  $(X_1, X_2, \dots, X_i, X_{i+1}, \dots)$  be the canonical sequence of  $T[M]$ . Let  $v \in V(T)$  be a *(M, X<sub>i</sub>)-conflicting* vertex. Then, the canonical sequence of  $T[M \cup \{v\}]$  is  $(X_1, X_2, \dots, X'_i, \{v\}, X''_i, X_{i+1}, \dots)$  where  $X'_i \cup X''_i = X_i$  such that  $X'_i, X''_i \neq \emptyset$ .*

► **Definition 12** (*M*-universal). Let  $T$  be a bipartite tournament and a subset  $M \subseteq V(T)$  such that  $T[M]$  is acyclic. Let  $(X_1, X_2, \dots)$  be the canonical sequence of  $T[M]$ . A vertex  $v \in V(T)$  is called *M*-universal if the following holds: (i)  $v$  is not  $(M, X_i)$ -equivalent for any  $X_i$ , (ii)  $T[M \cup \{v\}]$  is acyclic. (iii) There exists a topological sort of  $T[M \cup \{v\}]$  such that  $v$  is either the first vertex (called  $M^-$ -universal) or is the last vertex (called  $M^+$ -universal) in the ordering.

► **Lemma 13.** *Let  $T$  be an  $M$ -consistent bipartite tournament and let  $(X_1, X_2, \dots)$  be the canonical sequence of  $T[M]$ . Then, for every vertex  $v \in V(T)$ , there exists a unique index  $i$  such that  $v$  satisfies exactly one of the following properties: (i)  $v$  is  $(M, X_i)$ -equivalent, (ii)  $v$  is  $(M, X_i)$ -conflicting, (iii)  $v$  is  $M$ -universal.*

**Proof.** Since  $T$  is  $M$ -consistent,  $T[M \cup \{v\}]$  is acyclic. By definition,  $v$  can not satisfy more than one property. If  $v$  is  $M$ -universal, then,  $v$  is neither  $(M, X_i)$ -equivalent nor  $(M, X_i)$ -conflicting for any set  $X_i$ .

If  $v$  is  $(M, X_i)$ -equivalent to some set  $X_i$ , then by definition,  $v$  is not  $M$ -universal. In addition, for any set  $X_j$ ,  $v$  is not  $(M, X_j)$ -conflicting as no vertex in  $X_i$  is  $(M, X_j)$ -conflicting.

Suppose that  $v$  is neither  $(M, X_i)$ -equivalent for any  $X_i$  nor  $M$ -universal. We show that  $v$  is  $(M, X_i)$ -conflicting the first set  $X_i$  that contains an out-neighbor  $u_i$  of  $v$ . Suppose that there is an index  $j > i$  such that  $X_j$  contains an in-neighbor  $u_j$  of  $v$ . Since  $j - i \geq 2$ , there is an index  $i < l < j$  such that  $X_l$  lies in the partite set of  $T$  different from  $X_i \cup X_j$ . This gives us a cycle  $vu_iu_lu_jv$  where  $u_l \in X_l$  contradicting that  $T[M \cup \{v\}]$  is acyclic. If every vertex in  $X_i$  is an out-neighbor of  $v$ , then by definition of the canonical sequence,  $v$  is  $(M, X_{i-1})$ -equivalent contradicting the above assumption. Hence,  $X_i$  contains an in-neighbor of  $v$ , thereby proving that  $v$  is  $(M, X_i)$ -conflicting. ◀

► **Definition 14** (*M*-sequence). Let  $T$  be an  $M$ -consistent bipartite tournament and  $(X'_1, X'_2, \dots)$  be the canonical sequence of  $T[M]$ . An *M*-sequence  $(X_1, Y_1, X_2, Y_2, \dots, X_l, Y_l)$  of  $T$  is a sequence of subsets  $V(T)$  such that for every index  $i$ ,  $X_i$  is the set of all vertices in  $V(T)$  that are  $(M, X'_i)$ -equivalent and  $Y_i$  is the set of vertices that are  $(M, X'_i)$ -conflicting. In addition,  $Y_1$  contains every  $M^-$ -universal vertex and  $Y_l$  contains every  $M^+$ -universal vertex. For every  $i$ , the set  $X_i \cup Y_i$  is called a *block*,  $X_i$  is called the *M*-sub-block and  $Y_i$  is called the  $\bar{M}$ -sub-block.

► **Lemma 15.** *If  $T$  is an  $M$ -consistent bipartite tournament, then  $T$  has a unique  $M$ -sequence.*

**Proof.** The existence and the uniqueness of  $M$ -sequence follows from Lemma 13 and the uniqueness of the canonical sequence of  $T[M]$ . ◀

As a consequence of Lemma 4 and Lemma 13, we get the following lemma.

► **Lemma 16.** *Let  $T := (A, B, E)$  be an  $M$ -consistent bipartite tournament and  $(X'_1, X'_2, \dots)$  be the canonical sequence of  $T[M]$ . Let  $(X_1, Y_1, X_2, Y_2, \dots)$  be the  $M$ -sequence of  $T$ . The following statements hold: (i)  $X_1, Y_1, X_2, Y_2, \dots$  form a partition of  $V(T)$ , (ii) for each  $i$ ,  $X'_i \subseteq X_i$ , (iii) for each  $i$ ,  $Y_i \cap M = \emptyset$ , (iv) for every odd  $i$ ,  $X_i \subseteq A, Y_i \subseteq B$  and for every even  $i$ ,  $X_i \subseteq B, Y_i \subseteq A$ .*

► **Definition 17** (Refinement). A partition  $(V_1, V_2, \dots)$  of  $U$  is said to be a refinement of another partition  $(V'_1, V'_2, \dots)$  if for every set  $V_i$  and  $V'_j$ , either  $V_i \subseteq V'_j$  or  $V_i \cap V'_j = \emptyset$ .

► **Lemma 18.** *Let  $T$  be an acyclic bipartite tournament. Then, for any subset  $M \subseteq V(T)$ , the canonical sequence of  $T$  is a refinement of the  $M$ -sequence of  $T$ .*

**Proof.** Let  $(X'_1, X'_2, \dots)$  be the canonical sequence of  $T[M]$  and let  $(X_1, Y_1, \dots, X_l, Y_l)$  be the  $M$ -sequence of  $T$ . Let  $(V_1, V_2, \dots)$  be the canonical sequence of  $T$ . Since each set  $V_i$  are twins in  $T$ , if any vertex in  $V_i$  belongs to  $X_j$ , then every vertex in  $V_i$  belongs to  $X_j$ . If any vertex in  $V_i$  is  $(M, X'_j)$ -conflicting, then every vertex in  $V_i$  is  $(M, X'_j)$ -conflicting. Hence,  $V_i \subseteq Y'_j$ . If any vertex in  $V_i$  is  $M$ -universal, then every vertex in  $V_i$  is  $M$ -universal. Hence,  $V_i \subseteq Y_1$  or  $V_i \subseteq Y_l$ . The family of sets  $V_i$  that contain an  $M^-$ -universal vertex lie in  $Y_1$  and the family of sets  $V_i$  that contain an  $M^+$ -universal vertex lie in  $Y_l$ .  $\blacktriangleleft$

► **Lemma 19.** *Let  $T$  and  $T \cup \{v\}$  be two  $M$ -consistent bipartite tournaments and let  $(X_1, Y_1, \dots)$  be the  $M$ -sequence of  $T$ . Then, there exists an index  $i$ , such that the  $M$ -sequence of  $T \cup \{v\}$ , is either  $(X_1, Y_1, \dots, X_i \cup \{v\}, \dots)$  or  $(X_1, Y_1, \dots, Y_i \cup \{v\}, \dots)$ .*

**Proof.** The proof follows from Lemma 13.  $\blacktriangleleft$

► **Lemma 20.** *Let  $T$  be a bipartite tournament and  $H$  be a feedback vertex set of  $T$ . Let  $M \subseteq T - H$  and  $P \subseteq H$ . Let  $(X_1, Y_2, \dots, X_l, Y_l)$  be the  $M$ -sequence of  $T - H$  and  $(X'_1, Y'_1, \dots, X'_l, Y'_l)$  be the  $M$ -sequence of  $T - P$ . Then, for each index  $i$ ,  $X_i \subseteq X'_i$  and  $Y_i \subseteq Y'_i$ .*

#### 4 Constrained Feedback Vertex Set in Bipartite Tournaments

Given a tournament  $T$  and an integer  $k$ , in the first phase of the algorithm for feedback vertex set in tournaments of Kumar and Lokshtanov in [27], a family of sub-exponential size of vertex set pairs  $(M, P)$  was obtained such that the sought solution  $H$  is disjoint from  $M$  and contains  $P$ . The uniqueness of the topological sort of an acyclic tournament implied that any edge going from right to left (referred as *back edge*) over an  $M$ -vertex becomes a conflict edge and must be hit by  $H$ . The lack of a unique topological sort of an acyclic bipartite tournament breaks down this step as there may be a topological sort of the bipartite tournament such that a back edge is not a conflict edge. We notice that maintaining an addition subset of back edges  $F$  that must be hit by  $H$  helps in circumventing this issue. With this strategy in mind, we define the CONSTRAINED FEEDBACK VERTEX SET problem.

► **Definition 21** (Constrained Feedback Vertex Set(CFVS)). Let  $T$  be a bipartite tournament with vertex subsets  $M, P \subseteq V(T)$ , edge set  $F \subseteq E(T)$ . A feedback vertex set  $H$  of  $T$  is called  $(M, P, F)$ -constrained if  $M \cap H = \emptyset$ ,  $P \subseteq H$  and  $H$  is a vertex cover for  $F$ .

##### CONSTRAINED FEEDBACK VERTEX SET (CFVS)

**Input:** A bipartite tournament, vertex sets  $M, P \subseteq V(T)$ , edge set  $F \subseteq E(T)$  and positive integer  $k$ .

**Parameter:**  $k$

**Task:** determine whether  $T$  has an  $(M, P, F)$ -constrained CFVS  $H$  of size at most  $k$ .

In the rest of the paper, we assume that the size of the bipartite tournament is at most  $O(k^3)$  as a bi-product of the kernelization algorithm (Lemma 3). Given a topological sort  $\pi$  of an acyclic bipartite tournament  $T = (A, B, E)$ , we denote  $\pi_A$  to be the permutation of  $A$  when  $\pi$  is restricted to  $A$ . Similarly,  $\pi_B$  denotes the permutation of  $B$  when  $\pi$  is restricted to  $B$ . Next, we define a property of a feedback vertex set of a bipartite tournament and while solving for BTFVS, we will look for solutions  $H$  that have this property.

► **Definition 22** ( $M$ -homogeneous). Let  $T$  be a bipartite tournament and  $k$  be a positive integer. Let  $M \subseteq V(T)$  be a vertex subset such that  $T[M]$  is acyclic. A feedback vertex

set  $H$  of size at most  $k$  of  $T$  is called  $M$ -homogeneous if there exists a topological sort  $\pi$  of  $T - H$  such that every subset of  $10 \log^3 k$  consecutive vertices in  $\pi_{A-H}$  or  $\pi_{B-H}$  contains a vertex of  $M$ .

The algorithm for CFVS is primarily based on branching and often, given a CFVS instance, a family of CFVS instances with addition properties will be constructed. We abstract it out in the following definition.

► **Definition 23** ( $\gamma$ -reduction). A  $\gamma$ -reduction is an algorithm that given a CFVS instance  $(T, M, P, F, k)$  outputs in time  $\gamma$  a family  $\mathcal{C} := \{(T, M, P_1, F_1, k), (T, M, P_1, F_1, k), \dots\}$  of size  $\gamma$  of CFVS instances such that

**Forward direction** if  $(T, M, P, F, k)$  has an  $M$ -homogeneous  $(M, P, F)$ -solution, then there exists an instance  $(T, M, P_i, F_i, k) \in \mathcal{C}$  that has an  $M$ -homogeneous  $(M, P_i, F_i)$  solution.

**Backward direction** if there exists an instance  $(T, M, P_i, F_i, k) \in \mathcal{C}$  that has an  $(M, P_i, F_i)$ -CFVS solution, then  $(T, M, P, F, k)$  has an  $(M, P, F)$ -solution.

Now, we construct a family of sets  $\mathcal{M}$  such that if  $(T, k)$  has a solution  $H$  of size at most  $k$ , then there is a set  $M \in \mathcal{M}$  such that  $H$  is  $M$ -homogeneous, and hence we can restrict our attention to looking for feedback vertex sets which are  $M$ -homogeneous for some subset  $M$ .

► **Lemma 24.** *There exists an algorithm that given a bipartite tournament  $T$  and a positive integer  $k$  outputs in time  $\gamma$ , a family  $\mathcal{M}$  of size  $\gamma$  of subsets of  $V(T)$  for  $\gamma = 2^{O(\frac{k}{\log k})}$  such that for every feedback vertex set  $H$  of size at most  $k$  of  $T$ , there exists  $M \in \mathcal{M}$  such that  $H$  is  $M$ -homogeneous.*

**Proof.** Using  $T$  and  $k$ , we construct  $\mathcal{M}$ . Let  $n = |V(T)|$ ,  $t = 10 \log^3 k$ ,  $q = \log^2 k$ . As the first step, the algorithm uses Theorem 6 to construct a family of functions  $H_{n,t,q}$  from  $[n]$  to  $[q]$ . Next, the algorithm computes a family  $\mathcal{Z}$  of  $t$ -wise independent subsets of  $V(T)$ : For each  $f \in H_{n,t,q}$ , let  $Z_f := \{v_i \in V(T) \mid f(i) = 1\}$ . Add  $Z$  to  $\mathcal{Z}$ . In the next step, for every subset  $Z \in \mathcal{Z}$ , compute the family of subsets  $\mathcal{M}_Z := \{M := Z \setminus \hat{H} \mid \hat{H} \subseteq Z, |\hat{H}| \leq \frac{2k}{\log^2 k}\}$ . Finally, output  $\mathcal{M} := \bigcup_{Z \in \mathcal{Z}} \mathcal{M}_Z$ .

To argue about the correctness of the algorithm, first, we check that the size of  $\mathcal{M}$  computed by the above algorithm is consistent with the claim in the lemma. Clearly,  $|\mathcal{M}| \leq |H_{n,t,q}| \times |\mathcal{M}_Z| = O(n^t) O((k^3)^{\frac{2k}{\log^2 k}}) = 2^{O(\frac{k}{\log k})}$ . We need to show that for every feedback vertex set  $H$  of size  $k$  and for every topological sort  $\pi$  of  $T - H$ , there exists a function  $f \in H_{n,t,q}$  and a set  $\hat{H} \subseteq V(T)$  such that  $M := Z \setminus \hat{H}$  satisfies the required properties. Fix a feedback vertex set  $H$  of size  $k$  and a topological sort  $\pi$  of  $T - H$ . First, we prove the following claim:

► **Claim 25.** *If we pick  $f$  from  $H_{n,t,q}$  uniformly at random, then with non-zero probability, the following two events happen: (i) for every set of  $10 \log^3 k$  consecutive vertices in  $\pi_{A-H}$  or  $\pi_{B-H}$ , there is a vertex in  $Z_f$ , (ii)  $|Z_f \cap H| \leq \frac{2k}{\log^2 k}$ .*

**Proof.** By  $t$ -wise independence of  $H_{n,t,q}$ , the probability that no vertex is picked from  $t$  consecutive vertices in  $\pi_{A-H}$  or  $\pi_{B-H}$  is at most  $(1 - \frac{1}{q})^t$ . Let  $\mathcal{A}_1$  be the event that at least one set of  $t$ -consecutive vertices either in  $\pi_{A-H}$  or in  $\pi_{B-H}$  does not contain any vertex from  $Z$ . Since there are at most  $n$  sets of  $t$ -consecutive vertices, by union bound, the probability that event  $\mathcal{A}_1$  happens is at most  $n \times (1 - \frac{1}{q})^t \leq Ck^4 \times (1 - \frac{1}{\log^2 k})^{10 \log^3 k} = Ck^4 \times \frac{1}{k^{10}} \leq \frac{1}{k^5}$ . Let  $\mathcal{A}_2$  be the event that at least  $\frac{2k}{\log^2 k}$  vertices of  $H$  are in  $Z$ . The expected number of vertices of  $H$  that belong to  $Z$  is  $k \times \frac{1}{q} = \frac{k}{\log^2 k}$ . Therefore, by Markov's inequality, the probability that the event  $\mathcal{A}_2$  occurs is at most  $\frac{1}{2}$ . By union bound the probability that at least one of

the events  $\mathcal{A}_1$  or  $\mathcal{A}_2$  happen is at most  $\frac{1}{k^5} + \frac{1}{2}$ . Hence, the probability that none of  $\mathcal{A}_1$  and  $\mathcal{A}_2$  is at least  $1 - (\frac{1}{k^5} + \frac{1}{2}) > 0$ , thereby implying the claim.  $\blacktriangleleft$

Hence, the set of functions satisfying the properties in the above claim is non-empty. Let  $f$  be such a function. Since,  $\mathcal{M}_Z$  is the collection of sets  $Z \setminus \hat{H}$  such that  $|\hat{H}| \leq \frac{2k}{\log^2 k}$ , there exists a choice  $\hat{H}$  such that  $\hat{H} = Z \cap H$ . Hence,  $M := Z \setminus \hat{H}$  satisfies the required properties.

For the runtime of the algorithm,  $H_{n,t,q}$  can be constructed in  $O(n^t)$  time. For each function  $f \in H_{n,t,q}$ , the set  $Z$  can be obtained in  $O(n)$  time. For each  $Z$ ,  $\mathcal{M}_Z$  can be obtained in  $O(k^{\frac{2k}{\log^2 k}})$  time. Hence, the runtime of the algorithm is  $O(n^t) \cdot n \cdot O(k^{\frac{2k}{\log^2 k}}) = 2^{O(\frac{k}{\log k})}$ .  $\blacktriangleleft$

**► Lemma 26.** *There exists an algorithm that given a BTFVS instance  $(T, k)$  outputs in time  $\gamma$ , a family  $\mathcal{C} := \{(T, M_1, P_1, \emptyset, k), (T, M_2, P_2, \emptyset, k), \dots\}$  of size  $\gamma$  of CFVS instances for  $\gamma = 2^{O(\frac{k}{\log k})}$  such that (a) if  $(T, k)$  has a solution  $H$  of size at most  $k$ , then  $\mathcal{C}$  has a CFVS instance  $(T, M, P, \emptyset, k)$  that has an  $M$ -homogeneous solution of size at most  $k$  and (b) if  $\mathcal{C}$  has a  $(M, P, \emptyset)$ -constrained solution, then  $(T, k)$  has a feedback vertex set of size at most  $k$ .*

**► Definition 27 (boundary, vicinity).** Let  $T$  be an acyclic bipartite tournament. Let  $M$  be any subset of vertices and  $\pi$  be a topological sort of  $T$ . Let  $(X_1, Y_1, \dots)$  be the  $M$ -sequence of  $T$ . For any block  $X_i \cup Y_i$ , the set of vertices in  $X_i$  before the first  $M$ -vertex is called the left boundary of the block and the set of vertices in  $X_i$  after the last  $M$ -vertex is called the right boundary of the block. The vicinity of the block  $X_i \cup Y_i$  is the union of the boundaries of  $X_i \cup Y_i$ , the right boundary of  $X_{i-1} \cup Y_{i-1}$ ,  $Y_i$  and the left boundary of  $X_{i+1} \cup Y_{i+1}$ .

**► Lemma 28.** *Let  $H$  be an  $M$ -homogeneous solution for a bipartite tournament  $T$ . Then, in the  $M$ -sequence  $(X_1, Y_1, X_2, Y_2, \dots)$  of  $T - H$ , for each  $i$ ,  $\frac{|X_i|}{|X_i \cap M|} \leq 20 \log^3 k$  and  $|Y_i| \leq 10 \log^3 k$ . Further, there exists a topological sort of  $T - H$  such that the size of each boundary of any block is at most  $10 \log^3 k$  and the size of the vicinity of any block is at most  $30 \log^3 k$ .*

**Proof.** The lemma follows immediately after observing that the canonical sequence of  $T - H$  is a refinement of  $M$ -sequence of  $T - H$  and any topological sort of  $T - H$  preserves the canonical sequence of  $T - H$ .  $\blacktriangleleft$

**► Definition 29 (Back edge).** Let  $T$  be an  $M$ -consistent bipartite tournament for some  $M \subseteq V(T)$  and  $(X_1, Y_1, X_2, Y_2, \dots)$  be the  $M$ -sequence of  $T$ . An edge  $u_i u_j \in E(T)$  is called a *back edge* if  $u_i \in X_i \cup Y_i$ ,  $u_j \in X_j \cup Y_j$  and  $i - j \geq 1$ . Furthermore,  $u_i u_j$  is called *short back edge* if  $i - j = 1$  and it is called *long back edge* if  $i - j \geq 2$ .

**► Lemma 30.** *Any feedback vertex set disjoint from  $M$  must contain at least one end point of a long back edge.*

As we know that in the  $M$ -sequence of  $T - H$ , there may be back edges. Since  $T - H$  is acyclic, these edges do not participate in any cycle. We call them *simple* back edges. But, in the  $M$ -sequence of  $T - P$ , we may have back edges that form a cycle with two vertices of  $M$  and hence at least one end-point of these edges must belong to  $H$ . We call them *conflict* back edges. Hence, every back edge that is not a simple back edge is a conflict back edge. By Lemma 30, every long back edge is a conflict back edge. The  $M$ -homogeneity of  $H$  and Lemma 28 implies the following lemma.

**► Lemma 31.** *Let  $H$  be an  $M$ -homogeneous solution for  $T$ . Then, there exists a permutation of  $T - H$  such that the number of simple back edges between any consecutive blocks in the  $M$ -sequence of  $T - H$  is at most  $200 \log^6 k$ .*



Hence, if in the  $M$ -sequence of  $T - P$ , there are more than  $200 \log^6 k$  back edges between any consecutive pair of blocks, then we can branch on the choices of conflict back edges to be hit by  $H$ . The next definition and lemma captures this intuition.

► **Definition 32** (weakly-coupled). An instance  $(T, M, P, F, k)$  of CFVS is said to be weakly-coupled if in the  $M$ -sequence of  $T - P$ ,  $F$  is a subset of conflict back edges containing all long back edges such that the matching in back edges between any pair of consecutive blocks in  $T - P - F$  is at most  $201 \log^8 k$ .

Since we can find a matching in bipartite graphs in polynomial time, it can be checked in polynomial time whether a given CFVS instance  $(T, M, P, F, k)$  is weakly-coupled or not.

► **Lemma 33.** *There exists a  $\gamma$ -reduction from a CFVS instance  $(T, M, P, \emptyset, k)$  to a family  $\mathcal{C}_2 = \{(T, M, P, F_1, k), (T, M, P, F_2, k), \dots\}$  for  $\gamma = 2^{O(\frac{k}{\log k})}$  such that every instance in  $\mathcal{C}_2$  is weakly-coupled.*

► **Definition 34** (matched). An instance  $(T, M, P, F, k)$  of CFVS is said to be matched if  $F \cap E(T - P)$  forms a matching.

Note that it can be checked in polynomial time whether a given CFVS instance  $(T, M, P, F, k)$  is matched or not.

► **Lemma 35.** *There exists a  $\gamma$ -reduction from a weakly-coupled CFVS instance  $(T, M, P, F, k)$  to  $\mathcal{C}_3 := \{(T, M, P_1, F, k), (T, M, P_2, F, k), \dots\}$  for  $\gamma \leq 1.6181^k$  such that  $\mathcal{C}_3$  is weakly-coupled and matched. In addition, for each  $|P_i| = s \leq k$ ,  $\mathcal{C}_3$  has at most  $1.618^s$  CFVS instances.*

► **Definition 36** (LowBlockDegree). An instance  $(T, M, P, F, k)$  of CFVS is said to be LowBlockDegree if in the  $M$ -sequence  $(X_1, Y_1, X_2, Y_2, \dots)$  of  $T - P$ ,  $\text{long}(T, M, P) \subseteq F$  and for every set  $X_i \cup Y_i$ , at most  $201 \log^{10} k$  edges of  $F \setminus E(T - P)$  are incident on  $X_i \cup Y_i$ .

Note that it can be checked in polynomial time whether a given CFVS instance  $(T, M, P, F, k)$  is LowBlockDegree or not.

► **Definition 37** ( $X$ -preferred vertex cover). Given a bipartite graph  $G$  a set of vertices  $X \subseteq V(T)$  and a set of edges  $Q \subseteq E(G)$  such that  $Q$  is a matching in  $G$ , a minimum vertex cover  $C$  of  $Q$  is called  $X$ -vertex cover of  $Q$  if for every edge  $e \in Q$  such that  $e$  has exactly one endpoint in  $X$ ,  $C$  contains the endpoint of  $e$  in  $V(G) \setminus X$ .

Let  $T := (A, B, E)$  be a bipartite tournament and let  $X \subseteq A$ . Let  $\pi := (v_1, v_2, \dots, v_l)$  be a permutation of  $X$ . A vertex  $v \in B$  is called inconsistent with  $\pi$ , if there is no index  $i$  such that every vertex in  $\{v_1, v_2, \dots, v_i\}$  is an in-neighbor of  $v$  and every vertex in  $\{v_{i+1}, v_{i+2}, \dots, v_l\}$  is an out-neighbor of  $v$ . Given a CFVS instance  $(T, M, P, F, k)$ , a block in the  $M$ -sequence of  $T - P$  is said to have large conflict edge matching if the block is incident with at least  $201 \log^{10} k$  edges in  $F_1 := F \cap E(T - P)$ .

► **Lemma 38.** *There exists a  $\gamma$ -reduction from a weakly-coupled and matched CFVS instance  $(T, M, P, F, k)$  to  $\mathcal{C}_4 := \{(T, M, P_1, F, k), (T, M, P_2, F, k), \dots\}$  for  $\gamma = 2^{O(\frac{k}{\log k})}$  such that every instance in  $\mathcal{C}_4$  is weakly-coupled, matched and LowBlockDegree.*

**Proof.** Using  $M, P, F$ , we construct  $\mathcal{C}_4$ . Start with the  $M$ -sequence of  $T - P$ . Let  $n = |V(T)|$ ,  $t = \frac{2k}{201 \log^{10} k}$ ,  $P' := P$  and  $F' := F \cap E(T - P)$ . Branch on every family  $\mathcal{B}$  of blocks such that  $|\mathcal{B}| \leq t$ . Branch on every subset  $M'$  of size at most  $t \cdot 30 \log^3 k$ . Let  $X$  be the union of  $M$ -sub-blocks and  $Y$  be the union of  $\bar{M}$ -sub-blocks in  $\mathcal{B}$ . Add every vertex in  $Y \setminus M'$

to  $P'$ . Add every back edge neighbor of  $M'$  to  $P'$ . Branch on every permutation  $\pi$  of  $M'$ . Add every vertex of  $X \setminus M'$  not consistent with the permutation  $\pi$  to  $P'$ . Let  $E'$  be the set of back edges incident on  $X \setminus M'$ . Let  $G := (V(T), E')$ . Note that  $G$  is a bipartite graph. Branch on every minimum vertex cover of  $G$  by adding it to  $P'$ . Add a  $\bigcup \mathcal{B}$ -preferred cover of conflict edges in  $F'$  incident on  $(X \cup Y) \setminus P'$  to  $P'$ . Finally, we add a CFVS instance  $(T, M, P', F, k)$  to  $\mathcal{C}_4$  if  $(T, M, P', F, k)$  is LowBlockDegree.

*Correctness:* First we show that  $|\mathcal{C}_4| \leq \gamma$ .  $|\mathcal{C}_4|$  is bounded by the product of the number of family of blocks  $\mathcal{B}$ , the number of sets  $M'$ , the number of permutations of  $M'$  and the number of minimum vertex cover of  $G$ . The number of family of blocks  $\mathcal{B}$  is bounded by  $n^t$  as the number of blocks can be at most  $n$ . Similarly, the number of subsets  $M'$  is bounded by  $n^{t \cdot 30 \log^3 k}$ . The number of permutations is bounded by  $(t \cdot 30 \log^3 k)!$ . Since,  $(T, M, P, F, k)$  is weakly-coupled, the matching on back edges incident on any block is at most  $201 \log^8 k$ . Hence, the size of a maximum matching in  $G$  is at most  $t \cdot 201 \log^8 k$ . Hence, the number of minimal vertex cover of  $G$  is at most  $2^{t \cdot 201 \log^8 k}$ . Since,  $n = |V(T)| = O(k^3)$ , after little arithmetic manipulation, we have that  $|\mathcal{C}_4| \leq n^t \times n^{t \cdot 30 \log^3 k} \times (t \cdot 30 \log^3 k)! \times 2^{t \cdot 201 \log^8 k} = 2^{O(\frac{k}{\log k})}$ .

By the definition of the family  $\mathcal{C}_4$  and of  $\gamma$ -reduction, the backward direction is immediate. For the forward direction, let  $(T, M, P, F, k)$  be a weakly-coupled and matched CFVS instance and let  $H$  be an  $M$ -homogeneous solution of  $(T, M, P, F, k)$ . It is sufficient to show that  $\mathcal{C}_4$  has an instance  $(T, M, P', F, k)$  such that  $P' \subseteq H$ .

Consider the  $M$ -sequence of  $T - P$ . Fix a permutation  $\sigma$  of  $T - H$ . Consider a permutation  $\sigma'$  of vertices in  $T - P$  whose restriction to  $T - H$  is  $\sigma$ . Let  $\mathcal{B}$  be the family of blocks with *very large* matching in the set of conflict edges  $F'$ . Since  $|H| \leq k$ , the size of  $\mathcal{B}$  is less than  $t = \frac{2k}{201 \log^{10} k}$ . Since the size of vicinity of any block is at most  $30 \log^3 k$ , at most  $t \cdot 30 \log^3 k$  vertices form the vicinity  $M'$  of blocks in  $\mathcal{B}$ . Let  $X$  be the union of  $M$ -sub-blocks and  $Y$  be the union of  $\bar{M}$ -sub-blocks in  $\mathcal{B}$ . Then, vertices in  $Y \setminus M'$  belong to  $H$ . Since,  $M'$  is the vicinity of the blocks, every back edge incident on  $M'$  is a conflict edge. Hence, the back edge neighbor of  $M'$  belongs to  $H$ . For the same reason, the set of back edges  $E'$  incident on  $X \setminus M'$  are conflict edges and  $H$  contains a minimum cover of  $E'$ . As  $M' \cap H = \emptyset$ , any vertex inconsistent with  $\sigma_{M'}$  also belongs to  $H$ . Now, every block in  $\mathcal{B}$  is incident with conflict edges belong to  $F$  only which are disjoint, we can greedily include a vertex cover of these edges by preferring to pick the conflict edge neighbor of  $\bigcup \mathcal{B}$  into  $H$ . This implies that every block in  $\mathcal{B}$  after removing  $P'$  is not incident with any conflict edge and hence  $(T, M, P', F, k)$  is LowBlockDegree. Since  $P'$  includes all possibilities of the above choices, there is an instance  $(T, M, P', F, k)$  in  $\mathcal{C}_4$  that satisfies the required properties.  $\blacktriangleleft$

**► Definition 39 (Regular).** An instance  $(T, M, P, F, k)$  of CFVS is said to be regular if in the  $M$ -sequence  $(X_1, Y_1, X_2, Y_2, \dots)$  of  $T - P$ , for every set  $X_i$  of size at least  $10 \log^5 k$ , there are at least  $\frac{|X_i|}{10 \log^5 k}$  vertices in  $M$  and  $|Y_i| \leq 10 \log^5 k$ .

Note that it can be checked in polynomial time whether a given CFVS instance  $(T, M, P, F, k)$  is regular or not. Let  $\mathcal{L}$  be a function such that given a CFVS instance  $(T, M, P, F, k)$  outputs the family of sets of vertices which is the union of all sets  $X_i$  and  $Y_j$  in the  $M$ -sequence of  $T - P$  such that  $\frac{|X_i|}{m_i} \geq 10 \log^5 k$  where  $m_i = |X_i \cap M|$  and  $|Y_j| \geq 10 \log^5 k$ .

**► Lemma 40.** *There exists a  $\gamma$ -reduction from a CFVS instance  $(T, M, P, F, k)$  to a family  $\mathcal{C}_1 := \{(T, M, P_1, F, k), (T, M, P_2, F, k), \dots\}$  of CFVS instances for  $\gamma = 2^{O(\frac{k}{\log k})}$  such that every instance in  $\mathcal{C}_1$  is regular.*

As noted before BTFVS instance  $(T, k)$  is equivalent to CFVS instance  $(T, \emptyset, \emptyset, \emptyset, k)$ , we combine the results in the above Lemmas (abusing the notation slightly).



► **Lemma 41.** *There is a  $\gamma$ -reduction from a BTFVS instance  $(T, k)$  to a CFVS family  $\mathcal{C}'$  for  $\gamma \leq 1.6181^k$  such that every instance in  $\mathcal{C}'$  is regular, weakly-coupled, matched and LowBlockDegree. In addition, for each  $|P_2| = s \leq k$ ,  $\mathcal{C}'$  has at most  $1.618^s$  CFVS instances.*

We redefine the  $d$ -FEEDBACK VERTEX COVER defined in [27] with a slight modification. Let  $d, f$  and  $t$  be positive integers. Consider a class of mixed graphs  $\mathcal{G}(d, f, t)$  in which each member is a mixed multigraph  $\mathcal{T}$  with the vertex set  $V(\mathcal{T})$  partitioned into vertex sets  $V_1, V_2, \dots, V_t$  and an undirected edge set  $\mathcal{E}(\mathcal{T}) \subseteq \bigcup_{i < j} V_i \times V_j$  such that for each  $i \in [t]$ , (a)  $\mathcal{T}[V_i]$  is a bipartite tournament, (b) the size of the feedback vertex set  $H_i$  for  $\mathcal{T}[V_i]$  is at least  $f$  and at most  $4f$ , and (c)  $\deg_{\mathcal{E}}(V_i) \leq d$ .

Given a mixed multigraph  $\mathcal{T} \in \mathcal{G}(d, f, t)$ , a positive integer  $k$ , determine whether there exists a set  $H \subseteq V(\mathcal{T})$  such that  $|H| \leq k$  and  $\mathcal{T} - H$  contains no undirected edges and is acyclic. If  $\mathcal{E}(\mathcal{T})$  is disjoint, we call the problem as DISJOINT FEEDBACK VERTEX COVER.

► **Lemma 42.** *There exists a polynomial time algorithm that given a CFVS instance  $(T, M, P_2, F, k)$  that is regular, weakly-coupled, matched and LowBlockDegree outputs a partition  $(V_1, V_2, \dots, V_t)$  of  $V(T) \setminus P$  such that  $t \leq \frac{k}{201 \log^{12} k}$  and for each  $i \in [t]$   $V_i$  is a union of consecutive blocks in the  $M$ -sequence of  $T - P_2$  and at least one of these hold*

- *the size of feedback vertex set of  $T[V_i]$  is at least  $f = 201 \log^{12} k$  and at most  $804 \log^{12} k$ ,*
- *at least  $200 \log^{12} k$  and at most  $201 \log^{12} k$  edges in  $F \cap E(T - P_2)$  are incident on  $V_i$ .*

**Proof.** Let  $(X_1, Y_1 \dots)$  be the  $M$ -sequence of  $T - P_2$ . Consider the sequence of blocks  $(Z_1, Z_2 \dots)$  such that for each  $i$ ,  $Z_i := X_i \cup Y_i$ . Obtain the sequence  $i_1 = 1 < i_2 < \dots$  of indices such that  $V_j := \bigcup_{i=i_j}^{i_{j+1}-1} Z_i$  as follows: for each  $j$ , keep including  $Z_i$  for  $i \geq i_j$  into  $V_j$  and stop the moment at least one of the above conditions hold. To check the size of feedback vertex set in  $T[V_j]$  use the approximation algorithm in Lemma 2 i.e. check if Lemma 2 outputs a feedback vertex set for  $T[V_j]$  of size less than  $4f$ .

Since by regularity, the feedback vertex set of any block is at most  $10 \log^5 k$  and since the CFVS instance is weakly-coupled, the size of a maximum matching on back edges between any consecutive blocks is at most  $201 \log^8 k$ . Since the CFVS instance is matched and LowBlockDegree, the size of maximum matching in conflict edges is at most  $201 \log^{10} k$ . Hence, including any block into a set  $V_i$  increases the size of the feedback vertex set of  $T[V_i]$  by at most  $10 \log^5 k + 201 \log^{10} k$ . At the same time, the degree of  $V_i$  can increase by at most  $201 \log^{10} k$ . Hence, the above algorithm outputs the required partition. Note that edges in  $F \cap E(T - P_2)$  form a matching. Hence,  $t \leq \frac{k}{201 \log^{12} k}$ . ◀

► **Definition 43** (decoupled). An instance  $(T, M, P, F, k)$  of CFVS is said to be decoupled if there is a partition  $(V_1, V_2, \dots, V_t)$  of  $V(T) \setminus P$  such that  $t \leq \frac{k}{201 \log^{12} k}$  and for each  $i \in [t]$  (a)  $V_i$  is a union of consecutive blocks in the  $M$ -sequence of  $T - P$ , (b) the size of feedback vertex set of  $T[V_i]$  is at least  $f = 201 \log^{12} k$  and at most  $804 \log^{12} k$ , or at least  $200 \log^{12} k$  and at most  $d = 201 \log^{12} k$  edges in  $F \cap E(T - P)$  are incident on  $V_i$ . (c)  $F$  contains short conflict edges between any pair of sets  $V_i$  and  $V_j$ .

Note that it can be checked in polynomial time whether a given CFVS instance  $(T, M, P, F, k)$  is decoupled or not.

► **Lemma 44.** *There exists a  $\gamma$ -reduction from a regular, weakly-coupled, and matched CFVS instance  $(T, M, P_2, F, k)$  to a family  $\mathcal{C}_6$  for  $\gamma = 2^{O(\frac{k}{\log k})}$  such that every instance in  $\mathcal{C}_6$  is regular, weakly-coupled, matched, LowBlockDegree and decoupled.*

**Proof.** Given  $(T, M, P_2, F, k)$ , we construct the family  $\mathcal{C}_6$ . Using the algorithm of Lemma 42, we construct the partition  $(V_1, V_2, \dots, V_t)$  of  $V(T) \setminus P_2$ . For each  $V_i$ , let  $E_i$  be the set of back

edges incident on  $V_i$  from  $V(T) \setminus (P_2 \cup V_i)$ . Let  $J := \bigcup_{V_i} E_i$  be the union of such back edges. Now, we guess the subset  $B$  of back edges that are not hit by the required feedback vertex set. For every subset  $B \subseteq J$  of size at most  $2 \cdot t \cdot 200 \log^6 k$ , let  $J_B = J \setminus B$ . We require that the feedback vertex set hits at least one end point of every edge in  $J_B$ . Let  $D$  be the vertex cover of  $J_B$ . For every subset  $C \subseteq D$ , define  $P_C := C \cup N_{J_B}(D \setminus C)$ . For each  $P_C$ , we add the CFVC instance  $(T, M, P_3, F, k)$  where  $P_3 := P_2 \cup P_C$  into  $\mathcal{C}_6$  if  $(T, M, P_3, F, k)$  is regular, weakly-coupled, matched, LowBlockDegree and decoupled.

The backward direction is trivial. For the forward direction, let  $H$  be an  $M$ -homogeneous  $(M, P, F)$ -CFVS solution. Observe that all the above algorithm does is consider all possibilities via which  $H$  may hit the back edges between  $T[V_i \setminus P_2]$  and  $T[V_j \setminus P_2]$  for any  $i, j$ . The number of choices of sets  $B$  is at most  $(k^6)^{2 \cdot t \cdot 200 \log^6 k} = 2^{O(\frac{k}{\log k})}$ . Note that in the  $M$ -sequence of  $T - P_2$ , the matching on short back edges between any pair of consecutive blocks is at most  $201 \log^{10} k$ . Hence, the vertex cover of these back edges is at most  $201 \log^{10} k$ . Since the number of sets in the partition  $(V_1, V_2, \dots)$  is at most  $\frac{k}{f}$ , the size of the total matching on short back edges  $J$  is at most  $g = 201 \log^{10} k \times \frac{k}{f}$ . Hence, the number of choices for  $C$  is at most  $2^g = 2^{O(\frac{k}{\log k})}$ . Hence,  $\gamma = 2^{O(\frac{k}{\log k})} \times 2^{O(\frac{k}{\log k})} = 2^{O(\frac{k}{\log k})}$ .  $\blacktriangleleft$

► **Lemma 45.** *There is a polynomial time reduction from a CFVS instance  $(T, M, P_2, F, k)$  that is regular, weakly-coupled, matched, LowBlockDegree and decoupled to an instance of DISJOINT FEEDBACK VERTEX COVER  $(\mathcal{T}, k')$  for  $k' = k - |P_2|$ .*

**Proof.** Given  $(T, M, P_2, F, k)$ , construct the DFVS instance with vertex set  $V(T) \setminus (M \cup P_2)$  and make the edges in  $F \setminus E(T - P_2)$  between any two sets  $V_i$  and  $V_j$  undirected. For any solution  $H$  for  $(T, M, P_2, F, k)$ ,  $H \setminus P_2$  is a feedback vertex set of  $T - P_2$  that hits  $F \setminus E(T - P_2)$ . Hence,  $H \setminus P_2$  is a feedback vertex cover for  $(\mathcal{T}, k')$  for  $k' = k - |P_2|$ . In the backward direction, a solution  $S$  for  $(\mathcal{T}, k')$  hits  $F \setminus E(T - P_2)$  and is disjoint from  $M$ . Hence,  $S \cup P_2$  is a solution for  $(T, M, P_2, F, k)$ .  $\blacktriangleleft$

At this point, we can use the following lemma from [27] with the only difference being in the base case as we have a bipartite tournament instead of a *supertournament*. We replace the naive  $3^k$  algorithm by  $4^k$  algorithm to find a feedback vertex for each of  $T[V_i]$ . Note that bounding the size of feedback vertex set in each of  $T[V_i]$  to  $O(\log^{12} k)$  and the number of  $V_i$ 's to at most  $O(\frac{k}{\log^{12} k})$  implies that the maximum time spent in solving the base cases is at most  $O(\frac{k}{\log^{12} k}) \cdot 2^{O(\log^{12} k)}$ .

► **Lemma 46** ([27]). *There exists an algorithm running in  $1.5874^s \cdot 2^{O(df \log k + d \log s)} \cdot n^{O(1)}$  time which finds an optimal feedback vertex cover in a mixed multigraph  $\mathcal{T} \in \mathcal{G}(d, f, t)$  in which the undirected edge set  $\mathcal{E}(\mathcal{T})$  is disjoint and  $|\mathcal{E}(\mathcal{T})| = s$ .*

► **Theorem 47.** *There exists an algorithm for BTFVS running in  $1.6181^k + n^{O(1)}$  time.*

**Proof.** Using the algorithm of Lemma 41, construct the family  $\mathcal{C}_5$  of CFVS instances. For each instance  $(T, M, P_2, F, k) \in \mathcal{C}_5$ , using the algorithm of Lemma 44 construct the family  $\mathcal{C}_6$  of CFVS instances. Then for each CFVS instance  $(T, M, P, F, k)$  using Lemma 45, construct the DFVC instance  $(\mathcal{T}, k - |P|)$  which is solved using the algorithm of Lemma 46. If for any instance, the algorithm of Lemma 46 outputs a solution set  $S$  of size at most  $k - |P|$ , then we output YES, otherwise output NO.

The correctness of the algorithm follows from the correctness of the algorithms in the Lemma 41, 44, 45 and 46. The runtime of the algorithm is upper bounded by  $\sum_{s=1}^k 1.618^{k-s} \times 1.5874^s \cdot 2^{O(df \log k + d \log s)} \cdot n^{O(1)} \leq 1.6181^k \cdot 2^{O(d^2 + d \log k)} \cdot n^{O(1)}$ .  $\blacktriangleleft$

► **Proposition 48** ([16]). *If there exists a parameterized algorithm for any vertex deletion problem into a hereditary graph class with running time  $c^k n^{O(1)}$ , then there exists an exact-exponential-time algorithm for the problem with running time  $(2 - \frac{1}{c})^{n+o(n)} n^{O(1)}$ .*

The above proposition immediately implies the following theorem.

► **Theorem 49.** *There exists an algorithm for BTFVS running in  $1.3820^n$  time.*

---

## References

- 1 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.
- 2 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.*, 12(3):289–297, 1999.
- 3 Lowell W. Beineke and Charles H. C. Little. Cycles in bipartite tournaments. *Journal of Combinatorial Theory, Series B*, 32(2):140–145, 1982.
- 4 Mao-cheng Cai, Xiaotie Deng, and Wenan Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM J. Comput.*, 30(6):1993–2007, 2000.
- 5 Mao-cheng Cai, Xiaotie Deng, and Wenan Zang. A min-max theorem on feedback vertex sets. *Math. Oper. Res.*, 27(2):361–371, 2002. doi:10.1287/moor.27.2.361.328.
- 6 Jianer Chen, Yang Liu, Songjian Lu, Barry O’Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008. doi:10.1145/1411509.1411511.
- 7 C. R. J. Clapham. The bipartite tournament associated with a fabric. *Discrete Mathematics*, 57(1):195–197, 1985.
- 8 Vincent Cohen-Addad, Éric Colin de Verdière, Philip N. Klein, Claire Mathieu, and David Meierfrankenfeld. Approximating connectivity domination in weighted bounded-genus graphs. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 584–597, 2016.
- 9 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159, 2011.
- 10 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.
- 11 Erik D. Demaine and Mohammad Taghi Hajiaghayi. Bidimensionality: new connections between FPT algorithms and ptass. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 590–601, 2005.
- 12 Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010.
- 13 P. Erdős and L. Pósa. On independent circuits contained in a graph. *Canad. J. Math*, 17:347–352, 1965.
- 14 Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.
- 15 Paola Festa, Panos M. Pardalos, and Mauricio G. C. Resende. Feedback set problems. In *Handbook of combinatorial optimization*, pages 209–258. Springer, 1999.

- 16 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 764–775, 2016. doi:10.1145/2897518.2897551.
- 17 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Bidimensionality and EPTAS. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 748–759, 2011.
- 18 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 503–510, 2010.
- 19 Fedor V. Fomin and Yngve Villanger. Finding induced subgraphs via minimal triangulations. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, pages 383–394, 2010.
- 20 Leo Moser Frank Harary. The theory of round robin tournaments. *The American Mathematical Monthly*, 73(3):231–246, 1966. URL: <http://www.jstor.org/stable/2315334>.
- 21 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman and Co., 1979.
- 22 Venkatesan Guruswami, Johan Håstad, Rajsekar Manokaran, Prasad Raghavendra, and Moses Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM J. Comput.*, 40(3):878–914, 2011.
- 23 Gregory Gutin and Anders Yeo. Some parameterized problems on digraphs. *Comput. J.*, 51(3):363–371, 2008.
- 24 Sheng-Ying Hsiao. Fixed-parameter complexity of feedback vertex set in bipartite tournaments. In *Algorithms and Computation – 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, pages 344–353, 2011. doi:10.1007/978-3-642-25591-5\_36.
- 25 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pages 85–103, 1972.
- 26 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014.
- 27 Mithilesh Kumar and Daniel Lokshtanov. Faster exact and parameterized algorithm for feedback vertex set in tournaments. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 49:1–49:13, 2016. doi:10.4230/LIPIcs.STACS.2016.49.
- 28 Igor Razgon. Computing minimum directed feedback vertex set in  $o(1.9977^{\Omega})$ . In *Theoretical Computer Science, 10th Italian Conference, ICTCS 2007, Rome, Italy, October 3-5, 2007, Proceedings*, pages 70–81, 2007.
- 29 Bruce Reed, Neil Robertson, Paul Seymour, and Robin Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996.
- 30 Prashant Sasatte. Improved FPT algorithm for feedback vertex set problem in bipartite tournament. *Inf. Process. Lett.*, 105(3):79–82, 2008. doi:10.1016/j.ipl.2007.08.014.
- 31 A. Truß. *Parameterized algorithms for feedback set problems in tournaments*. PhD thesis, Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2005.
- 32 Shuichi Ueno, Yoji Kajitani, and Shin'ya Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1-3):355–360, 1988.

- 33 Anke van Zuylen. Linear programming based approximation algorithms for feedback set problems in bipartite tournaments. *Theor. Comput. Sci.*, 412(23):2556–2561, 2011. doi: 10.1016/j.tcs.2010.10.047.
- 34 Mingyu Xiao and Hiroshi Nagamochi. An improved exact algorithm for undirected feedback vertex set. *J. Comb. Optim.*, 30(2):214–241, 2015.



# Probabilistic Mu-Calculus: Decidability and Complete Axiomatization\*

Kim G. Larsen<sup>1</sup>, Radu Mardare<sup>2</sup>, and Bingtian Xue<sup>3</sup>

- 1 Aalborg University, Denmark  
kgl@cs.aau.dk
- 2 Aalborg University, Denmark  
mardare@cs.aau.dk
- 3 Aalborg University, Denmark  
bingt@cs.aau.dk

---

## Abstract

We introduce a version of the probabilistic mu-calculus (PMC) built on top of a probabilistic modal logic that allows encoding  $n$ -ary inequational conditions on transition probabilities. PMC extends previously studied calculi and we prove that, despite its expressiveness, it enjoys a series of good meta-properties. Firstly, we prove the decidability of satisfiability checking by establishing the small model property. An algorithm for deciding the satisfiability problem is developed. As a second major result, we provide a complete axiomatization for the alternation-free fragment of PMC. The completeness proof is innovative in many aspects combining various techniques from topology and model theory.

**1998 ACM Subject Classification** F.4.1 Mathematical Logic

**Keywords and phrases** Markov process, probabilistic modal  $\mu$ -calculus,  $n$ -ary (in-)equational modalities, satisfiability, axiomatization

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.25

## 1 Introduction

From the perspective of industrial practice, especially in the area of embedded and cyber-physical systems, an essential problem is how to deal with the high complexity of the systems, while still meeting the requirements of correctness, predictability, performance and also non-functional properties. In this respect, for embedded systems, specification and verification should not only consider functional properties but also non-functional properties. Particularly, effort has been put into formalisms and logics that address stochastic aspects of a system. The seminal work of Hansson and Jonsson [13] introduced pCTL, a probabilistic extension of CTL. In a number of recent work results related to decidability and complexity of model checking and satisfiability checking of (variants of) pCTL have been established [1, 14, 4, 22, 24, 5].

In parallel, various probabilistic modal  $\mu$ -calculi have been considered. Typically, one characterizes the probabilistic bisimulation by using a probabilistic version of modal logic with the modality indexed by a subunital positive real: e.g.  $\langle \rangle_{>p}\phi$  describes that the probability of reaching a next state satisfying  $\phi$  is greater than  $p$ . Whereas the resulting logic does fully characterize probabilistic bisimulation, it is not sufficiently expressive with respect to

---

\* This work was partially supported by the Sino-Danish Basic Research Center IDEA4CPS (grant number: 26-333-86-7/JM).



decomposition of properties under static operators. To address this, in [21] an extended  $n$ -ary next-state modality (*(in-)equational modality*) was introduced: e.g.  $[\langle x \rangle \phi_1, \langle y \rangle \phi_2 : x + y \leq 0.7]$  describes that the probabilities  $x$  and  $y$  of reaching next-states satisfying  $\phi_1$  and  $\phi_2$  respectively must satisfy the constraint  $x + y \leq 0.7$ . This modality allows one to encode complex linear constraints on probabilities.

In this paper we introduce a probabilistic  $\mu$ -calculus (PMC) for specifying and reasoning about the Markov processes. PMC extends with block sequences (equation systems) the modal logic of [21]. As a first main result, we prove the decidability of satisfiability checking by establishing a small model property for this logic. As a second main result, we provide a sound and complete axiomatization for the alternation-free fragment of PMC.

**Related Work.** The satisfiability problem for the probabilistic logics with fixed points has been a hot topic for a number of years. While this is still an open problem for pCTL and pCTL\*, various fragments have been solved. In [14, 4], it is shown that qualitative pCTL (expressing only whether a probability is bigger than 0 or equal to 1) has no finite model property and its satisfiability problem is ExpTime-complete. Moreover, it is proven that satisfiability checking for pCTL against models with bounded branching degree is highly undecidable; however, every satisfiable formula has a model with branching degree bounded by the size of the formula. More recently, in [1], pCTL satisfiability problem for bounded-size models is studied and proved to be decidable. In [22, 24], the qualitative fragment of pCTL\* is proved to be decidable too. In recent works [23, 6], the satisfiability problem for an extension of the logic in [20] with fixed points is proven to be decidable. This logic only involves probabilistic next-state operator and it cannot express the (in-)equational modalities of [21].

The decidability of probabilistic  $\mu$ -calculus of [23] also derives as a particular case of the more general results proven in [7], where it is shown that the decidability of coalgebraic mu-calculi parametrized by a tractable set of so-called one-step rules is in ExpTime; in [17] such a rule set has been exhibited for probabilistic modal logic with linear inequalities. However, all these works were done only for finite sets with discrete probability distributions.

In [26, 27, 25] probabilistic modal  $\mu$ -calculus, Łukasiewicz  $\mu$ -calculus, probabilistic modal  $\mu$ -calculus with independent product are studied in the context of denotational semantics and game semantics, relying on a satisfiability relation that is not essentially boolean but rather quantitative.

Another fixed points probabilistic logic is proposed in [8]. Its syntax is divided into a probabilistic part (so called state formulas) and a non-probabilistic part involving fixed points (so called fuzzy formulas). This logic can encode the probabilistic modal logic and pCTL\* and it is studied from the perspective of (finite) model checking and bisimulation checking.

Considering the axiomatization, complete axiomatizations for the qualitative fragment of pCTL\* are shown in [22], but only for bounded finite systems.

**Our Work.** With respect to the related work described above, our probabilistic  $\mu$ -calculus involves the equational modalities of [21], thus allowing us to encode (in-)equational conditions on probabilities. The logic is definitely more general than that of [23]. The semantics, with respect to the other related works such as [7, 17] is in term of general (analytical) measurable sets and not just finite spaces with discrete sigma-algebras; it has been repeatedly proven in literature, see e.g. [28], that going from discrete systems to continuous systems is far from being a trivial step, and complex topological and measure theoretical arguments applied to model theory must be invoked.



Our logic is incomparable with pCTL and pCTL\* as it cannot express modalities such as “probabilistic Until”. The work here includes a modality extension but does not try to add this modality to the more complicated logics of [5, 26, 27, 25]. However, our logic can be used to approximate pCTL formulas with arbitrary precision when restricting to finite models. This is interesting as the satisfiability problem for quantitative pCTL is still open.

We prove that this logic enjoys the finite model property and its satisfiability problem is decidable. We develop an algorithm that checks the satisfiability of a formula and, if the formula is satisfiable, it constructs a finite model. Being the aforementioned state of the art in the field, these are important results presenting our logic as a good trade-off between expressiveness and decidability. Moreover, these results generalize the ones in [23] while our proof constructs on top of the classic tableau method [33, 15, 34].

Another key contribution of our paper is the complete axiomatization that we propose for the alternation-free fragment of PMC. At the best of our knowledge, the problem of axiomatizing probabilistic  $\mu$ -calculus has not been previously approached at this level of generality. The completeness proof is a non-standard extension of the filtration method relying on topological facts such as Rasiowa-Sikorski lemma and its relation to Lindenbaum’s lemma (following the technique developed by the first two authors in collaboration with Dexter Kozen and Prakash Panangaden [16]). The proof also applies the technique developed in [18] by the authors for proving the completeness of fixed points logics. These can be easily adapted to other versions of probabilistic  $\mu$ -calculus.

Due to space limit, most of the results stated here are without proofs. For a detailed presentation with the proofs and some of the classical definitions and lemmas, the reader is referred to <http://people.cs.aau.dk/~bingt/probaMuCalc.pdf>

## 2 Probabilistic Mu-Calculus

Probabilistic  $\mu$ -Calculus (PMC) that we develop in this paper encodes properties of Markov processes. As usual with  $\mu$ -Calculus based on equation systems, the syntax is given in two stages: we firstly introduce the basic formulas and secondly use them to define blocks. The basic formulas are boolean formulas, constructed on top of a set  $\mathcal{A}$  of atomic propositions and involving the following:

- *recursive-variables* range over the set  $\mathcal{X}$ ; they are used to define simultaneous recursive equations in order to express maximal and minimal fixed points, in the style of [19, 9, 10, 2];
- *(in-)equational modalities* of type  $\langle x_1 \rangle \phi_1, \dots, \langle x_n \rangle \phi_n : \sum_{i=1}^n a_i x_i \geq r$  where  $x_1, \dots, x_n$  are *probability variables* ranging over a set  $\mathcal{V}$  and  $a_1, \dots, a_n, r \in \mathbb{Q}$ .

► **Definition 1** (Basic formulas). The *basic formulas* of PMC are defined by the following grammar, for arbitrary  $p \in \mathcal{A}$ ,  $X \in \mathcal{X}$ ,  $a_1, \dots, a_n, r \in \mathbb{Q}$ ,  $x_1, \dots, x_n \in \mathcal{V}$ :

$$\mathcal{L} : \quad \phi := p \mid \neg \phi \mid \phi \vee \phi \mid \langle x_1 \rangle \phi_1, \dots, \langle x_n \rangle \phi_n : \sum_{i=1}^n a_i x_i \geq r \mid X .$$

**Notation:** For arbitrary  $\bar{x} \in \mathcal{V}^n$ ,  $\bar{a} \in \mathbb{Q}^n$ ,  $r \in \mathbb{Q}$  and  $\bar{\phi} \in \mathcal{L}^n$ , instead of  $\langle x_1 \rangle \phi_1 \dots \langle x_n \rangle \phi_n$ , we simply write  $\overline{\langle x \rangle \phi}$  and instead of  $\sum_{i=1}^n a_i x_i \geq r$  we write  $\bar{a} \cdot \bar{x} \geq r$ . This will simplify the syntax of the equational modalities and instead of  $\langle x_1 \rangle \phi_1 \dots \langle x_n \rangle \phi_n : \sum_{i=1}^n a_i x_i \geq r$ , we will write  $\overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \geq r$ . In this case,  $n$  is called the *length* of  $\overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \geq r$ . If  $\overline{\langle x \rangle \phi} = \langle x_1 \rangle \phi_1 \dots \langle x_n \rangle \phi_n$  and  $\bar{a} \cdot \bar{x} = \sum_{i=1}^n a_i x_i$ , for  $k < n$ , let  $\overline{\langle x \rangle \phi} \Big|_k \stackrel{\text{def}}{=} \langle x_1 \rangle \phi_1 \dots \langle x_k \rangle \phi_k$  and  $\bar{a} \cdot \bar{x} \Big|_k \stackrel{\text{def}}{=} \sum_{i=1}^k a_i x_i$ .

Observe that in the basic formulas we only allow one inequality using  $\geq$  to specify the constraints on  $\bar{x}$ . However, we can, for instance, encode reversed inequalities since we are

using all rationals; and we can encode a finite set of constraints by involving conjunctions of the equational modalities.

The dual of  $\overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \geq r$  can be defined as  $\overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} < r$ ; for this reason, we write constraints freely using  $\geq, \leq, >$  or  $<$ . We use both  $\sqtriangleleft$  and  $\sqtriangleright$  to range over the set  $\{\leq, \geq\}$  such that  $\{\sqtriangleleft, \sqtriangleright\} = \{\leq, \geq\}$ . Similarly, we use  $\triangleleft$  and  $\triangleangleright$  to range over the set  $\{<, >\}$  such that  $\{\triangleleft, \triangleangleright\} = \{<, >\}$ .

Now we introduce the equation blocks. Given  $\phi, \psi_1, \dots, \psi_h \in \mathcal{L}$  and  $X_1, \dots, X_h \in \mathcal{X}$ , let  $\phi\{\psi_1/X_1, \dots, \psi_h/X_h\}$  be the formula obtained by substituting each occurrence of  $X_i$  in  $\phi$  with  $\psi_i$  for  $i = 1, \dots, h$ ; denoted shortly  $\phi\{\bar{\psi}/\bar{X}\}$ , where  $\bar{\psi} = (\psi_1, \dots, \psi_h)$  and  $\bar{X} = (X_1, \dots, X_h)$ . Following [9, 10, 2], we allow sets of the maximal or minimal *blocks* of mutually recursive equations in PMC.

► **Definition 2** (Equation Blocks). An equation block  $B$  over the set  $\mathcal{X}_B = \{X_1, \dots, X_N\} \subseteq \mathcal{X}$  of pairwise distinct variables has one of two forms –  $\min\{E\}$  or  $\max\{E\}$ , where  $E$  is a system of (mutually recursive) equations such that for any  $i, j \in \{1, \dots, N\}$ ,  $\phi_i$  is monotonic in  $X_j$ .

$$E : \quad \langle X_1 = \phi_1, \dots, X_N = \phi_N \rangle.$$

If  $B = \max\{E\}$  or  $B = \min\{E\}$ , the elements of  $\mathcal{X}_B$  are called max-variables or min-variables respectively. Given the system  $E$  of equations in the previous definition, its *dual* is

$$\tilde{E} : \quad \langle X_1 = \neg\phi_1\{\neg X_1/X_1, \dots, \neg X_N/X_N\}, \dots, X_N = \neg\phi_N\{\neg X_1/X_1, \dots, \neg X_N/X_N\} \rangle.$$

If  $B = \max\{E\}$  or  $B = \min\{E\}$ , then its *dual* is  $\tilde{B} = \min\{\tilde{E}\}$  or  $\tilde{B} = \max\{\tilde{E}\}$  respectively.

We say that a formula  $\phi \in \mathcal{L}$  *depends on*  $B$  if it involves variables in  $\mathcal{X}_B$ . If  $\mathcal{X}_B \cap \mathcal{X}_{B'} = \emptyset$ , we say that  $B$  is *dependent on*  $B'$  if the right hand side formulas of the equations in  $B$  depend on  $B'$ .

► **Definition 3** (Block Sequence). A sequence  $\mathcal{B} = B_1, \dots, B_m$  of  $m \geq 1$  pairwise-distinct equation blocks is a *block sequence* if  $\mathcal{X}_{B_i} \cap \mathcal{X}_{B_j} = \emptyset$  for  $i \neq j$ . A block sequence  $\mathcal{B} = B_1, \dots, B_m$  of  $m \geq 1$  is called *alternation-free* if  $B_i$  is not dependent on  $B_j$  whenever  $i < j$ .

A formula  $\phi \in \mathcal{L}$  is *dependent on*  $\mathcal{B}$  if it is dependent of each block in the sequence.

The semantics of our calculus is defined in terms of (probabilistic) Markov processes [28].

► **Definition 4** (Markov Process). A (*probabilistic*) *Markov process* (PMP) is a tuple  $\mathcal{M} = (M, \Sigma, l, \theta)$  with  $(M, \Sigma)$  an analytic measurable space<sup>1</sup> of states,  $l : M \rightarrow 2^A$  a labeling function associating a set of state labels (i.e., atomic propositions) to each state and  $\theta : M \rightarrow \Pi(M, \Sigma)$  the transition function associating a probability measure over  $(M, \Sigma)$  to each state.

Given a PMP  $\mathcal{M} = (M, \Sigma, l, \theta)$ , an *environment* is a function  $\rho : \mathcal{X} \rightarrow 2^M$  that interprets the recursive-variables as sets of states. We use  $\emptyset$  as the empty environment that associates  $\emptyset$  to all recursive-variables. Given an environment  $\rho$  and  $S \subseteq M$ , let  $\rho[X \mapsto S]$  be the environment that interprets  $X$  as  $S$  and all the other recursive-variables as  $\rho$  does. Similarly, for a pairwise-disjoint tuple  $\bar{X} = (X_1, \dots, X_N) \in \mathcal{X}^N$  and  $\bar{S} = (S_1, \dots, S_N) \subseteq M^N$ , let  $\rho[\bar{X} \mapsto \bar{S}]$  be the environment that interprets  $X_i$  as  $S_i$  for all  $i = 1, \dots, N$  and all the other variables as  $\rho$  does.

<sup>1</sup> An analytic space<sup>1</sup> is a continuous image of a Polish space in a Polish space; a Polish space is the topological space underlying a complete separable metric space.

Given a PMP  $\mathcal{M} = (M, \Sigma, l, \theta)$  and an environment  $\rho$ , the semantics for the basic formulas in  $\mathcal{L}$  is defined, on top of the classic semantics for Boolean logic, inductively as follows,

$$\begin{aligned} \mathcal{M}, m, \rho &\models p \text{ iff } p \in l(m); \\ \mathcal{M}, m, \rho &\models \neg\phi \text{ iff } \mathcal{M}, m, \rho \not\models \phi; \\ \mathcal{M}, m, \rho &\models \phi_1 \vee \phi_2 \text{ iff } \mathcal{M}, m, \rho \models \phi_1 \text{ or } \mathcal{M}, m, \rho \models \phi_2; \\ \mathcal{M}, m, \rho &\models X \text{ iff } m \in \rho(X); \\ \mathcal{M}, m, \rho &\models \overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \geq r \text{ iff } \sum_{i=1}^n a_i \theta(m)(\llbracket \phi_i \rrbracket_{\rho}^{\mathcal{M}}) \geq r, \end{aligned}$$

where  $\llbracket \phi \rrbracket_{\rho}^{\mathcal{M}} = \{m \in M \mid \mathcal{M}, m, \rho \models \phi\}$ .

Following [19, 9, 10, 2], we extend now the semantics to include the restrictions imposed by a sequence of blocks and obtain the so-called block-semantics.

Given a set of equations  $E$  with  $\bar{X} = (X_1, \dots, X_N)$ , an environment  $\rho$  and  $\bar{\Upsilon} = (\Upsilon_1, \dots, \Upsilon_N) \subseteq 2^M$ , let the function  $f_E^{\rho} : (2^M)^N \rightarrow (2^M)^N$  be defined as:  $f_E^{\rho}(\bar{\Upsilon}) = \langle \llbracket \phi_1 \rrbracket_{\rho[\bar{X} \mapsto \bar{\Upsilon}]}, \dots, \llbracket \phi_N \rrbracket_{\rho[\bar{X} \mapsto \bar{\Upsilon}]} \rangle$ .

Observe that  $(2^M)^N$  forms a complete lattice with the ordering, join and meet operations defined as the point-wise extensions of the set-theoretic inclusion, union and intersection, respectively. Moreover, for any  $E$  and  $\rho$ ,  $f_E^{\rho}$  is monotonic with respect to the order of the lattice and therefore, it has a greatest fixed point denoted by  $\nu \bar{X}. f_E^{\rho}$  and a least fixed point denoted by  $\mu \bar{X}. f_E^{\rho}$  [9]. These can be characterized as:  $\nu \bar{X}. f_E^{\rho} = \bigcup \{ \bar{\Upsilon} \mid \bar{\Upsilon} \subseteq f_E^{\rho}(\bar{\Upsilon}) \}$ ,  $\mu \bar{X}. f_E^{\rho} = \bigcap \{ \bar{\Upsilon} \mid f_E^{\rho}(\bar{\Upsilon}) \subseteq \bar{\Upsilon} \}$ .

The blocks  $\max\{E\}$  and  $\min\{E\}$  define environments that satisfy all the equations in  $E$ ;  $\max\{E\}$  is the greatest fixed point and  $\min\{E\}$  is the least fixed point. The environment defined by the block  $B$  is denoted by  $\llbracket B \rrbracket_{\rho}$ . Given a block sequence  $\mathcal{B} = B_1, \dots, B_m$  and an environment  $\rho_0$ , let  $\rho_1, \dots, \rho_m$  be defined by  $\rho_i = \llbracket B_i \rrbracket_{\rho_{i-1}}$  for  $i = 1, \dots, m$ . The semantics of  $\mathcal{B}$  is then given by

$$\llbracket \mathcal{B} \rrbracket_{\rho_0} = \rho_m.$$

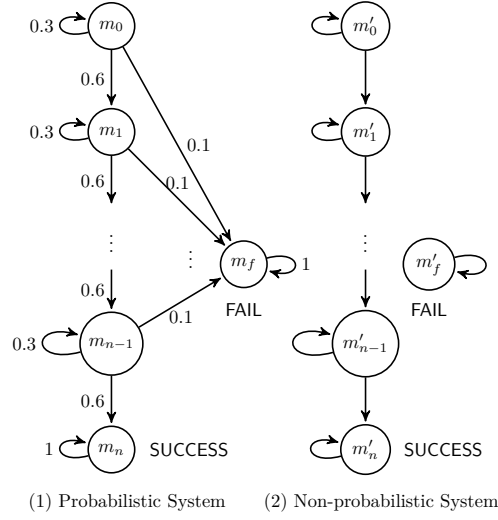
► **Definition 5** (Block-Semantics). Given a block sequence  $\mathcal{B}$ , the  $\mathcal{B}$ -semantics of a formula  $\phi \in \mathcal{L}$  that depends on  $\mathcal{B}$  is given for a PMP  $\mathcal{M} = (M, \Sigma, l, \theta)$  with  $m \in M$  and an environment  $\rho$ , as follows,

$$\mathcal{M}, m, \rho \models_{\mathcal{B}} \phi \text{ iff } \mathcal{M}, m, \llbracket \mathcal{B} \rrbracket_{\rho} \models \phi.$$

We say that a formula  $\phi$  is  $\mathcal{B}$ -satisfiable if there exists at least one PMP that satisfies it for the block sequence  $\mathcal{B}$  in one of its states under some environment;  $\phi$  is a  $\mathcal{B}$ -validity, written  $\models_{\mathcal{B}} \phi$ , if it is satisfied for  $\mathcal{B}$  in all states of any PMP under any environment.

► **Example 6.** Suppose a file is divided into  $n$  blocks that are distributed among several peers in a peer-to-peer network. When a user wants to get the complete file from the network, he needs to download all  $n$  blocks. When the user tries to download a block, there are three possibilities: (1) he gets the block successfully and he will try to download the next block (with probability 0.6); (2) the block is not available anymore, in which case it is not possible to get the complete file (with probability 0.1); (3) the peer did not response within a time limit and the user retries (with probability 0.3). To simplify the example, we assume that only one block can be downloaded at one time. The system (1) in Figure 1 is one of this type.

Consider the safety property that “it will never FAIL to get the file” as shown in the system (2) in Figure 1. In the non-probabilistic case, this can be specified by the *mu*-calculus



■ **Figure 1** Peer-to-peer file sharing network.

formula  $\phi$ :

$$\phi = \text{SUCCESS} \vee X$$

$$B = \max \left\{ X = \begin{array}{l} \neg \text{FAIL} \wedge \neg \text{SUCCESS} \\ \wedge (\langle \rangle \text{SUCCESS} \vee \langle \rangle X) \end{array} \right\},$$

where  $\phi$  is satisfied by  $m'_0, \dots, m'_n$  and  $X$  is satisfied by  $m'_0, \dots, m'_{n-1}$ .

Consider the probabilistic safety property that “at any moment, the probability of FAIL to get the file is less than or equal to 0.1”. This requirements can be expressed in PMC as:

$$\phi = \text{SUCCESS} \vee X$$

$$B = \max \left\{ X = \begin{array}{l} \neg \text{FAIL} \wedge \neg \text{SUCCESS} \wedge \\ (\langle x_1 \rangle \text{SUCCESS}, \langle x_2 \rangle X : x_1 + x_2 \geq 0.9) \end{array} \right\},$$

where  $\phi$  is satisfied by  $m'_0, \dots, m'_n$  and  $X$  by  $m'_0, \dots, m'_{n-1}$ . Notice that, they still hold when the system is infinite, i.e.,  $n$  goes to  $+\infty$ .

### 3 Decidability and finite model property

In this section, we prove that the  $\mathcal{B}$ -satisfiability problem of PMC is decidable, i.e., it is decidable whether a given formula  $\phi$  of PMC which is closed w.r.t. a block sequence  $\mathcal{B}$  is satisfiable. We do this by involving the tableau construction [33, 15, 34] that will eventually help us constructing a model for  $\phi$ . We show that PMC enjoys the finite model property and present a decision procedure. This work is done for the entire PMC and not only for the alternation-free fragment.

Given a formula  $\phi$  dependent on  $\mathcal{B}$ , the construction of the model follows 4 steps (in brief):

1. Find the so-called co-prime formula  $\phi^c$  for  $\phi$ , which has a special format and admits the same models as  $\phi$ . Similarly, we construct a co-prime block sequence  $\mathcal{B}^c$ . Both  $\phi^c$  and  $\mathcal{B}^c$  only involve integer inequalities with co-prime coefficients in the equational modalities.
2. Construct a set of formulas which is vital in constructing the tableau for  $\phi^c$ . In contrast to that of the classical  $\mu$ -calculus, this set not only contains the subformulas of  $\phi^c$  and  $\mathcal{B}^c$

but also it is still a finite set of formulas. This construction involves complex continuity arguments on rationals and this makes it particularly different from any similar techniques used previously with  $\mu$ -calculi. The basic idea behind it is that every rational inequality (system) has (at least) one rational solution.

3. Construct a tableau for  $\phi^c$  by adapting the classical tableau method. The key here is to use maximal sets as nodes, in order to get the probability distributions over the state space.
4. The tableau provides a PMP, which is also a model for  $\phi^c$ , hence also for  $\phi$ .

► **Definition 7 (Co-Prime).** A block sequence  $\mathcal{B}$  (a formula  $\phi \in \mathcal{L}$  dependent on  $\mathcal{B}$ ) is said to be *co-prime* iff for any  $\langle x \rangle \psi: \bar{a} \cdot \bar{x} \geq r$  that appears in  $\mathcal{B}$  (in  $\phi$  or  $\mathcal{B}$ ),  $a_1, \dots, a_n$  are co-prime integers.

For any inequality  $\sum_{i=1}^n a_i x_i \geq r$ , one can divide both sides of the inequality by the greatest common divisor of  $a_1, \dots, a_n$  to get an inequality that has the same solution. Hence, for any block sequence (formula), one can get its *co-prime block sequence (co-prime formula)* by changing all inequalities in it by the above mentioned method.

#### Properties:

1. For any block sequence  $\mathcal{B}$ , there exists a unique co-prime block sequence denoted by  $\mathcal{B}^c$ ; for any formula  $\phi$ , there exists a unique co-prime formula denoted by  $\phi^c$ .
2. For any formula  $\langle x \rangle \phi: ax \geq r$ ,  $(\langle x \rangle \phi: ax \geq r)^c \in \{ \langle x \rangle \phi: x \leq \frac{r}{a}, \langle x \rangle \phi: x \geq \frac{r}{a} \}$ .

► **Proposition 8.** For any  $\phi \in \mathcal{L}$  dependent on  $\mathcal{B}$  and its co-prime formula  $\phi^c$ , any model satisfying one also satisfies the other, i.e., for any PMP  $\mathcal{M} = (M, \Sigma, l, \theta)$  with  $m \in M$  and any environment  $\rho$ ,

$$\mathcal{M}, m, \rho \models_{\mathcal{B}} \phi \text{ iff } \mathcal{M}, m, \rho \models_{\mathcal{B}^c} \phi^c.$$

Therefore, for solving the satisfiability problem of a formula, it is sufficient to solve the satisfiability problem of its co-prime formula.

Consider  $\phi \in \mathcal{L}$  dependent on  $\mathcal{B}$ . The set of all the recursive-variables in  $\phi$  and  $\mathcal{B}$  is denoted  $\mathcal{X}[\phi, \mathcal{B}]$ . Let  $R[\phi, \mathcal{B}] \subseteq \mathbb{Q}$  be the set of all rationals in  $\phi$  or  $\mathcal{B}$ ; let  $R^*[\phi, \mathcal{B}] \subseteq \mathbb{Q}$  be the set of all  $\frac{r}{a_i}$  s.t.  $\langle x \rangle \psi: (a_1, \dots, a_n) \cdot \bar{x} \geq r$  appears in  $\phi$  or  $\mathcal{B}$  and  $a_i \neq 0$ . Obviously,  $R[\phi, \mathcal{B}]$  and  $R^*[\phi, \mathcal{B}]$  are both finite.

- The *granularity* of  $\phi$  dependent on  $\mathcal{B}$ , denoted by  $gr(\phi, \mathcal{B})$ , is the least common denominator of the elements of  $R^*[\phi, \mathcal{B}]$ . Let  $I[\phi, \mathcal{B}]$  be the set of all rationals of type  $\frac{p}{gr(\phi, \mathcal{B})}$  in the interval  $[\min(R^*[\phi, \mathcal{B}]), \max(R^*[\phi, \mathcal{B}])]$ , for  $p \in \mathbb{Z}$ . Notice that  $I[\phi, \mathcal{B}] = \emptyset$  whenever  $R^*[\phi, \mathcal{B}] = \emptyset$ .
- The *modal depth* of  $\phi$  dependent on  $\mathcal{B}$ , denoted by  $md(\phi, \mathcal{B})$ , is defined inductively by

$$md(\phi, \mathcal{B}) = \begin{cases} 0, & \text{if } \phi = p \text{ or } \phi = X \\ md(\psi, \mathcal{B}), & \text{if } \phi = \neg\psi \\ \max\{md(\psi), md(\psi')\}, & \text{if } \phi = \psi \vee \psi' \\ \max\{md(\psi_i) \mid i = 1, \dots, n\} + 1, & \text{if } \phi = \langle x \rangle \psi: \bar{a} \cdot \bar{x} \geq r \end{cases}$$

- The *modality length* of  $\phi$  dependent on  $\mathcal{B}$ , denoted by  $ml(\phi, \mathcal{B})$ , is largest length of the sub-formula  $\langle x \rangle \psi: \bar{a} \cdot \bar{x} \geq r$  that appears in  $\phi$  or  $\mathcal{B}$ .

In the following, we fix a co-prime formula  $\phi^c \in \mathcal{L}$  dependent on a co-prime block sequence  $\mathcal{B}^c$  and we construct a model for it. Let

$$(\otimes) \quad \begin{aligned} \mathcal{L}[\phi^c, \mathcal{B}^c] &= \{ \phi \in \mathcal{L} \mid \mathcal{X}[\phi, \mathcal{B}^c] \subseteq \mathcal{X}[\phi^c, \mathcal{B}^c], R[\phi, \mathcal{B}] \subseteq R[\phi^c, \mathcal{B}^c], \\ & I[\phi, \mathcal{B}] \subseteq I[\phi^c, \mathcal{B}^c], md(\phi, \mathcal{B}^c) \leq md(\phi^c, \mathcal{B}^c), ml(\phi, \mathcal{B}^c) \leq ml(\phi^c, \mathcal{B}^c) \}. \end{aligned}$$

The classical construction will take sets of formulas from the set  $\mathcal{L}[\phi^c, \mathcal{B}^c]$ , which are propositional maximal as defined in the next definition. However, in our setting, the set  $\mathcal{L}[\phi^c, \mathcal{B}^c]$  does not contain enough quantitative information for constructing the model yet. Therefore, there are two extension steps to gather all the quantitative information to get the right candidate for the states of the model, which are quantitative maximal and quantitative complete as defined in Definition 10 and Definition 11. This information will make sure that we are able to find the rational solutions for all the inequalities, which will be used to define the probabilities on the transitions.

► **Definition 9** (Propositional Maximal Set). A set  $\Lambda \subseteq \mathcal{L}[\phi^c, \mathcal{B}^c]$  is (propositional) maximal iff:

1. if  $\phi \in \Lambda$ , then  $\neg\phi \notin \Lambda$ ; if  $\phi \vee \psi \in \Lambda$ , then  $\phi \in \Lambda$  or  $\psi \in \Lambda$ ; if  $X \in \Lambda$  and  $X = \phi \in \mathcal{B}^c$ , then  $\phi \in \Lambda$ ;
2. for all  $\phi \in \mathcal{L}[\phi^c, \mathcal{B}^c]$ ,  $\langle x \rangle \phi : x \geq 0 \in \Lambda$  and  $\langle x \rangle \phi : x \leq 1 \in \Lambda$ ;
3. if  $\langle x \rangle \psi : \bar{a} \cdot \bar{x} \triangleleft r \in \Lambda$ , then  $\langle x \rangle \psi : \bar{a} \cdot \bar{x} \trianglelefteq r \in \Lambda$ .

Let  $\Pi[\phi^c, \mathcal{B}^c]$  the set of all the (propositional) maximal sets of  $\mathcal{L}[\phi^c, \mathcal{B}^c]$ . Since  $\mathcal{L}[\phi^c, \mathcal{B}^c]$  is finite,  $\Pi[\phi^c, \mathcal{B}^c]$  is finite and any  $\Lambda \in \Pi[\phi^c, \mathcal{B}^c]$  is finite. As we mentioned earlier,  $\mathcal{L}[\phi^c, \mathcal{B}^c]$  is not sufficient for constructing the model, so we will extend  $\mathcal{L}[\phi^c, \mathcal{B}^c]$  and  $\Pi[\phi^c, \mathcal{B}^c]$  in two steps. Firstly,  $\Lambda \in \Pi[\phi^c, \mathcal{B}^c]$  is not quantitatively maximized defined as follows:

► **Definition 10** (Quantitatively Maximized Set). A set  $A \subseteq \mathcal{L}$  is quantitatively maximized iff

1. if  $\langle x \rangle \phi : x \trianglelefteq r \in A$ , then  $\langle x \rangle \neg\phi : x \triangleright 1 - r \in A$ ;
2. if  $\langle x \rangle (\phi \wedge \psi) : x \trianglelefteq r_1 \in A$  and  $\langle x \rangle (\phi \wedge \neg\psi) : x \trianglelefteq r_2 \in A$ , then  $\langle x \rangle \phi : x \trianglelefteq r_1 + r_2 \in A$ ;
3. if  $\langle x_n \rangle \phi_n : x_n \triangleright r_n \in A$ ,  $\langle x \rangle \phi : \bar{a}\bar{x} \trianglelefteq r \in A$  and  $a_n \geq 0$ , then  $\langle x \rangle \phi \Big|_{n-1} : \bar{a}\bar{x} \Big|_{n-1} \trianglelefteq r - a_n r_n \in A$ ;
4. if  $\langle x_n \rangle \phi_n : x_n \triangleright r_n \in A$ ,  $\langle x \rangle \phi : \bar{a}\bar{x} \triangleleft r \in A$  and  $a_n \geq 0$ , then  $\langle x \rangle \phi \Big|_{n-1} : \bar{a}\bar{x} \Big|_{n-1} \triangleleft r - a_n r_n \in A$ ;
5. if  $\langle x_n \rangle \phi_n : x_n \trianglelefteq r_n \in A$ ,  $\langle x \rangle \phi : \bar{a}\bar{x} \trianglelefteq r \in A$  and  $a_n \leq 0$ , then  $\langle x \rangle \phi \Big|_{n-1} : \bar{a}\bar{x} \Big|_{n-1} \trianglelefteq r - a_n r_n \in A$ ;
6. if  $\langle x_n \rangle \phi_n : x_n \trianglelefteq r_n \in A$ ,  $\langle x \rangle \phi : \bar{a}\bar{x} \triangleleft r \in A$  and  $a_n \leq 0$ , then  $\langle x \rangle \phi \Big|_{n-1} : \bar{a}\bar{x} \Big|_{n-1} \triangleleft r - a_n r_n \in A$ .

The quantitative maximization extends the lower bound and upper bound of all the rationals considered. This makes sure that all the numbers related to the given formula are included. These numbers are needed in order to find all the solutions for the inequalities in  $\phi^c$  and  $\mathcal{B}^c$ .

**Extension Step I:** Let  $\frac{p_{max}}{gr(\phi^c, \mathcal{B}^c)}, \frac{p_{min}}{gr(\phi^c, \mathcal{B}^c)}$  with  $p_{max}, p_{min} \in \mathbb{Z}$  and  $\max', \min' \in \mathbb{Q}$  be such that if the conditions 1 – 4 below are satisfied, then for any  $\Lambda \in \Pi[\phi^c, \mathcal{B}^c]$ , there exists  $\Lambda' \in \Pi'[\phi^c, \mathcal{B}^c]$  such that  $\Lambda \subseteq \Lambda'$  and  $\Lambda'$  is quantitatively maximized,

1.  $I'[\phi^c, \mathcal{B}^c]$  be the set of all  $\frac{p}{gr(\phi^c, \mathcal{B}^c)}$  in the interval  $[\frac{p_{max}}{gr(\phi^c, \mathcal{B}^c)}, \frac{p_{min}}{gr(\phi^c, \mathcal{B}^c)}]$  for any  $p \in \mathbb{Z}$ ;
2.  $R'[\phi^c, \mathcal{B}^c] = \{r \in \mathbb{Q} \mid \min' \leq r \leq \max'\}$ ;
3.  $\mathcal{L}'[\phi^c, \mathcal{B}^c] \supseteq \mathcal{L}[\phi^c, \mathcal{B}^c]$  is the set of formulas defined as  $(\otimes)$  based on  $I'[\phi^c, \mathcal{B}^c]$  and  $R'[\phi^c, \mathcal{B}^c]$ ;
4.  $\Pi'[\phi^c, \mathcal{B}^c]$  the set of the propositional maximal sets of  $\mathcal{L}'[\phi^c, \mathcal{B}^c]$ .

In order to find the maximal number related to 2 in Definition 10, one can start with adding the  $\langle x \rangle \phi : x \trianglelefteq r_1 + r_2$  and its negation which are not in  $\mathcal{L}[\phi^c, \mathcal{B}^c]$  to get  $\mathcal{L}'[\phi^c, \mathcal{B}^c]$  and continue doing the same to the new  $\mathcal{L}'[\phi^c, \mathcal{B}^c]$ . Since  $\mathcal{L}[\phi^c, \mathcal{B}^c]$  is finite, this procedure will terminate. Similarly one can do the same for the others and find the numbers. It is obvious that  $\mathcal{L}'[\phi^c, \mathcal{B}^c]$  and  $\Pi'[\phi^c, \mathcal{B}^c]$  are still finite. For any  $\Lambda \in \Pi[\phi^c, \mathcal{B}^c]$ , choose  $\Lambda' \in \Pi'[\phi^c, \mathcal{B}^c]$

■ **Table 1** Tableau Rules.

$$\begin{aligned}
(\wedge) \frac{\{\phi_1, \phi_2, \Delta\} \subseteq \Lambda^+}{\{\phi_1 \wedge \phi_2, \Delta\} \subseteq \Lambda^+} \quad (\vee) \frac{\{\phi_i, \Delta\} \subseteq \Lambda^+}{\{\phi_1 \vee \phi_2, \Delta\} \subseteq \Lambda^+} \quad \phi_i \in \Lambda^+, i = 1 \text{ or } 2 \quad (\text{Reg}) \frac{\{\phi_X, \Delta\} \subseteq \Lambda^+}{\{X, \Delta\} \subseteq \Lambda^+} \quad X = \phi_X \in \mathcal{B} \\
(\text{Mod}) \frac{\Delta_1 \subseteq \Lambda_1^+ \cdots \Delta_k \subseteq \Lambda_k^+}{\Delta \subseteq \Lambda^+} \quad \emptyset \neq \Delta_j \subseteq \bigcup_{\langle x \rangle \phi: \bar{a} \cdot \bar{x} \geq r \in \Delta} \{\phi_1, \dots, \phi_n\} \subseteq \bigcup_{j=1, \dots, k} \Delta_j
\end{aligned}$$

s.t.  $\Lambda \subseteq \Lambda'$  and  $\Lambda'$  is quantitatively maximized. Let the set of the chosen  $\Lambda'$  be  $\Omega'[\phi^c, \mathcal{B}^c]$ , which is finite.

In order to define the distribution on the model correctly, we need to obtain more information about the maximal sets, which is the quantitative completeness defined as follows.

► **Definition 11** (Quantitatively Complete Set). Given any finite set  $\mathcal{L}^* \subseteq \mathcal{L}$ . A propositional maximal set  $\Lambda^*$  of  $\mathcal{L}^*$  is called quantitatively complete iff  $ul_{\Lambda^*}^\phi = ur_{\Lambda^*}^\phi$  for any  $\phi \in \mathcal{L}^*$ , where

$$ul_{\Lambda^*}^\phi = \max\{r \in \mathbb{Q} \mid \langle x \rangle \phi : x \geq r \in \Lambda^*\}, \quad ur_{\Lambda^*}^\phi = \min\{s \in \mathbb{Q} \mid \langle x \rangle \phi : x \leq s \in \Lambda^*\}.$$

The above notion captures the accuracy of the rationals, which states how precise we can express in the logic. This makes sure that we include (at least) one rational solution for every inequality.

► **Lemma 12.** For any  $\phi \in \mathcal{L}'[\phi^c, \mathcal{B}^c]$  ( $\mathcal{L}[\phi^c, \mathcal{B}^c]$ ) and any  $\Lambda' \in \Omega'[\phi^c, \mathcal{B}^c]$  ( $\Lambda \in \Pi[\phi^c, \mathcal{B}^c]$ ),

1.  $ul_{\Lambda'}^\phi, ur_{\Lambda'}^\phi \in [0, 1] \cap \mathbb{Q}$ ;
2. either  $ul_{\Lambda'}^\phi = ur_{\Lambda'}^\phi$ , or  $ul_{\Lambda'}^\phi + \frac{1}{gr(\phi^c, \mathcal{B}^c)} = ur_{\Lambda'}^\phi$ .

**Extension Step II:** Let  $h \in \mathbb{N}$  be such that if the conditions 1 – 3 below are satisfied, then for any  $\Lambda' \in \Omega'[\phi^c, \mathcal{B}^c]$ , there exists  $\Lambda^+ \in \Pi^+[\phi^c, \mathcal{B}^c]$  such that  $\Lambda' \subseteq \Lambda^+$  and  $\Lambda^+$  is quantitatively complete.

1.  $gr^+(\phi^c, \mathcal{B}^c) = gr(\phi, \mathcal{B}) \cdot 2^h$ ;
2.  $\mathcal{L}^+[\phi^c, \mathcal{B}^c] \supseteq \mathcal{L}'[\phi^c, \mathcal{B}^c]$  is the set of formulas defined as  $(\otimes)$  based on  $gr^+(\phi^c, \mathcal{B}^c)$ ;
3.  $\Pi^+[\phi^c, \mathcal{B}^c]$  the set of the propositional maximal sets of  $\mathcal{L}^+[\phi^c, \mathcal{B}^c]$ .

Since  $\mathcal{L}'[\phi^c, \mathcal{B}^c]$  is finite and all the numbers in the constraints on the quantitative variables are rationals, there exist rational solutions for the inequality systems. Hence, we can find such an  $h$  in finitely many steps by multiplying the granularity by 2 every time. Obviously  $\mathcal{L}^+[\phi^c, \mathcal{B}^c]$  and  $\Pi^+[\phi^c, \mathcal{B}^c]$  are finite. For any  $\Lambda' \in \Omega'[\phi^c, \mathcal{B}^c]$ , choose  $\Lambda^+ \in \Pi^+[\phi^c, \mathcal{B}^c]$  s.t.  $\Lambda' \subseteq \Lambda^+$  and  $\Lambda^+$  is quantitatively complete. Let the set of the chosen  $\Lambda^+$  be  $\Omega^+[\phi^c, \mathcal{B}^c]$ .

► **Lemma 13.** For any  $\phi \in \mathcal{L}^+[\phi^c, \mathcal{B}^c]$  and any  $\Lambda^+ \in \Omega^+[\phi^c, \mathcal{B}^c]$ ,

$$ul_{\Lambda^+}^\phi = ur_{\Lambda^+}^\phi \in [0, 1] \cap \mathbb{Q}.$$

In what follows, let  $u_{\Lambda^+}^\phi = ul_{\Lambda^+}^\phi = ur_{\Lambda^+}^\phi$ . Now we are ready to construct a model for  $\phi^c$  dependent on  $\mathcal{B}^c$ . We construct a tableau  $\mathcal{T}[\phi^c, \mathcal{B}^c]$  for  $\phi^c$  with  $\Lambda^+ \subseteq \Omega^+[\phi^c, \mathcal{B}^c]$  as the nodes. The reason for here, unlike in the standard construction [33, 15, 34], we consider  $\Lambda^+$  as a node is because we need to derive information about probabilities from the nodes. The tableau rules are listed in Table 1, where  $\Delta \subseteq \Lambda^+$  denotes  $\Lambda^+$  including  $\Delta$  and  $\{\phi, \Delta\}$  denotes  $\{\phi\} \cup \Delta$ .

If (Mod) is applied at node  $t$ , the nodes  $\Delta_j \subseteq \Lambda_j^+$  obtained from  $\overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \geq r$  s.t.  $\phi_i \in \Delta_j$  are called  $\phi_i$ -sons of  $t$ . The tableaux may be infinite. However, because  $\Omega^+[\phi^c, \mathcal{B}^c]$



and any  $\Lambda^+ \in \Omega^+[\phi^c, \mathcal{B}^c]$  are both finite, the nodes of the type  $\Delta \subseteq \Lambda^+$  appear in  $\mathcal{T}[\phi^c, \mathcal{B}^c]$  are finitely many.

As in the classic method for  $\mu$ -calculus [33, 15, 34], we use *max-trace*, *min-trace* to capture the idea of a history of the regeneration of a formula (similar to the classic definitions and presented in the full version of the paper). We adapt the notions of *markings*, *consistent markings* to the probability case to characterize  $\mathcal{B}$ -satisfiability of a formula in a state of a PMP.

► **Definition 14** (Marking). For a tableau  $\mathcal{T}$ , we define its *marking* with respect to a PMP  $\mathcal{M} = (M, \Sigma, l, \theta)$  and state  $m_0 \in M$  to be a relation  $\mathfrak{M} \subseteq M \times \mathcal{T}$  satisfying the following conditions:

- (i)  $(m_0, t_0) \in \mathfrak{M}$ , where  $t_0$  is the root of  $\mathcal{T}$ ;
- (ii) if  $(m, t) \in \mathfrak{M}$  and a rule other than (Mod) was applied at  $t$ , then for the son  $t'$  of  $t$ ,  $(m, t') \in \mathfrak{M}$ ;
- (iii) if  $(m, t) \in \mathfrak{M}$  with  $t = (\Delta \subseteq \Lambda^+)$  and rule (Mod) was applied at  $t$ , then for any  $\langle x \rangle \phi: \bar{a} \cdot \bar{x} \geq r \in \Delta$ , there exists  $F_1, \dots, F_n \subseteq M$  s.t. for any  $i = 1, \dots, n$ :
  - (a) for every  $\phi_i$ -son  $t'$  of  $t$ , there exists a state  $m' \in F_i$  s.t.  $(m', t') \in \mathfrak{M}$ , and
  - (b) for every state  $m' \in F_i$ , there exists a  $\phi_i$ -son  $t'$  of  $t$  s.t.  $(m', t') \in \mathfrak{M}$ , and
  - (c)  $u_{\Lambda^+}^{\phi_i} = \theta(m)(F_i)$ .

► **Definition 15** (Consistent Marking). A marking  $\mathfrak{M}$  of  $\mathcal{T}$  is *consistent* with respect to  $\mathcal{M} = (M, \Sigma, l, \theta)$  and  $m_0 \in M$ , if and only if  $\mathfrak{M}$  satisfies the following conditions:

- *local consistency*: for any node  $t = (\Delta \subseteq \Lambda^+) \in \mathcal{T}$  and state  $m \in M$ , if  $(m, t) \in \mathfrak{M}$  then for any  $\psi \in \Delta$ ,  $\mathcal{M}, m, 0 \models_{\mathcal{B}} \psi$ ;
- *global consistency*: for every path  $\mathcal{P} = t_0, t_1, \dots$  of  $\mathcal{T}$  s.t. there exist  $\pi_i$  with  $(\pi_i, t_i) \in \mathfrak{M}$  for  $i = 0, 1, \dots$ , there is no min-trace on  $\mathcal{P}$ .

► **Lemma 16**.  $\phi^c$  is satisfied at state  $m_0$  in a PMP  $\mathcal{M} = (M, \Sigma, l, \theta)$  if and only if there is a consistent marking of  $\mathcal{T}[\phi^c, \mathcal{B}^c]$  with respect to  $\mathcal{M}$  and  $m_0$ .

The proof of Lemma 16 relies on notion of *signature*, similar to that considered by Streett and Emerson [33]. These notions come from the characterization of fixed point formulas by means of transfinite chains of approximations, which have been extended to the setting with fixed points defined with blocks in [9, 10]. Involving these, the previous lemma is proven similarly to the case of classic  $\mu$ -calculus [33, 15, 34]. The correctness of the cases with probability is guaranteed by the quantitative maximization and quantitative completeness defined in Definition 10 and 11.

This lemma allows us to prove the finite model property for PMC, by following the classic proof strategy of [15]; the only difference consists in managing the probability modalities.

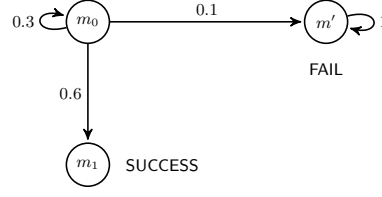
► **Theorem 17** (Finite Model Property). Let  $\phi_0 \in \mathcal{L}$  be a formula that depends of  $\mathcal{B}_0$ . If  $\phi_0$  is  $\mathcal{B}_0$ -satisfiable, then there exists a finite PMP  $\mathcal{M}^f = (M^f, \Sigma^f, l^f, \theta^f)$  with  $m^f \in M^f$  and an environment  $\rho^f$  such that  $\mathcal{M}^f, m^f, \rho^f \models_{\mathcal{B}_0} \phi_0$ .

According to Proposition 8, Lemma 16 and Theorem 17, we can obtain an algorithm to decide the satisfiability of a given PMC formula.

► **Algorithm**. Given a PMC formula  $\phi_0 \in \mathcal{L}$  dependent on the block sequence  $\mathcal{B}_0$ , the algorithm constructs a finite PMP  $\mathcal{M}^f = (M^f, \Sigma^f, l^f, \theta^f)$  and an environment  $\rho^f$  such that  $\mathcal{M}^f, m^f, \rho^f \models_{\mathcal{B}_0} \phi_0$  in the following steps:

1. Construct the co-prime block sequence  $\mathcal{B}_0^c$  of  $\mathcal{B}_0$  and the co-prime formula  $\phi_0^c$  of  $\phi_0$ .
2. Construct  $\mathcal{L}[\phi_0^c, \mathcal{B}_0^c]$  and  $\Pi[\phi_0^c, \mathcal{B}_0^c]$ , which are finite.





■ **Figure 2** Small model Construction.

3. Construct  $\mathcal{L}'[\phi_0^c, \mathcal{B}_0^c]$  and  $\Omega'[\phi_0^c, \mathcal{B}_0^c]$  by **Extension Step I**.  $\mathcal{L}'[\phi_0^c, \mathcal{B}_0^c]$  and  $\Omega'[\phi_0^c, \mathcal{B}_0^c]$  are finite.
4. Construct  $\mathcal{L}^+[\phi_0^c, \mathcal{B}_0^c]$  and  $\Omega^+[\phi_0^c, \mathcal{B}_0^c]$  by **Extension Step II**.  $\mathcal{L}^+[\phi_0^c, \mathcal{B}_0^c]$  and  $\Omega^+[\phi_0^c, \mathcal{B}_0^c]$  are finite.
5. Construct the tableau  $\mathcal{T}[\phi_0^c, \mathcal{B}_0^c]$  according to the rules in Table 1.
6. Construct  $\mathcal{M}^f = (M^f, \Sigma^f, \theta^f)$  as follows:
  - $M^f$  is the set of the nodes of  $t \in \mathcal{T}[\phi_0^c, \mathcal{B}_0^c]$  such that either (Mod) is applied at  $t$  or no rules are applicable at  $t$  ( $t$  is a leaf).
  - Let  $\langle \phi_i \rangle = \{\Lambda_i^+ \mid \phi_i \in \Lambda_i^+ \in M^f\}$ ,  $N = \{\langle \psi \rangle \mid \text{(Mod) is applied in } \mathcal{T}[\phi_0^c, \mathcal{B}_0^c] \text{ for } \overline{\langle x \rangle} \phi^n : \bar{\alpha} \bar{x} \geq r, \psi = \phi_i \text{ for some } i\}$ . Then  $\Sigma^f = \sigma(N)$ .
  - $l^f$  is defined as: for any  $t = \Lambda^+ \in M^f$ ,  $l^f(t) = \{p \in \mathcal{A} \mid p \in \Lambda^+\}$ .
  - For  $t = \Lambda^+ \in M^f$  where (Mod) is applied, let  $\theta^f(t)(\langle \phi \rangle) = u_{\Lambda^+}^\phi$  for any  $\langle \phi \rangle \in N$ . Let  $\rho^f(X) = \{\Lambda^+ \mid X \in \Lambda^+\}$  for  $X \in \mathcal{X}$ . By Theorem 17  $\mathcal{M}^f, t, \rho^f \models_{\mathcal{B}_0^c} \phi_0^c$  for  $t = \Lambda^+$  s.t.  $\phi_0^c \in \Lambda^+$ .
7. Therefore,  $\mathcal{M}^f, t, \rho^f \models_{\mathcal{B}_0} \phi_0$ , by Theorem 8.

► **Example 18.** Consider the property in Example 6:

$$\phi = \text{SUCCESS} \vee X$$

$$B = \max \left\{ \begin{array}{l} X = \neg \text{FAIL} \wedge \neg \text{SUCCESS} \\ \wedge (\langle x_1 \rangle \text{SUCCESS}, \langle x_2 \rangle X : x_1 + x_2 \geq 0.9) \end{array} \right\},$$

As discussed in Example 6,  $\phi$  is satisfiable. We can use the above algorithm to construct a model for it (the smallest one), as shown in Figure 2. The detailed steps of construction is omitted here.

► **Theorem 19** (Decidability of  $\mathcal{B}$ -Satisfiability). *The  $\mathcal{B}$ -satisfiability problem for PMC is decidable.*

PMC can be used to approximate pCTL formulas with arbitrary precision when restricting to finite models. Our approximation is based on a partition  $P : 0 < \pi_1 < \dots < \pi_k < 1$  of  $[0, 1]$ . To (under-)approximate the pCTL formula  $\phi_i = \mathbb{P}_{\geq \pi_i}(\phi_1 U \phi_2)$  in PMC, we define recursively:

$$B_P = \min \{ X_i^u = \phi_2^u \vee (\phi_1^u \wedge \overline{\langle x_j \rangle X_j^u} : \overline{\langle x_j - x_{j+1} \rangle} \pi_j \geq \pi_i \mid i = 1 \dots k) \}.$$

Let  $S_i$  be the set of states satisfying the pCTL formula  $\phi_i$ . Then the vector  $\langle S_i : i = 1 \dots k \rangle$  is a fixed point to the block  $B$  above, and it follows (from minimal fixed point semantics of  $B$ ) that  $X_i^u \Rightarrow \phi_i$ . Thus successful application of our finite-model property construction to  $X_i^u$  will provide a model for  $\phi_i$  as well. We conjecture that if there is a finite model  $\mathcal{M}$  satisfying  $\phi_i = \mathbb{P}_{\geq \pi}(\phi_1 U \phi_2)$ , then for any  $\epsilon > 0$  we can find a partitioning  $P$  such that  $\mathcal{M} \models_{B_P} X_i^u$  where  $\pi_i \geq \pi - \epsilon$ . This will be an alternative to the construction for pCTL

■ **Table 2** Axiomatic System of PMC basic formulas.

- (A1):  $\vdash \langle x \rangle \phi : x \geq 0 \wedge \langle x \rangle \phi : x \leq 1$   
 (A2):  $\vdash \overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \geq r \vee \overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \leq r$   
 (A3):  $\vdash \overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \leq r \rightarrow \langle x \rangle \phi : \bar{a} \cdot \bar{x} < s, r < s$   
 (A4):  $\vdash \neg(\overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \geq r) \leftrightarrow \overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} < r$   
 (A5):  $\vdash \langle x \rangle \phi : a \cdot x \geq r \leftrightarrow \langle x \rangle \neg \phi : a \cdot x < a - r$   
 (A6):  $\vdash \langle x_1 \rangle (\phi \wedge \psi) : x_1 \leq r_1 \wedge \langle x_2 \rangle (\phi \wedge \neg \psi) : x_2 \leq r_2 \rightarrow \langle x \rangle \phi : x \leq r_1 + r_2$   
 (A7):  $\vdash \overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \geq r \rightarrow \overline{\langle x \rangle \phi} : \alpha \cdot (\bar{a} \cdot \bar{x}) \geq \alpha r, \alpha \in \mathbb{Q}_{\geq 0}$   
 (A8):  $\vdash \overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \geq r \wedge \overline{\langle x \rangle \phi} : \bar{b} \cdot \bar{x} \geq s \rightarrow \overline{\langle x \rangle \phi} : (\bar{a} + \bar{b}) \cdot \bar{x} \geq r + s$   
 (A9): if  $a_n = 0$ , then  $\vdash \overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \geq r \rightarrow \overline{\langle x \rangle \phi} \Big|_{n-1} : \bar{a} \cdot \bar{x} \Big|_{n-1} \geq r$   
 (R1): if  $\vdash \phi \leftrightarrow \psi$ , then  $\vdash \langle x \rangle \phi : x \leq r \leftrightarrow \langle x \rangle \psi : x \leq r$   
 (R2):  $\{C[\overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \leq r] \mid r \triangleright s\} \vdash C[\overline{\langle x \rangle \phi} : \bar{a} \cdot \bar{x} \leq s]$

satisfiability problem in [1]. This also shows that, when restricting to finite models, even though we could not encode pCTL in PMC (e.g., the until operator), we could use a PMC theory (a (infinite) set of formulas) to approximate it.

## 4 Axiomatization for Alternation-free PMC

In this section, we propose an axiomatization for the validities of alternation-free fragment of PMC with respect to the PMP-semantics and prove it sound and (weak-)complete.

### 4.1 Sound axiomatization

In order to state the axioms for PMC we need to establish some notions. Let  $X$  be a metavariable quantifying over  $\mathcal{L}$  and  $\overline{\langle x \rangle \phi}(\mathbb{X}) = \langle x_1 \rangle \phi_1, \dots, \langle x_i \rangle \phi_i[\mathbb{X}], \dots, \langle x_n \rangle \phi_n$ . For arbitrary sequences  $\bar{\phi}_j = \phi_{j1} \dots \phi_{jk_j}$  and  $\bar{x}_j = x_{j1} \dots x_{jk_j}$ ,  $j = 1, \dots, l$ , we construct the following generic formula involving  $\mathbb{X}$ :

$$C[\mathbb{X}] = \overline{\langle x_1 \rangle \phi_1}(\overline{\langle x_2 \rangle \phi_2}(\dots(\overline{\langle x_l \rangle \phi_l}(\mathbb{X}) : \bar{a}_l \cdot \bar{x}_l \geq r_l) \dots) : \bar{a}_2 \cdot \bar{x}_2 \geq r_2) : \bar{a}_1 \cdot \bar{x}_1 \geq r_1.$$

We call  $C[X]$  a *context*; it can be instantiated to a PMC formula  $C[\phi]$  for  $\phi \in \mathcal{L}$ . Also  $\epsilon[X]$  is a context - the empty one - and for  $\phi \in \mathcal{L}$ ,  $\epsilon[\phi] = \phi$ . Notice that the metavariable  $\mathbb{X}$  only appears once in the syntax of the context, i.e., we only consider contexts with one hole.

The axiomatization of PMC is given in two phases. Firstly, we provide axioms for deriving the validities that do not depend on sequences of blocks; and secondly, we extend the axiomatization to recursive constructs. The axioms and rules presented in Table 2 together with the axioms and the rules of propositional logic axiomatize a classic deducibility relation (see [12]) for the non-recursive validities of PMC denoted by  $\vdash$ . The axioms and the rules are stated for arbitrary  $\phi, \psi \in \mathcal{L}$ ,  $r, s \in \mathbb{Q}$ ,  $x, y \in \mathcal{V}$  and arbitrary context  $C[\mathbb{X}]$ , where  $\{\leq, \triangleright\} = \{\leq, \geq\}$  and  $\triangleright \in \{<, >\}$ .

The axiom (A1) states that  $x$  is a probability. The axioms (A2)-(A3) state simple arithmetic facts. (A4) states that the dual of the equatioanl modality is itself. (A5) and (A6) state that the probability of reaching  $\phi$  or  $\neg \phi$  is 1. The axioms (A7)-(A9) show the arithmetic transformation of inequalities. The rule (R1) states that the probabilities of reaching two equivalent formulas are the same. (R2) is infinitary and encode the Archimedean properties of rational numbers.

■ **Table 3** Axiomatic System of Maximal Equation Blocks.

$$\begin{aligned}
(\text{max-R1}): & \text{ If } \vdash^* \phi, \text{ then } \vdash_B^* \phi \\
(\text{max-A1}): & \vdash_B^* \bigwedge_{i=1, \dots, N} (X_i \rightarrow \phi_i) \\
(\text{max-R2}): & \text{ If } \vdash_B^* \bigwedge_{i=1, \dots, N} (\psi_i \rightarrow \phi_i \{\overline{\Psi}/\overline{X}\}), \\
& \text{ then } \vdash_B^* \bigwedge_{i=1, \dots, N} (\psi_i \rightarrow X_i)
\end{aligned}$$

■ **Table 4** Axiomatic System of Minimum Equation Blocks.

$$\begin{aligned}
(\text{min-R1}): & \text{ If } \vdash^* \phi, \text{ then } \vdash_B^* \phi \\
(\text{min-A1}): & \vdash_B^* \bigwedge_{i=1, \dots, N} (\phi_i \rightarrow X_i) \\
(\text{min-R2}): & \text{ If } \vdash_B^* \bigwedge_{i=1, \dots, N} (\phi_i \{\overline{\Psi}/\overline{X}\} \rightarrow \psi_i), \\
& \text{ then } \vdash_B^* \bigwedge_{i=1, \dots, N} (X_i \rightarrow \psi_i)
\end{aligned}$$

► **Theorem 20** (Soundness). *The axiomatic system of  $\vdash$  is sound, i.e., for arbitrary  $\phi \in \mathcal{L}$ ,  $\vdash \phi$  implies  $\models \phi$ .*

Now we can proceed with the recursive constructs.

Given a maximal equation block  $B = \max\{X_1 = \phi_1, \dots, X_N = \phi_N\}$  and an arbitrary classical deducibility relation  $\vdash^*$ , we define the deducibility relation  $\vdash_B^*$  as the extension of  $\vdash^*$  given by the axioms and rules in Table 3, which are the equation-version of the classic fixed points axioms of  $\mu$ -calculus [15, 32, 29]. These are stated for arbitrary  $\phi \in \mathcal{L}$  and  $\overline{\Psi} = (\psi_1, \dots, \psi_N) \in \mathcal{L}^N$ , where  $\overline{X} = (X_1, \dots, X_N)$ . Similarly, we define a classical deducibility relation  $\vdash_B^*$  for a minimal equation block  $B = \min\{X_1 = \phi_1, \dots, X_N = \phi_N\}$  based on  $\vdash^*$  by using the axioms and rules in Table 4.

Given an alternation-free block sequence  $\mathcal{B} = B_1, \dots, B_m$ , we define the classical deducibility relations  $\vdash_0, \vdash_1, \dots, \vdash_m$  as follows and consequently get  $\vdash_{\mathcal{B}} = \vdash_m$ .

$$\vdash_0 = \vdash; \quad \vdash_i = \vdash_{B_i}^{i-1} \quad \text{for } i = 1, \dots, m$$

As usual, we say that a formula  $\phi$  (or a set  $\Phi$  of formulas) is  $\mathcal{B}$ -provable, denoted by  $\vdash_{\mathcal{B}} \phi$  (respectively  $\vdash_{\mathcal{B}} \Phi$ ), if it can be proven from the given axioms and rules of  $\vdash_{\mathcal{B}}$ . We denote by  $\overline{\Psi} = \{\phi \in \mathcal{L} \mid \Psi \vdash_{\mathcal{B}} \phi\}$ . An induction on the structure of the alternation-free blocks shows that all the theorems of  $\vdash_{\mathcal{B}}$  are sound in the PMC-semantics.

► **Theorem 21** (Extended Soundness). *The axiomatic system of  $\vdash_{\mathcal{B}}$  is sound, i.e., for any  $\phi \in \mathcal{L}$ ,*

$$\vdash_{\mathcal{B}} \phi \text{ implies } \models_{\mathcal{B}} \phi.$$

## 4.2 Completeness

In the rest of this section we prove that the axiomatic system of  $\vdash_{\mathcal{B}}$  is not only sound, but also (weak-) complete, meaning that all the  $\mathcal{B}$ -validities can be proved, as theorems, from the proposed axioms and rules, i.e., for arbitrary  $\phi \in \mathcal{L}$ ,  $\models_{\mathcal{B}} \phi$  implies  $\vdash_{\mathcal{B}} \phi$ . To complete this proof it is sufficient to show that any  $\mathcal{B}$ -consistent formula has a model.

For some set  $S \subseteq \mathcal{L}$ ,  $\Phi$  is  $(S, \mathcal{B})$ -maximally consistent if it is  $\mathcal{B}$ -consistent and no formula in  $S$  can be added to  $\Phi$  without making it inconsistent.  $\Phi$  is  $\mathcal{B}$ -maximally-consistent if it is  $(\mathcal{L}, \mathcal{B})$ -maximally-consistent.

In the following we fix a consistent formula  $\phi_0$  depending on a fixed alternation-free sequence  $\mathcal{B}_0$  and we construct a model. Let

$$\begin{aligned}
(\otimes) \quad \mathcal{L}[\phi_0, \mathcal{B}_0] &= \{\phi \in \mathcal{L} \mid \mathcal{X}[\phi, \mathcal{B}_0] \subseteq \mathcal{X}[\phi_0, \mathcal{B}_0], R[\phi, \mathcal{B}] \subseteq R[\phi_0, \mathcal{B}_0], \\
& I[\phi, \mathcal{B}] \subseteq I[\phi_0, \mathcal{B}_0], md(\phi, \mathcal{B}_0) \leq md(\phi_0, \mathcal{B}_0), ml(\phi, \mathcal{B}_0) \leq ml(\phi_0, \mathcal{B}_0)\}
\end{aligned}$$

and  $\Pi[\phi_0, \mathcal{B}_0]$  be the set of all the maximal consistent sets of  $\mathcal{L}[\phi_0, \mathcal{B}_0]$ . Similar to the arguments in Section 3,  $\mathcal{L}[\phi_0, \mathcal{B}_0]$  and  $\Pi[\phi_0, \mathcal{B}_0]$  are finite. Let  $\Pi$  be the set of the  $\mathcal{L}$ -maximal consistent sets.

Different from the model construction in Section 3, we take  $\mathcal{L}$ -maximally consistent sets as states. However, we don't take all the  $\mathcal{L}$ -maximally consistent sets as the state space, which are countably many. We will develop a finite state space as follows.

Since the set of instances of the infinitary rule in Table 2 is countable, we can use the *Rasiowa-Sikorski Lemma* [30, 12] to prove Lindenbaum's Lemma [11, 12] for PMC, following the technique in [16]. These lemmas are presented in the full version of the paper. Suppose that for each  $\Lambda \in \Pi[\phi_0, \mathcal{B}_0]$  we chose one  $\Gamma \in \Pi$  such that  $\Lambda \subseteq \Gamma$  (Lindenbaum's Lemma); to identify it, we denote this  $\Gamma$  by  $\Lambda^e$ . Let  $\Theta = \{\Lambda^e \in \Pi \mid \Lambda \in \Pi[\phi_0, \mathcal{B}_0]\}$ . Since  $\Pi[\phi_0, \mathcal{B}_0]$  is finite,  $\Theta$  is obviously finite as well.

In what follows we will construct a PMP  $\mathcal{M} = (\Theta, \Sigma, l, \theta)$  that satisfies  $\phi_0$  in one of its states. To do this, we have to properly define  $l, \Sigma$  and  $\theta$ .  $l$  is defined as:  $l(\Gamma) = \{p \in \mathcal{A} \mid p \in \Gamma\}$  for any  $\Gamma \in \Theta$ .

For defining  $\Sigma$  and  $\theta$ , we firstly observe that given a  $\mathcal{B}_0$ -maximally-consistent set of formulas, the information contained about the resource-variable for a given formula is complete, in the sense that we can really identify its value, since any real number can be seen as the limit of some sequences of rational numbers. This is exactly what the next lemma states.

► **Lemma 22.** *For arbitrary  $\Gamma \in \Theta$  and  $\phi \in \mathcal{L}[\phi_0, \mathcal{B}_0]$ ,*

$$\sup\{r \in \mathbb{Q}_{\geq 0} \mid \langle x \rangle \phi : x \geq r \in \Gamma\} = \inf\{s \in \mathbb{Q} \mid \langle x \rangle \phi : x \leq s \in \Gamma\} \in \mathbb{R} \cup [0, 1].$$

► **Lemma 23.** *Let  $\langle \phi \rangle = \{\Gamma \in \Theta \mid \phi \in \Gamma\}$  and  $N = \{\langle \phi \rangle \mid \phi \in \mathcal{L}[\phi_0, \mathcal{B}_0]\}$ . Then  $2^\Theta = N$ .*

Then let  $\Sigma = \sigma(N)$ , where  $\sigma(N)$  is the least  $\sigma$ -algebra generated by  $N$ . Then the previous lemmas allow us to define, for any  $\Gamma \in \Theta$  and  $\phi \in \mathcal{L}[\phi_0, \mathcal{B}_0]$ ,

$$\theta(\Gamma)(\langle \phi \rangle) = \sup\{r \in \mathbb{Q}_{\geq 0} \mid \langle x \rangle \phi : x \geq r \in \Gamma\}.$$

$\theta(\Gamma)$  is a set function defined on the field  $N$ . According to Theorem 11.3 of [3]<sup>2</sup>,  $\theta(\Gamma)$  can be uniquely extended to a measure on  $\Sigma$  if it is finitely additive and countably subadditive on  $\langle \phi \rangle$ . Since  $\Theta$  is finite, we only need to prove that  $\theta(\Gamma)$  is finitely additive, as stated in the following lemma. Notice also that since  $\Theta$  is finite,  $(\Theta, \Sigma)$  is an analytic space.

► **Lemma 24.** *For any  $\Gamma \in \Theta$ , the function  $\theta(\Gamma)$  is finitely additive, i.e., for any  $\langle \phi_1 \rangle$  and  $\langle \phi_2 \rangle$  s.t.  $\langle \phi_1 \rangle \cap \langle \phi_2 \rangle = \emptyset$ ,  $\theta(\Gamma)(\langle \phi_1 \rangle \cup \langle \phi_2 \rangle) = \theta(\Gamma)(\langle \phi_1 \rangle) + \theta(\Gamma)(\langle \phi_2 \rangle)$ .*

► **Theorem 25.**  *$\mathcal{M} = (\Theta, \Sigma, l, \theta)$  is a probabilistic Markov process.*

Let  $\rho_0$  be the environment defined as: for any  $X \in \mathcal{X}$ , by  $\rho_0(X) = \{\Gamma \mid X \in \Gamma\}$ .

Firstly, we prove the restricted truth lemma that does not consider recursive constructs.

► **Lemma 26 (Restricted Truth Lemma).** *For  $\phi \in \mathcal{L}[\phi_0, \mathcal{B}_0]$  and  $\Gamma \in \Theta$ ,*

$$\mathcal{M}, \Gamma, \rho_0 \models \phi \text{ iff } \phi \in \Gamma.$$

On the restricted truth lemma we can base the following two results that indicate how we can extend the results to include the recursive cases, as developed in [18].

<sup>2</sup> If  $\mathcal{F} \subseteq 2^M$  is a field of sets and  $\mu : \mathcal{F} \rightarrow \mathbb{R}_{\geq 0}$  is finitely additive and countably subadditive, then  $\mu$  extends uniquely to a measure on  $\sigma(\mathcal{F})$

► **Lemma 27.** *Let  $B = \max\{X_1 = \phi_1, \dots, X_N = \phi_N\}$  be an equation block in the sequence  $\mathcal{B}_0$  and  $\rho$  an environment such that  $\rho(X_i) = \{\Gamma \mid X_i \in \Gamma\}$  for any  $i = 1, \dots, N$ . For any  $\phi \in \mathcal{L}[\phi_0, \mathcal{B}_0]$  and  $\Gamma \in \Theta$ ,*

*if  $[\mathcal{M}, \Gamma, \rho \models \phi \text{ iff } \phi \in \Gamma]$ , then  $[\mathcal{M}, \Gamma, \llbracket B \rrbracket_\rho \models \phi \text{ iff } \phi \in \Gamma]$ .*

Since the minimal blocks are dual of the maximal ones, we have a similar lemma for minimal blocks.

► **Lemma 28.** *Let  $B = \min\{X_1 = \phi_1, \dots, X_N = \phi_N\}$  be an equation block in the sequence  $\mathcal{B}_0$  and  $\rho$  an environment such that  $\rho(X_i) = \{\Gamma \mid X_i \in \Gamma\}$  for any  $i = 1, \dots, N$ . For any  $\phi \in \mathcal{L}[\phi_0, \mathcal{B}_0]$  and  $\Gamma \in \Theta$ ,*

*if  $[\mathcal{M}, \Gamma, \rho \models \phi \text{ iff } \phi \in \Gamma]$ , then  $[\mathcal{M}, \Gamma, \llbracket B \rrbracket_\rho \models \phi \text{ iff } \phi \in \Gamma]$ .*

These lemmas allow us to prove the stronger version of the truth lemma.

► **Theorem 29 (Extended Truth Lemma).** *For  $\phi \in \mathcal{L}[\phi_0, \mathcal{B}_0]$  and  $\Gamma \in \Theta$ ,*

*$\mathcal{M}, \Gamma, \rho_0 \models_{\mathcal{B}} \phi \text{ iff } \phi \in \Gamma$ .*

A direct consequence of Theorem 29 is the completeness<sup>3</sup> of the axiomatic system.

► **Theorem 30 (Completeness).** *The axiomatic system of  $\vdash_{\mathcal{B}}$  is complete, i.e., for arbitrary  $\phi \in \mathcal{L}$ ,*

*$\models_{\mathcal{B}} \phi \text{ implies } \vdash_{\mathcal{B}} \phi$ .*

## 5 Conclusions

In this paper we have extended the probabilistic modal logic of [21], which is a modal logic allowing (in-)equational conditions on probabilities, with fixed point constructions in the form of block sequences, thus obtaining the probabilistic  $\mu$ -calculus (PMC).

We prove that PMC enjoys the finite model property and its satisfiability problem is decidable. In order to do this, we involved the classic tableau construction that had to be adapted to the more challenging probabilistic settings. These results generalize previous results from [23] and recommend our logic as a good trade-off between expressiveness and decidability. The second key contribution of our paper is the sound-complete axiomatization that we propose for the alternation-free fragment of PMC. At the best of our knowledge, the problem of axiomatizing probabilistic  $\mu$ -calculus has not been previously approached. The completeness proof is a non-standard extension of the filtration method, which can be easily adapted to other versions of probabilistic  $\mu$ -calculus.

Unlike for the standard  $\mu$ -calculus, the complexity of our algorithm is not clear. This is because for every formula, we only know that there exists the number  $h$  in **Extension Step II** such that all the inequalities in the given formula have rational solutions that can be expressed according to the accuracy defined, but we do not know how big  $h$  would be. The complexity of the satisfiability algorithm will be studied in the future work.

One might wonder whether there exists a finite axiomatization, as the model construction here is similar to that in Section 3 and the rules there are all finite. However, how we

<sup>3</sup> In this context by completeness we mean the weak-completeness. Since PMC is not compact, the weak- and strong-completeness do not coincide.

define the probability on the transition in Section 3 is using the truth that there is always rational solution(s) for any rational inequality. In [35], Zhou proved that there exists a finite axiomatization for Markov Logic by involving a finitary Archimedean rule (similar to our Rule (R2)). The idea there is similar to our satisfiability algorithm. We believe that similar arguments for finite axiomatization can be made for our logic as well by applying the Fourier–Motzkin elimination method [31] as in [35]. However, it is difficult to formalize this finite axiomatization. As we discussed in the last paragraph, we cannot know how precise we need to be in the logic in order to specify the solutions for the inequalities. Whether one can axiomatize and if yes how to will be interesting to look into.

Moreover, axiomatization for the full logic will also be considered. In the axiomatization here, the axioms and rules for fixed points are the same as those for the  $\mu$ -calculus. Hence, for the full PMC, we believe that the axiomatization would look the same. However, the difficulty for proving the completeness will be at least that for the full modal  $\mu$ -calculus [34].

---

## References

- 1 Nathalie Bertrand, John Fearnley, and Sven Schewe. Bounded satisfiability for PCTL. In *Computer Science Logic (CSL'12) – 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, pages 92–106. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.CSL.2012.92.
- 2 Girish Bhat and Rance Cleaveland. Efficient model checking via the equational  $\mu$ -calculus. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 304–312. IEEE Computer Society, 1996. doi:10.1109/LICS.1996.561358.
- 3 Patrick Billingsley. *Probability and Measure*. Wiley-Interscience, 1995.
- 4 Tomas Brazdil, Vojtech Forejt, Jan Kretinsky, and Antonın Kucera. The satisfiability problem for probabilistic CTL. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 391–402. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.21.
- 5 Pablo F. Castro, Cecilia Kilmurray, and Nir Piterman. Tractable probabilistic mu-calculus that expresses probabilistic temporal logics. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 211–223. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.211.
- 6 S. Chakraborty and J-P. Katoen. On the satisfiability of some simple probabilistic logics. In *Proceedings of the Thirty-First Annual IEEE Symposium on Logic in Computer Science, LICS 2016, 5-8 July 2016, New York City, USA, 2016*.
- 7 Corina Cırstea, Clemens Kupke, and Dirk Pattinson. EXPTIME tableaux for the coalgebraic  $\mu$ -calculus. In *Computer Science Logic, 23rd international Workshop, CSL 2009, 18th Annual Conference of the EACSL, Coimbra, Portugal, September 7-11, 2009. Proceedings*, pages 179–193. Springer, 2009. doi:10.1007/978-3-642-04027-6\_15.
- 8 Rance Cleaveland, S. Purushothaman Iyer, and Murali Narasimha. Probabilistic temporal logics via the modal mu-calculus. *Theor. Comput. Sci.*, 342(2-3):316–350, 2005. doi:10.1016/j.tcs.2005.03.048.
- 9 Rance Cleaveland, Marion Klein, and Bernhard Steffen. Faster model checking for the modal mu-calculus. In *Computer Aided Verification, Fourth International Workshop, CAV1992, Montreal, Canada, June 29 – July 1, 1992, Proceedings*, pages 410–422. Springer, 1992. doi:10.1007/3-540-56496-9\_32.

- 10 Rance Cleaveland and Bernhard Steffen. A linear-time model-checking algorithm for the alternation-free modal  $\mu$ -calculus. *Formal Methods in System Design*, 2(2):121–147, 1993. doi:10.1007/BF01383878.
- 11 Robert Goldblatt. On the role of the baire category theorem and dependent choice in the foundations of logic. *J. Symb. Log.*, 50(2):412–422, 1985. doi:10.2307/2274230.
- 12 Robert Goldblatt. Topological proofs of some rasiowa-sikorski lemmas. *Studia Logica*, 100(1-2):175–191, 2012. doi:10.1007/s11225-012-9374-2.
- 13 Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994. doi:10.1007/BF01211866.
- 14 Sergiu Hart and Micha Sharir. Probabilistic propositional temporal logics. *Information and Control*, 70(2/3):97–155, 1986. doi:10.1016/S0019-9958(86)80001-8.
- 15 Dexter Kozen. Results on the propositional  $\mu$ -calculus. In *Automata, Languages and Programming, 9th Colloquium, Aarhus, Denmark, July 12-16, 1982, Proceedings*, pages 348–359. Springer, 1982. doi:10.1007/BFb0012782.
- 16 Dexter Kozen, Kim G. Larsen, Radu Mardare, and Prakash Panangaden. Stone duality for markov processes. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 321–330. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.38.
- 17 Clemens Kupke and Dirk Pattinson. On modal logics of linear inequalities. In *Advances in Modal Logic 8, papers from the eighth conference on "Advances in Modal Logic," held in Moscow, Russia, 24-27 August 2010*, pages 235–255. College Publications, 2010.
- 18 Kim G. Larsen, Radu Mardare, and Bingtian Xue. Alternation-free weighted  $\mu$ -calculus: Decidability and completeness. *Electr. Notes Theor. Comput. Sci.*, 319:289–313, 2015. doi:10.1016/j.entcs.2015.12.018.
- 19 Kim Guldstrand Larsen. Proof systems for satisfiability in hennessy-milner logic with recursion. *Theor. Comput. Sci.*, 72(2&3):265–288, 1990. doi:10.1016/0304-3975(90)90038-J.
- 20 Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991. doi:10.1016/0890-5401(91)90030-6.
- 21 Kim Guldstrand Larsen and Arne Skou. Compositional verification of probabilistic processes. In *CONCUR'92, Third International Conference on Concurrency Theory, Stony Brook, NY, USA, August 24-27, 1992, Proceedings*, pages 456–471. Springer, 1992. doi:10.1007/BFb0084809.
- 22 Daniel J. Lehmann and Saharon Shelah. Reasoning with time and chance. *Information and Control*, 53(3):165–198, 1982. doi:10.1016/S0019-9958(82)91022-1.
- 23 Wanwei Liu, Lei Song, Ji Wang, and Lijun Zhang. A simple probabilistic extension of modal  $\mu$ -calculus. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 882–888. AAAI Press, 2015.
- 24 Henryk Michalewski and Matteo Mio. Baire category quantifier in monadic second order logic. In *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 362–374. Springer, 2015. doi:10.1007/978-3-662-47666-6\_29.
- 25 Matteo Mio. Probabilistic modal  $\mu$ -calculus with independent product. *Logical Methods in Computer Science*, 8(4), 2012. doi:10.2168/LMCS-8(4:18)2012.
- 26 Matteo Mio and Alex Simpson. Łukasiewicz  $\mu$ -calculus. In *Proceedings Workshop on Fixed Points in Computer Science, FICS 2013, Turino, Italy, September 1st, 2013.*, pages 87–104, 2013. doi:10.4204/EPTCS.126.7.
- 27 Matteo Mio, Alex Simpson, and Colin Stirling. *Game Semantics for Probabilistic Modal  $M$ -calculi*. University of Edinburgh, 2012.



- 28 Prakash Panangaden. *Labelled Markov Processes*. Imperial College Press, 2009.
- 29 Vaughan R. Pratt. A decidable mu-calculus: Preliminary report. In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 421–427. IEEE Computer Society, 1981. doi:10.1109/SFCS.1981.4.
- 30 H. Rasiowa and R. Sikorski. A proof of the completeness theorem of Gödel. *Fund. Math*, 37:193–200, 1950.
- 31 Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, 1986.
- 32 D. Scott and J. de Bakker. A theory of programs. *unpublished notes*, 1969.
- 33 Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Inf. Comput.*, 81(3):249–264, 1989. doi:10.1016/0890-5401(89)90031-X.
- 34 Igor Walukiewicz. Completeness of kozen’s axiomatisation of the propositional  $\mu$ -calculus. *Inf. Comput.*, 157(1-2):142–182, 2000. doi:10.1006/inco.1999.2836.
- 35 Chunlai Zhou. A complete deductive system for probability logic. *J. Log. Comput.*, 19(6):1427–1454, 2009. doi:10.1093/logcom/exp031.



# Interval vs. Point Temporal Logic Model Checking: an Expressiveness Comparison

Laura Bozzelli<sup>1</sup>, Alberto Molinari<sup>2</sup>, Angelo Montanari<sup>3</sup>,  
Adriano Peron<sup>4</sup>, and Pietro Sala<sup>5</sup>

- 1 Technical University of Madrid (UPM), Madrid, Spain
- 2 University of Udine, Udine, Italy
- 3 University of Udine, Udine, Italy
- 4 University of Napoli “Federico II”, Napoli, Italy
- 5 University of Verona, Verona, Italy

---

## Abstract

Model checking is a powerful method widely explored in formal verification to check the (state-transition) model of a system against desired properties of its behaviour. Classically, properties are expressed by formulas of a temporal logic, such as LTL, CTL, and CTL\*. These logics are “point-wise” interpreted, as they describe how the system evolves state-by-state. On the contrary, Halpern and Shoham’s interval temporal logic (HS) is “interval-wise” interpreted, thus allowing one to naturally express properties of computation stretches, spanning a sequence of states, or properties involving temporal aggregations, which are inherently “interval-based”.

In this paper, we study the expressiveness of HS in model checking, in comparison with that of the standard logics LTL, CTL, and CTL\*. To this end, we consider HS endowed with three semantic variants: the state-based semantics, introduced by Montanari et al., which allows branching in the past and in the future, the linear-past semantics, allowing branching only in the future, and the linear semantics, disallowing branching. These variants are compared, as for their expressiveness, among themselves and to standard temporal logics, getting a complete picture. In particular, HS with linear (resp., linear-past) semantics is proved to be equivalent to LTL (resp., finitary CTL\*).

**1998 ACM Subject Classification** F.4.1 Mathematical Logic, D.2.4 Software/Program Verification

**Keywords and phrases** Interval Temporal Logics, Expressiveness, Model Checking

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.26

## 1 Introduction

*Point-based temporal logics* (PTLs) provide a fundamental framework for the specification of the behavior of reactive systems, that makes it possible to describe how the system evolves state-by-state (“point-wise” view). PTLs have been successfully employed in *model checking* (MC), which enables one to automatically verify complex finite-state systems usually modelled as finite propositional Kripke structures. The MC methodology considers two types of PTLs – *linear* and *branching* – which differ in the underlying model of time. In linear temporal logics, such as LTL [23], each moment in time has a unique possible future: formulas are interpreted over paths of a Kripke structure, and thus they refer to a single computation of the system. In branching temporal logics, such as CTL and CTL\* [8], each moment in time may evolve into several possible futures: formulas are interpreted over states of the Kripke structure, hence referring to all the possible computations of a system.



© Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron, and Pietro Sala;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 26; pp. 26:1–26:14



Leibniz International Proceedings in Informatics

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

*Interval temporal logics* (ITLs) have been proposed as an alternative setting for reasoning about time [10, 22]. Unlike standard PTLs, they take intervals, rather than points, as their primitive entities. ITLs allow one to specify relevant temporal properties that involve, for instance, actions with duration, accomplishments, and temporal aggregations, which are inherently “interval-based”, and thus cannot be naturally expressed by PTLs. They have been applied in various areas of computer science, including formal verification, computational linguistics, planning, and multi-agent systems [14, 22, 24]. Halpern and Shoham’s modal logic of time intervals HS [10] is the most popular among ITLs. It features one modality for each of the 13 possible ordering relations between pairs of intervals (the so-called Allen’s relations [1]), apart from equality. Its *satisfiability problem* turns out to be undecidable for all interesting (classes of) linear orders [10]; the same happens with most of its fragments [7, 13, 17].

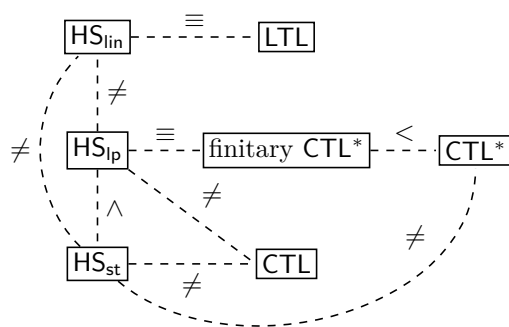
In this paper, we focus on the *model checking problem* for HS. In order to check interval properties of computations, one needs to collect information about states into computation stretches (i.e., finite paths of the Kripke structure, *tracks* for short): each track is interpreted as an interval, whose labelling is defined on the basis of the labelling of the component states. This approach to MC has independently and simultaneously been proposed by Molinari et al. in [18] and by Lomuscio and Michaliszyn in [14, 15, 16].

The semantics proposed in [18] is *state-based*, featuring intervals/tracks which are forgetful of the history leading to the starting state of the interval itself. Since the starting state (resp., ending state) of an interval may feature several predecessors (resp., successors), this interpretation induces a branching reference in both future and past. The other relevant choice in this approach concerns the labeling of intervals: a natural principle, known as the *homogeneity assumption*, is adopted, according to which a proposition holds over an interval if and only if it holds over each component state. Under this semantics, the MC problem for full HS turns out to be decidable – it is **EXPSpace**-hard, while the only known upper bound is non-elementary. The exact complexity of almost all the meaningful syntactic fragments of HS has been recently determined in a series of papers (e.g., [4, 6, 18, 19, 20, 21]).

The approach followed in [14, 15] is more expressive than the one in [18] since it relies on the extension of HS with knowledge modalities typical of the epistemic logics, which allow one to relate distinct paths of a Kripke structure. Additionally, the semantic assumptions differ from those of [18]: the logic is interpreted over the unwinding of the Kripke structure (*computation-tree-based* approach), and the interval labeling takes into account only the endpoints of the interval itself. A more expressive definition of interval labeling, obtained by associating each proposition with a regular expression over the set of states of the Kripke structure, was recently proposed in [16]. The decidability status of MC for full epistemic HS is currently unknown [14, 15].

In this paper, we study the expressiveness of HS, in the context of MC, in comparison with that of the standard PTLs LTL, CTL, and CTL\*. The investigation is carried on enforcing the homogeneity assumption. We prove that HS endowed with the state-based semantics proposed in [18] (hereafter denoted as  $HS_{st}$ ) is not comparable with LTL, CTL, and CTL\*. On the one hand, the result supports the intuition that  $HS_{st}$  gains some expressiveness by the ability to branch in the past. On the other hand,  $HS_{st}$  does not feature the possibility to force the verification of a property over an infinite path, thus implying that the formalisms are not comparable. With the aim of having a more “effective” comparison base, we consider two semantic variants of HS, besides the state-based semantics  $HS_{st}$ , namely, the *computation-tree-based semantics* (denoted as  $HS_{lp}$ ) and the *trace-based semantics* ( $HS_{lin}$ ).

The state-based and computation-tree-based approaches rely on a branching-time setting and differ in the nature of past. In the latter approach, the past is linear: each interval



■ **Figure 1** Overview of the expressiveness results.

may have several possible futures, but it has a unique past. Moreover, the past is assumed to be finite and cumulative (i.e., the history of the current situation increases with time, and is never forgotten). The trace-based approach relies on a linear-time setting, where the infinite paths (computations) of the given Kripke structure are the main semantic entities. Branching is neither allowed in the past nor in the future.

The variant  $HS_{ip}$  is a natural candidate for an expressiveness comparison with the branching time logics  $CTL$  and  $CTL^*$ . The more interesting and technically involved result is the characterization of  $HS_{ip}$ , which turns out to be expressively equivalent to  $\text{finitary } CTL^*$ , i.e., the variant of  $CTL^*$  with quantification over finite paths. As for  $CTL$ , a non-comparability result can be stated. Conversely,  $HS_{lin}$  is a natural candidate for an expressiveness comparison with  $LTL$ . As a matter of fact, we prove that  $HS_{lin}$  and  $LTL$  are equivalent (even for a small fragment of  $HS_{lin}$ ). We complete the picture with a comparison of the three semantic variants  $HS_{st}$ ,  $HS_{ip}$ , and  $HS_{lin}$ . We prove that, as expected,  $HS_{lin}$  is not comparable with either the branching versions,  $HS_{ip}$  and  $HS_{st}$ . The interesting result is that, on the other hand,  $HS_{ip}$  is strictly included in  $HS_{st}$ : this supports  $HS_{st}$ , adopted in [18, 19, 20, 21, 4, 6], as a reasonable and adequate semantic choice. The complete picture of the expressiveness results is reported in Figure 1 (the symbols  $\neq$ ,  $\equiv$  and  $<$  denote incomparability, equivalence, and strict expressiveness inclusion, respectively).

The paper is structured as follows. In Section 2, we introduce some preliminary notions. In Section 3 we prove the expressiveness results. In particular, in Section 3.1 we prove the equivalence between  $LTL$  and  $HS_{lin}$ ; in Section 3.2 we prove the equivalence between  $HS_{ip}$  and  $\text{finitary } CTL^*$ ; finally, in Section 3.3 we compare the logics  $HS_{st}$ ,  $HS_{ip}$ , and  $HS_{lin}$ .

## 2 Preliminaries

Let  $(\mathbb{N}, <)$  be the set of natural numbers equipped with the standard linear ordering. For all  $i, j \in \mathbb{N}$ , with  $i \leq j$ ,  $[i, j]$  denotes the set of natural numbers  $h$  such that  $i \leq h \leq j$ .

Let  $\Sigma$  be an alphabet and  $w$  be a non-empty finite or infinite word over  $\Sigma$ . We denote by  $|w|$  the length of  $w$  (we set  $|w| = \infty$  if  $w$  is infinite). For all  $i, j \in \mathbb{N}$ , with  $i \leq j$ ,  $w(i)$  denotes the  $i$ -th letter of  $w$ , while  $w[i, j]$  denotes the finite subword of  $w$  given by  $w(i) \cdots w(j)$ . If  $w$  is finite and  $|w| = n + 1$ , we define  $\text{fst}(w) = w(0)$  and  $\text{lst}(w) = w(n)$ .  $\text{Pref}(w) = \{w[0, i] \mid 0 \leq i \leq n - 1\}$  and  $\text{Suff}(w) = \{w[i, n] \mid 1 \leq i \leq n\}$  are the sets of all proper prefixes and suffixes of  $w$ , respectively.



■ **Figure 2** The Kripke structure  $\mathcal{K}$ .

## 2.1 Kripke structures and interval structures

► **Definition 1** (Kripke structure). A *Kripke structure* over a finite set  $\mathcal{AP}$  of proposition letters is a tuple  $\mathcal{K} = (\mathcal{AP}, S, \delta, \mu, s_0)$ , where  $S$  is a set of states,  $\delta \subseteq S \times S$  is a left-total transition relation,  $\mu : S \mapsto 2^{\mathcal{AP}}$  is a total labelling function assigning to each state  $s$  the set of propositions that hold over it, and  $s_0 \in S$  is the initial state. For  $(s, s') \in \delta$ , we say that  $s'$  is a successor of  $s$ , and  $s$  is a predecessor of  $s'$ . Finally, we say that  $\mathcal{K}$  is finite if  $S$  is finite.

Figure 2 depicts the finite Kripke structure  $\mathcal{K} = (\{p, q\}, \{s_0, s_1\}, \delta, \mu, s_0)$ , where  $\delta = \{(s_i, s_j) \mid i, j = 0, 1\}$ ,  $\mu(s_0) = \{p\}$ , and  $\mu(s_1) = \{q\}$ . The initial state  $s_0$  is marked by a double circle.

Let  $\mathcal{K} = (\mathcal{AP}, S, \delta, \mu, s_0)$  be a Kripke structure. An infinite path  $\pi$  of  $\mathcal{K}$  is an infinite word over  $S$  such that  $(\pi(i), \pi(i+1)) \in \delta$  for all  $i \geq 0$ . A *track* (or finite path) of  $\mathcal{K}$  is a non-empty prefix of some infinite path of  $\mathcal{K}$ . A finite or infinite path is *initial* if it starts from the initial state of  $\mathcal{K}$ . Let  $\text{Trk}_{\mathcal{K}}$  be the (*infinite*) set of all tracks of  $\mathcal{K}$  and  $\text{Trk}_{\mathcal{K}}^0$  be the set of initial tracks of  $\mathcal{K}$ . For a track  $\rho$ ,  $\text{states}(\rho)$  denotes the set of states occurring in  $\rho$ , i.e.,  $\text{states}(\rho) = \{\rho(0), \dots, \rho(n)\}$ , where  $|\rho| = n + 1$ .

► **Definition 2** (*D-tree structure*). For a given set  $D$  of directions, a *D-tree structure* (over  $\mathcal{AP}$ ) is a Kripke structure  $\mathcal{K} = (\mathcal{AP}, S, \delta, \mu, s_0)$  such that  $s_0 \in D$ ,  $S$  is a prefix closed subset of  $D^+$ , and  $\delta$  is the set of pairs  $(s, s') \in S \times S$  such that there exists  $d \in D$  for which  $s' = s \cdot d$  (note that  $\delta$  is completely specified by  $S$ ). The states of a *D-tree structure* are called *nodes*.

A Kripke structure  $\mathcal{K} = (\mathcal{AP}, S, \delta, \mu, s_0)$  induces an *S-tree structure*, called the *computation tree of  $\mathcal{K}$* , denoted by  $\mathcal{C}(\mathcal{K})$ , which is obtained by unwinding  $\mathcal{K}$  from the initial state. Formally,  $\mathcal{C}(\mathcal{K}) = (\mathcal{AP}, \text{Trk}_{\mathcal{K}}^0, \delta', \mu', s_0)$ , where the set of nodes is the set of initial tracks of  $\mathcal{K}$  and for all  $\rho, \rho' \in \text{Trk}_{\mathcal{K}}^0$ ,  $\mu'(\rho) = \mu(\text{lst}(\rho))$  and  $(\rho, \rho') \in \delta'$  iff  $\rho' = \rho \cdot s$  for some  $s \in S$ .

Given a strict partial ordering  $\mathbb{S} = (X, <)$ , an *interval* in  $\mathbb{S}$  is an ordered pair  $[x, y]$  such that  $x, y \in X$  and  $x \leq y$ . The interval  $[x, y]$  denotes the subset of  $X$  given by the set of points  $z \in X$  such that  $x \leq z \leq y$ . We denote by  $\mathbb{I}(\mathbb{S})$  the set of intervals in  $\mathbb{S}$ .

► **Definition 3** (*Interval structure*). An *interval structure*  $\mathbb{IS}$  over  $\mathcal{AP}$  is a pair  $\mathbb{IS} = (\mathbb{S}, \sigma)$  such that  $\mathbb{S} = (X, <)$  is a strict partial ordering and  $\sigma : \mathbb{I}(\mathbb{S}) \mapsto 2^{\mathcal{AP}}$  is a labeling function assigning a set of proposition letters to each interval over  $\mathbb{S}$ .

## 2.2 Standard temporal logics

In this subsection we recall the standard propositional temporal logics CTL\*, CTL, and LTL [8, 23]. For a set of proposition letters  $\mathcal{AP}$ , the formulas  $\varphi$  of CTL\* are defined as follows:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi \mid \exists\varphi,$$

where  $p \in \mathcal{AP}$ ,  $\mathbf{X}$  and  $\mathbf{U}$  are the “next” and “until” temporal modalities, and  $\exists$  is the existential path quantifier. We also use standard shorthands:  $\forall\varphi := \neg\exists\neg\varphi$  (“universal path quantifier”),  $\mathbf{F}\varphi := \top\mathbf{U}\varphi$  (“eventually”) and its dual  $\mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$  (“always”). The logic CTL is the fragment of CTL\* where each temporal modality is immediately preceded by a path quantifier, while LTL corresponds to the fragment of the formulas devoid of path quantifiers.

■ **Table 1** Allen’s relations and corresponding HS modalities.

Allen relation	HS	Definition w.r.t. interval structures	Example
MEETS	$\langle A \rangle$	$[x, y] \mathcal{R}_A [v, z] \iff y = v$	
BEFORE	$\langle L \rangle$	$[x, y] \mathcal{R}_L [v, z] \iff y < v$	
STARTED-BY	$\langle B \rangle$	$[x, y] \mathcal{R}_B [v, z] \iff x = v \wedge z < y$	
FINISHED-BY	$\langle E \rangle$	$[x, y] \mathcal{R}_E [v, z] \iff y = z \wedge x < v$	
CONTAINS	$\langle D \rangle$	$[x, y] \mathcal{R}_D [v, z] \iff x < v \wedge z < y$	
OVERLAPS	$\langle O \rangle$	$[x, y] \mathcal{R}_O [v, z] \iff x < v < y < z$	

Given a Kripke structure  $\mathcal{K} = (\mathcal{AP}, S, \delta, \mu, s_0)$ , an infinite path  $\pi$  of  $\mathcal{K}$ , and a position  $i \geq 0$  along  $\pi$ , the satisfaction relation  $\mathcal{K}, \pi, i \models \varphi$  for CTL\*, written simply  $\pi, i \models \varphi$  when  $\mathcal{K}$  is clear from the context, is defined as follows (Boolean connectives are treated as usual):

$$\begin{aligned}
\pi, i \models p &\iff p \in \mu(\pi(i)), \\
\pi, i \models \mathbf{X}\varphi &\iff \pi, i + 1 \models \varphi, \\
\pi, i \models \varphi_1 \mathbf{U} \varphi_2 &\iff \text{for some } j \geq i : \pi, j \models \varphi_2 \text{ and } \pi, k \models \varphi_1 \text{ for all } i \leq k < j, \\
\pi, i \models \exists \varphi &\iff \text{for some infinite path } \pi' \text{ starting from } \pi(i), \pi', 0 \models \varphi.
\end{aligned}$$

We say that  $\mathcal{K}$  is a model of  $\varphi$ , written  $\mathcal{K} \models \varphi$ , if for all initial infinite paths  $\pi$  of  $\mathcal{K}$ , it holds that  $\mathcal{K}, \pi, 0 \models \varphi$ . We also consider a variant of CTL\*, called *finitary* CTL\*, where the path quantifier  $\exists$  of CTL\* is replaced with the finitary path quantifier  $\exists_f$ . In this setting, path quantification ranges over the tracks (finite paths) starting from the current state. The satisfaction relation  $\rho, i \models \varphi$ , where  $\rho$  is a track and  $i$  is a position along  $\rho$ , is similar to that given for CTL\* with the only difference of finiteness of paths, and the fact that for a formula  $\mathbf{X}\varphi$ ,  $\rho, i \models \mathbf{X}\varphi$  iff  $i + 1 < |\rho|$  and  $\rho, i + 1 \models \varphi$ . A Kripke structure  $\mathcal{K}$  is a model of a finitary CTL\* formula if for each initial track  $\rho$  of  $\mathcal{K}$ , it holds that  $\mathcal{K}, \rho, 0 \models \varphi$ .

### 2.3 The interval temporal logic HS

An interval algebra was proposed by Allen in [1] to reason about intervals and their relative order, while a systematic logical study of interval representation and reasoning was done a few years later by Halpern and Shoham, who introduced the interval temporal logic HS featuring one modality for each Allen relation, but equality [10]. Table 1 depicts 6 of the 13 Allen’s relations, together with the corresponding HS (existential) modalities. The other 7 relations are the 6 inverse relations (given a binary relation  $\mathcal{R}$ , the inverse relation  $\overline{\mathcal{R}}$  is such that  $b\overline{\mathcal{R}}a$  if and only if  $a\mathcal{R}b$ ) and equality.

For a set of proposition letters  $\mathcal{AP}$ , the formulas  $\psi$  of HS are defined as follows:

$$\psi ::= p \mid \neg\psi \mid \psi \wedge \psi \mid \langle X \rangle \psi,$$

where  $p \in \mathcal{AP}$  and  $X \in \{A, L, B, E, D, O, \overline{A}, \overline{L}, \overline{B}, \overline{E}, \overline{D}, \overline{O}\}$ . For any modality  $\langle X \rangle$ , the dual universal modality  $[X]\psi$  is defined as  $\neg \langle X \rangle \neg\psi$ . For any subset of Allen’s relations  $\{X_1, \dots, X_n\}$ , let  $X_1 \cdots X_n$  be the HS fragment featuring modalities for  $X_1, \dots, X_n$  only.

We assume the *non-strict semantics* of HS, which admits intervals consisting of a single point.<sup>1</sup> Under such an assumption, all HS modalities can be expressed in terms of modalities

<sup>1</sup> All the results we prove in the paper hold for the strict semantics as well.

$\langle B \rangle$ ,  $\langle E \rangle$ ,  $\langle \bar{B} \rangle$ , and  $\langle \bar{E} \rangle$  [27], e.g., modality  $\langle A \rangle$  can be expressed in terms of  $\langle E \rangle$  and  $\langle \bar{B} \rangle$  as  $\langle A \rangle \varphi := ([E] \perp \wedge (\varphi \vee \langle \bar{B} \rangle \varphi)) \vee \langle E \rangle ([E] \perp \wedge (\varphi \vee \langle \bar{B} \rangle \varphi))$ . We also use the derived operator  $\langle G \rangle$  of HS (and its dual  $[G]$ ), which allows one to select arbitrary subintervals of the given interval and is defined as:  $\langle G \rangle \psi := \psi \vee \langle B \rangle \psi \vee \langle E \rangle \psi \vee \langle B \rangle \langle E \rangle \psi$ .

HS can be viewed as a multi-modal logic with  $\langle B \rangle$ ,  $\langle E \rangle$ ,  $\langle \bar{B} \rangle$ , and  $\langle \bar{E} \rangle$  as primitive modalities and its semantics can be defined over a multi-modal Kripke structure, called *abstract interval model*, where intervals are treated as atomic objects and Allen's relations as binary relations over intervals.

► **Definition 4** (Abstract interval model [18]). An *abstract interval model* over  $\mathcal{AP}$  is a tuple  $\mathcal{A} = (\mathcal{AP}, \mathbb{I}, B_{\mathbb{I}}, E_{\mathbb{I}}, \sigma)$ , where  $\mathbb{I}$  is a set of worlds,  $B_{\mathbb{I}}$  and  $E_{\mathbb{I}}$  are two binary relations over  $\mathbb{I}$ , and  $\sigma : \mathbb{I} \mapsto 2^{\mathcal{AP}}$  is a labeling function assigning a set of proposition letters to each world.

Let  $\mathcal{A} = (\mathcal{AP}, \mathbb{I}, B_{\mathbb{I}}, E_{\mathbb{I}}, \sigma)$  be an abstract interval model. In the interval setting,  $\mathbb{I}$  is interpreted as a set of intervals, and  $B_{\mathbb{I}}$  and  $E_{\mathbb{I}}$  as the Allen's relations  $B$  (*started-by*) and  $E$  (*finished-by*), respectively;  $\sigma$  assigns to each interval in  $\mathbb{I}$  the set of proposition letters that hold over it. Given an interval  $I \in \mathbb{I}$ , the truth of an HS formula over  $I$  is inductively defined as follows (Boolean connectives are treated as usual):

- $\mathcal{A}, I \models p$  iff  $p \in \sigma(I)$ , for any  $p \in \mathcal{AP}$ ;
- $\mathcal{A}, I \models \langle X \rangle \psi$ , for  $X \in \{B, E\}$ , iff there exists  $J \in \mathbb{I}$  such that  $I X_{\mathbb{I}} J$  and  $\mathcal{A}, J \models \psi$ ;
- $\mathcal{A}, I \models \langle \bar{X} \rangle \psi$ , for  $\bar{X} \in \{\bar{B}, \bar{E}\}$ , iff there exists  $J \in \mathbb{I}$  such that  $J X_{\mathbb{I}} I$  and  $\mathcal{A}, J \models \psi$ .

► **Definition 5** (Abstract interval model induced by an interval structure). An interval structure  $\mathcal{IS} = (\mathbb{S}, \sigma)$ , with  $\mathbb{S} = (X, <)$ , *induces* the abstract interval model  $\mathcal{A}_{\mathcal{IS}} = (\mathcal{AP}, \mathbb{I}(\mathbb{S}), B_{\mathbb{I}(\mathbb{S})}, E_{\mathbb{I}(\mathbb{S})}, \sigma)$ , where  $[x, y] B_{\mathbb{I}(\mathbb{S})} [v, z]$  iff  $x = v$  and  $z < y$ , and  $[x, y] E_{\mathbb{I}(\mathbb{S})} [v, z]$  iff  $y = z$  and  $x < v$ .

For an interval  $I$  and an HS formula  $\psi$ , we write  $\mathcal{IS}, I \models \psi$  to mean that  $\mathcal{A}_{\mathcal{IS}}, I \models \psi$ .

## 2.4 Three variants of HS semantics for model checking

In this section, we define the three variants of HS semantics  $\text{HS}_{\text{st}}$  (state-based semantics),  $\text{HS}_{\text{tp}}$  (computation-tree-based semantics), and  $\text{HS}_{\text{lin}}$  (trace-based semantics) for model checking HS against Kripke structures. For each such variant  $\mathcal{S}$ , the related (finite) model checking problem is deciding whether a finite Kripke structure is a model of an HS formula under  $\mathcal{S}$ .

Let us start with the *state-based semantics* [18], where an abstract interval model is naturally associated with a given Kripke structure  $\mathcal{K}$  by considering the set of intervals as the set  $\text{Trk}_{\mathcal{K}}$  of tracks of  $\mathcal{K}$ .

► **Definition 6** (Abstract interval model induced by a Kripke structure). The *abstract interval model induced by a Kripke structure*  $\mathcal{K} = (\mathcal{AP}, S, \delta, \mu, s_0)$  is  $\mathcal{A}_{\mathcal{K}} = (\mathcal{AP}, \mathbb{I}, B_{\mathbb{I}}, E_{\mathbb{I}}, \sigma)$ , where  $\mathbb{I} = \text{Trk}_{\mathcal{K}}$ ,  $B_{\mathbb{I}} = \{(\rho, \rho') \in \mathbb{I} \times \mathbb{I} \mid \rho' \in \text{Pref}(\rho)\}$ ,  $E_{\mathbb{I}} = \{(\rho, \rho') \in \mathbb{I} \times \mathbb{I} \mid \rho' \in \text{Suff}(\rho)\}$ , and  $\sigma : \mathbb{I} \mapsto 2^{\mathcal{AP}}$  is such that  $\sigma(\rho) = \bigcap_{s \in \text{states}(\rho)} \mu(s)$ , for all  $\rho \in \mathbb{I}$ .

According to the definition of  $\sigma$ ,  $p \in \mathcal{AP}$  holds over  $\rho = v_1 \cdots v_n$  if and only if it holds over all the states  $v_1, \dots, v_n$  of  $\rho$ . This conforms to the *homogeneity principle*, according to which a proposition letter holds over an interval if and only if it holds over all its subintervals [25].

► **Definition 7** (State-based semantics). Let  $\mathcal{K}$  be a Kripke structure and  $\psi$  be an HS formula. A track  $\rho \in \text{Trk}_{\mathcal{K}}$  satisfies  $\psi$  under the state-based semantics, denoted as  $\mathcal{K}, \rho \models_{\text{st}} \psi$ , if it holds that  $\mathcal{A}_{\mathcal{K}}, \rho \models \psi$ . Moreover,  $\mathcal{K}$  is a model of  $\psi$  under the state-based semantics, denoted as  $\mathcal{K} \models_{\text{st}} \psi$ , if for all *initial* tracks  $\rho \in \text{Trk}_{\mathcal{K}}^0$ , it holds that  $\mathcal{K}, \rho \models_{\text{st}} \psi$ .

We now introduce the *computation-tree-based semantics*, where we simply consider the abstract interval model *induced by the computation tree* of the Kripke structure. Notice that since each state in a computation tree has a unique predecessor (with the exception of the initial state), this HS semantic variant induces a linear reference in the past.

► **Definition 8** (Computation-tree-based semantics). A Kripke structure  $\mathcal{K}$  is a model of an HS formula  $\psi$  under the computation-tree-based semantics, written  $\mathcal{K} \models_{\text{ip}} \psi$ , if  $\mathcal{C}(\mathcal{K}) \models_{\text{st}} \psi$ .

Finally, we propose the *trace-based semantics*, which exploits the interval structures induced by the infinite paths of the Kripke structure.

► **Definition 9** (Interval structure induced by an infinite path). For a Kripke structure  $\mathcal{K} = (\mathcal{AP}, S, \delta, \mu, s_0)$  and an infinite path  $\pi = \pi(0)\pi(1)\dots$  of  $\mathcal{K}$ , the *interval structure induced by  $\pi$*  is  $IS_{\mathcal{K},\pi} = ((\mathbb{N}, <), \sigma)$ , where for each interval  $[i, j]$ ,  $\sigma([i, j]) = \bigcap_{h=i}^j \mu(\pi(h))$ .

► **Definition 10** (Trace-based semantics). A Kripke structure  $\mathcal{K}$  is a model of an HS formula  $\psi$  under the trace-based semantics, denoted as  $\mathcal{K} \models_{\text{lin}} \psi$ , iff for each initial infinite path  $\pi$  and for each initial interval  $[0, i]$ , it holds that  $IS_{\mathcal{K},\pi}, [0, i] \models \psi$ .

### 3 Expressiveness

In this section, we compare the expressive power of the logics  $\text{HS}_{\text{st}}$ ,  $\text{HS}_{\text{ip}}$ ,  $\text{HS}_{\text{lin}}$ , LTL, CTL, and CTL\* when interpreted over finite Kripke structures. Given two logics  $L_1$  and  $L_2$ , and two formulas  $\varphi_1 \in L_1$  and  $\varphi_2 \in L_2$ , we say that  $\varphi_1$  in  $L_1$  is *equivalent* to  $\varphi_2$  in  $L_2$  if, for every finite Kripke structure  $\mathcal{K}$ ,  $\mathcal{K}$  is a model of  $\varphi_1$  in  $L_1$  if and only if  $\mathcal{K}$  is a model of  $\varphi_2$  in  $L_2$ . When comparing the expressive power of two logics  $L_1$  and  $L_2$ , we say that  $L_2$  is *subsumed by*  $L_1$ , denoted as  $L_1 \geq L_2$ , if for each formula  $\varphi_2 \in L_2$ , there exists a formula  $\varphi_1 \in L_1$  such that  $\varphi_1$  in  $L_1$  is equivalent to  $\varphi_2$  in  $L_2$ . Moreover,  $L_1$  is *as expressive as*  $L_2$  (or,  $L_1$  and  $L_2$  have *the same expressiveness*), written  $L_1 \equiv L_2$ , if both  $L_1 \geq L_2$  and  $L_2 \geq L_1$ . We say that  $L_1$  is *more expressive than*  $L_2$  if  $L_1 \geq L_2$  and  $L_2 \not\geq L_1$ . Finally,  $L_1$  and  $L_2$  are *expressively incomparable* if both  $L_1 \not\geq L_2$  and  $L_2 \not\geq L_1$ .

#### 3.1 Equivalence between LTL and $\text{HS}_{\text{lin}}$

In this section we show that  $\text{HS}_{\text{lin}}$  is as expressive as LTL even for small syntactical fragments of  $\text{HS}_{\text{lin}}$ . For this purpose, we exploit the well-known equivalence between LTL and First Order Logic (FO) over infinite words. Recall that given a countable set  $\{x, y, z, \dots\}$  of (position) variables, FO formulas  $\varphi$  over a set of proposition symbols  $\mathcal{AP} = \{p, \dots\}$  are defined as:

$$\varphi := \top \mid p \in x \mid x \leq y \mid x < y \mid \neg \varphi \mid \varphi \wedge \varphi \mid \exists x. \varphi.$$

We interpret FO formulas  $\varphi$  over infinite paths  $\pi$  of Kripke structures  $\mathcal{K} = (\mathcal{AP}, S, \delta, \mu, s_0)$ . Given a variable *valuation*  $g$ , assigning to each variable a position  $i \geq 0$ , the satisfaction relation  $(\pi, g) \models \varphi$  corresponds to the standard satisfaction relation  $(\mu(\pi), g) \models \varphi$ , where  $\mu(\pi)$  is the infinite word over  $2^{\mathcal{AP}}$  given by  $\mu(\pi(0))\mu(\pi(1))\dots$  (for the details, see [5]). We write  $\pi \models \varphi$  to mean that  $(\pi, g_0) \models \varphi$ , where  $g_0(x) = 0$  for each variable  $x$ . An FO sentence is a formula with no free variables. The following is a well-known result [11].

► **Proposition 11.** *Given a FO sentence  $\varphi$  over  $\mathcal{AP}$ , one can construct an LTL formula  $\psi$  such that for all Kripke structures  $\mathcal{K}$  over  $\mathcal{AP}$  and infinite paths  $\pi$ , it holds that  $\pi \models \varphi$  iff  $\pi, 0 \models \psi$ .*



Given a  $\text{HS}_{\text{lin}}$  formula  $\psi$ , we construct a FO sentence  $\psi_{\text{FO}}$  such that, for all Kripke structures  $\mathcal{K}$ ,  $\mathcal{K} \models_{\text{lin}} \psi$  iff for each initial infinite path  $\pi$  of  $\mathcal{K}$ ,  $\pi \models \psi_{\text{FO}}$ . The formula  $\psi_{\text{FO}}$  is given by  $\exists x((\forall z.z \geq x) \wedge \forall y.h(\psi, x, y))$ , where  $h(\psi, x, y)$  is a FO formula having  $x$  and  $y$  as free variables (intuitively, representing the endpoints of the current interval) and ensuring that for each infinite path  $\pi$  and interval  $[i, j]$ ,  $\mathcal{IS}_{\mathcal{K}, \pi}, [i, j] \models \psi$  iff  $(\pi, g) \models h(\psi, x, y)$  for any valuation  $g$  such that  $g(x) = i$  and  $g(y) = j$ . The construction of  $h(\psi, x, y)$  is straightforward (for the details, see the report [5]). Thus, by Proposition 11, we obtain the following result.

► **Theorem 12.**  $\text{LTL} \geq \text{HS}_{\text{lin}}$ .

Conversely, we show that LTL can be translated in linear-time into  $\text{HS}_{\text{lin}}$  (actually, the fragment AB, featuring only modalities for  $A$  and  $B$ , is expressive enough for the purpose). In the following we will make use of the B formula  $\text{length}_n$ , with  $n \geq 1$ , characterizing the intervals of length  $n$ , which is defined as follows:  $\text{length}_n := (\underbrace{\langle B \rangle \dots \langle B \rangle}_{n-1} \top) \wedge (\underbrace{[B] \dots [B]}_n \perp)$ .

► **Theorem 13.** *Given an LTL formula  $\varphi$ , one can construct in linear-time an AB formula  $\psi$  such that  $\varphi$  in LTL is equivalent to  $\psi$  in  $\text{AB}_{\text{lin}}$ .*

**Proof.** Let  $f : \text{LTL} \mapsto \text{AB}$  be the mapping homomorphic w. r. to the Boolean connectives, defined as follows for each proposition  $p$  and for the temporal modalities X and U:

$$\begin{aligned} f(p) &= p, & f(\text{X}\psi) &= \langle A \rangle (\text{length}_2 \wedge \langle A \rangle (\text{length}_1 \wedge f(\psi))), \\ f(\psi_1 \text{U} \psi_2) &= \langle A \rangle \left( \langle A \rangle (\text{length}_1 \wedge f(\psi_2)) \wedge [B] (\langle A \rangle (\text{length}_1 \wedge f(\psi_1))) \right). \end{aligned}$$

Given a Kripke structure  $\mathcal{K}$ , an infinite path  $\pi$ , a position  $i \geq 0$ , and an LTL formula  $\psi$ , by a straightforward induction on the structure of  $\psi$  we can show that  $\pi, i \models \psi$  iff  $\mathcal{IS}_{\mathcal{K}, \pi}, [i, i] \models f(\psi)$ . Hence  $\mathcal{K} \models \psi$  iff  $\mathcal{K} \models_{\text{lin}} \text{length}_1 \rightarrow f(\psi)$ . ◀

► **Corollary 14.**  $\text{HS}_{\text{lin}}$  and LTL have the same expressiveness.

### 3.2 A characterization of $\text{HS}_{\text{lp}}$

In this section we show that  $\text{HS}_{\text{lp}}$  is as expressive as *finitary*  $\text{CTL}^*$ . Actually, the result can be proved to hold already for the syntactical fragment ABE (which does not feature transposed modalities). In addition, we show that  $\text{HS}_{\text{lp}}$  is subsumed by  $\text{CTL}^*$ .

We first show that finitary  $\text{CTL}^*$  is subsumed by  $\text{HS}_{\text{lp}}$ . The result is proved by exploiting a preliminary property stating that, when interpreted over finite words, the BE fragment of HS and LTL define the same class of finitary languages. For an LTL formula  $\varphi$  with proposition symbols over an alphabet  $\Sigma$  (in our case  $\Sigma$  is  $2^{\mathcal{AP}}$ ),  $L_{\text{act}}(\varphi)$  denotes the set of non-empty finite words over  $\Sigma$  satisfying  $\varphi$  under the standard action-based semantics of LTL, interpreted over finite words (see [26]). A similar notion can be given for BE formulas  $\varphi$  with propositional symbols in  $\Sigma$  (considered under the homogeneity principle). Then  $\varphi$  denotes a language, written  $L_{\text{act}}(\varphi)$ , of non-empty finite words over  $\Sigma$ , inductively defined as:

- $L_{\text{act}}(a) = a^+$  for each  $a \in \Sigma$ ;
- $L_{\text{act}}(\neg\varphi) = \Sigma^+ \setminus L_{\text{act}}(\varphi)$ ;
- $L_{\text{act}}(\varphi_1 \wedge \varphi_2) = L_{\text{act}}(\varphi_1) \cap L_{\text{act}}(\varphi_2)$ ;
- $L_{\text{act}}(\langle B \rangle \varphi) = \{w \in \Sigma^+ \mid \text{Pref}(w) \cap L_{\text{act}}(\varphi) \neq \emptyset\}$ ;
- $L_{\text{act}}(\langle E \rangle \varphi) = \{w \in \Sigma^+ \mid \text{Suff}(w) \cap L_{\text{act}}(\varphi) \neq \emptyset\}$ .



We prove that under the action-based semantics, BE formulas and LTL formulas define the same class of finitary languages. By proceeding as in Section 3.1, one can easily show that, over finite words, the class of languages defined by the fragment BE is subsumed by that defined by LTL. To prove the converse direction we exploit an algebraic condition introduced in [28], here called *LTL-closure*, which gives, for a class of finitary languages, a sufficient condition to guarantee the inclusion of the class of LTL-definable languages.

► **Definition 15** (LTL-closure). A class  $\mathcal{C}$  of languages of finite words over finite alphabets is *LTL-closed* iff the following conditions are satisfied, where  $\Sigma$  and  $\Delta$  are finite alphabets,  $b \in \Sigma$  and  $\Gamma = \Sigma \setminus \{b\}$ :

1.  $\mathcal{C}$  is closed under language complementation and language intersection.
2. If  $L \in \mathcal{C}$  with  $L \subseteq \Gamma^+$ , then  $\Sigma^*bL$ ,  $\Sigma^*b(L + \varepsilon)$ ,  $Lb\Sigma^*$ ,  $(L + \varepsilon)b\Sigma^*$  are in  $\mathcal{C}$ .
3. Let  $U_0 = \Gamma^*b$ ,  $h_0 : U_0 \rightarrow \Delta$  and  $h : U_0^+ \rightarrow \Delta^+$  be defined by  $h(u_0u_1 \dots u_n) = h_0(u_0) \dots h_0(u_n)$ . Assume that for each  $d \in \Delta$ , the language  $L_d = \{u \in \Gamma^+ \mid h_0(ub) = d\}$  is in  $\mathcal{C}$ . Then for each language  $L \in \mathcal{C}$  s.t.  $L \subseteq \Delta^+$ , the language  $\Gamma^*bh^{-1}(L)\Gamma^*$  is in  $\mathcal{C}$ .

► **Theorem 16** ([28]). Any LTL-closed class  $\mathcal{C}$  of finitary languages includes the class of LTL-definable finitary languages.

► **Theorem 17**. Let  $\varphi$  be an LTL formula over a finite alphabet  $\Sigma$ . Then there exists a BE formula  $\varphi_{\text{HS}}$  over  $\Sigma$  such that  $L_{\text{act}}(\varphi_{\text{HS}}) = L_{\text{act}}(\varphi)$ .

**Proof.** It suffices to prove that the class of finitary languages definable by BE formulas is LTL-closed, and to apply Theorem 16 (the proof of LTL-closure is reported in [5]). ◀

By exploiting Theorem 17, we establish the following result.

► **Theorem 18**. Let  $\varphi$  be a finitary CTL\* formula over  $\mathcal{AP}$ . Then there is an ABE formula  $\varphi_{\text{HS}}$  over  $\mathcal{AP}$  s.t. for all Kripke structures  $\mathcal{K}$  over  $\mathcal{AP}$  and tracks  $\rho$ ,  $\mathcal{K}, \rho, 0 \models \varphi$  iff  $\mathcal{K}, \rho \models_{\text{st}} \varphi_{\text{HS}}$ .

**Proof.** The proof is by induction on the nesting depth of modality  $\exists_f$  in  $\varphi$ . The base case ( $\varphi$  is a finitary LTL formula over  $\mathcal{AP}$ ) is similar to the inductive step, thus we can focus our attention on the latter. Let  $H$  be the non-empty set of subformulas of  $\varphi$  of the form  $\exists_f\psi$  which do not occur in the scope of the path quantifier  $\exists_f$ . Then  $\varphi$  can be seen as an LTL formula over the set of atomic propositions  $\overline{\mathcal{AP}} = \mathcal{AP} \cup H$ . Let  $\Sigma = 2^{\overline{\mathcal{AP}}}$  and  $\overline{\varphi}$  be the LTL formula over  $\Sigma$  obtained from  $\varphi$  by replacing each occurrence of  $p \in \overline{\mathcal{AP}}$  in  $\varphi$  with the formula  $\bigvee_{P \in \Sigma : p \in P} P$ , according to the LTL action-based semantics.

Given a Kripke structure  $\mathcal{K}$  over  $\mathcal{AP}$  with labeling  $\mu$  and a track  $\rho$  of  $\mathcal{K}$ , we denote by  $\rho_H$  the finite word over  $2^{\overline{\mathcal{AP}}}$  of length  $|\rho|$  defined as  $\rho_H(i) = \mu(\rho(i)) \cup \{\exists_f\psi \in H \mid \mathcal{K}, \rho, i \models \exists_f\psi\}$ , for all  $i \in [0, |\rho| - 1]$ . One can easily show by structural induction on  $\varphi$  that:

*Claim 1:*  $\mathcal{K}, \rho, 0 \models \varphi$  iff  $\rho_H \in L_{\text{act}}(\overline{\varphi})$ .

By Theorem 17, there exists a BE formula  $\overline{\varphi}_{\text{HS}}$  over  $\Sigma$  such that  $L_{\text{act}}(\overline{\varphi}) = L_{\text{act}}(\overline{\varphi}_{\text{HS}})$ . Moreover, by the induction hypothesis, for each formula  $\exists_f\psi \in H$ , there exists an ABE formula  $\psi_{\text{HS}}$  such that for all Kripke structures  $\mathcal{K}$  and tracks  $\rho$  of  $\mathcal{K}$ ,  $\mathcal{K}, \rho, 0 \models \psi$  iff  $\mathcal{K}, \rho \models_{\text{st}} \psi_{\text{HS}}$ . Since  $\rho$  is arbitrary,  $\mathcal{K}, \rho, i \models \exists_f\psi$  iff  $\mathcal{K}, \rho[i, i], 0 \models \exists_f\psi$  iff  $\mathcal{K}, \rho[i, i] \models_{\text{st}} \langle A \rangle \psi_{\text{HS}}$ , for each  $i \geq 0$ . Let  $\varphi_{\text{HS}}$  be the ABE formula over  $\mathcal{AP}$  obtained from the BE formula  $\overline{\varphi}_{\text{HS}}$  by replacing each occurrence of  $P \in \Sigma$  in  $\overline{\varphi}_{\text{HS}}$  with the formula

$$[G] \left( \text{length}_1 \rightarrow \bigwedge_{\exists_f\psi \in H \cap P} \langle A \rangle \psi_{\text{HS}} \wedge \bigwedge_{\exists_f\psi \in H \setminus P} \neg \langle A \rangle \psi_{\text{HS}} \wedge \bigwedge_{p \in \mathcal{AP} \cap P} p \wedge \bigwedge_{p \in \mathcal{AP} \setminus P} \neg p \right).$$

Since for all  $i \geq 0$  and  $\exists_f\psi \in H$ ,  $\mathcal{K}, \rho, i \models \exists_f\psi$  iff  $\mathcal{K}, \rho[i, i] \models_{\text{st}} \langle A \rangle \psi_{\text{HS}}$ , by a straightforward induction on the structure of  $\overline{\varphi}_{\text{HS}}$ , for all Kripke structures  $\mathcal{K}$  and tracks  $\rho$  of  $\mathcal{K}$  we have

$\mathcal{K}, \rho \models_{\text{st}} \varphi_{\text{HS}}$  iff  $\rho_H \in L_{\text{act}}(\overline{\varphi}_{\text{HS}})$ . Therefore, since  $L_{\text{act}}(\overline{\varphi}) = L_{\text{act}}(\overline{\varphi}_{\text{HS}})$ , by Claim 1  $\mathcal{K}, \rho, 0 \models \varphi$  iff  $\mathcal{K}, \rho \models_{\text{st}} \varphi_{\text{HS}}$ , for arbitrary Kripke structures  $\mathcal{K}$  and tracks  $\rho$  of  $\mathcal{K}$ .  $\blacktriangleleft$

Since for the fragment ABE of HS the computation-tree-based semantics coincides with the state-based semantics, by Theorem 18 we obtain the following corollary.

► **Corollary 19.** *Finitary CTL\* is subsumed by both HS<sub>st</sub> and HS<sub>lp</sub>.*

Conversely, we show now that HS<sub>lp</sub> is subsumed by both CTL\* and the finitary variant of CTL\*. For this purpose, we first introduce a hybrid and linear-past extension of CTL\*, called *hybrid CTL\*<sub>lp</sub>*, and its finitary variant, called *finitary hybrid CTL\*<sub>lp</sub>*. Hybrid logics (see [3]), besides standard modalities, make use of explicit variables and quantifiers that bind them. Variables and binders allow us to easily mark points in a path, which will be considered as starting and ending points of intervals, thus permitting a natural encoding of HS<sub>lp</sub>. Actually, we will show that the restricted form of use of variables and binders exploited in our encoding does not increase the expressive power of (finitary) CTL\* (as it happens for an unrestricted use), thus proving the desired result. We start by defining *hybrid CTL\*<sub>lp</sub>*.

For a countable set  $\{x, y, z, \dots\}$  of (position) variables, the set of formulas  $\varphi$  of hybrid CTL\*<sub>lp</sub> over  $\mathcal{AP}$  is defined as follows:

$$\varphi ::= \top \mid p \mid x \mid \neg\varphi \mid \varphi \vee \varphi \mid \downarrow x.\varphi \mid X\varphi \mid \varphi U\varphi \mid X^-\varphi \mid \varphi U^-\varphi \mid \exists\varphi,$$

where  $X^-$  (“previous”) and  $U^-$  (“since”) are the past counterparts of the “next” and “until” modalities  $X$  and  $U$ , and  $\downarrow x$  is the *downarrow binder operator* [3], which binds  $x$  to the current position along the given initial infinite path. We also use the standard shorthands  $F^-\varphi := \top U^-\varphi$  (“eventually in the past”) and its dual  $G^-\varphi := \neg F^-\neg\varphi$  (“always in the past”). As usual, a sentence is a formula with no free variables.

Let  $\mathcal{K}$  be a Kripke structure and  $\varphi$  be a hybrid CTL\*<sub>lp</sub> formula. For an *initial* infinite path  $\pi$  of  $\mathcal{K}$ , a variable valuation  $g$  assigning to each variable  $x$  a position along  $\pi$ , and  $i \geq 0$ , the satisfaction relation  $\pi, g, i \models \varphi$  is defined as follows (we omit the clauses for the Boolean connectives and for  $U$  and  $X$ ):

$$\begin{aligned} \pi, g, i \models X^-\varphi & \Leftrightarrow i > 0 \text{ and } \pi, g, i-1 \models \varphi, \\ \pi, g, i \models \varphi_1 U^-\varphi_2 & \Leftrightarrow \text{for some } j \leq i : \pi, g, j \models \varphi_2 \text{ and } \pi, g, k \models \varphi_1 \text{ for all } j < k \leq i, \\ \pi, g, i \models \exists\varphi & \Leftrightarrow \text{for some initial infinite path } \pi' \text{ s.t. } \pi'[0, i] = \pi[0, i], \pi', g, i \models \varphi, \\ \pi, g, i \models x & \Leftrightarrow g(x) = i, \\ \pi, g, i \models \downarrow x.\varphi & \Leftrightarrow \pi, g[x \leftarrow i], i \models \varphi, \end{aligned}$$

where  $g[x \leftarrow i](x) = i$  and  $g[x \leftarrow i](y) = g(y)$  for  $y \neq x$ . A Kripke structure  $\mathcal{K}$  is a model of a formula  $\varphi$  if for each initial infinite path  $\pi$ ,  $\pi, g_0, 0 \models \varphi$ , where  $g_0$  assigns 0 to each variable. Note that path quantification is “memoryful”, i.e., it ranges over infinite paths that start at the root and visit the current node of the computation tree. Clearly, the semantics for the syntactical fragment CTL\* coincides with the standard one. If we disallow the use of variables and binder modalities, we obtain the logic CTL\*<sub>lp</sub>, a well-known linear-past and equally expressive extension of CTL\* [12]. We also consider the finitary variant of hybrid CTL\*<sub>lp</sub>, where the path quantifier  $\exists$  is replaced with the finitary path quantifier  $\exists_f$ . This logic corresponds to an extension of finitary CTL\* and its semantics is similar to that of hybrid CTL\*<sub>lp</sub> with the exception that path quantification ranges over the *finite* paths (tracks) that start at the root and visit the current node of the computation tree.

In the following we will use the fragment of hybrid CTL\*<sub>lp</sub> consisting of *well-formed* formulas, namely, formulas  $\varphi$  where: (1) *each subformula  $\exists\psi$  of  $\varphi$  has at most one free*

variable; (2) each subformula  $\exists\psi(x)$  of  $\varphi$  having  $x$  as free variable occurs in  $\varphi$  in the context  $(F^-x) \wedge \exists\psi(x)$ . Intuitively, for each state subformula  $\exists\psi$ , the unique free variable (if any) refers to ancestors of the current node in the computation tree. The notion of well-formed formula of finitary hybrid  $\text{CTL}_{lp}^*$  is similar: the path quantifier  $\exists$  is replaced by its finitary version  $\exists_f$ . The well-formedness constraint ensures that a formula captures only branching regular requirements. As an example, the formula  $\exists F\downarrow x.G^-(\neg X^-T \rightarrow \forall F(x \wedge p))$  is *not* well-formed and requires that there is a level of the computation tree such that each node in the level satisfies  $p$ . This represents a well-known non-regular context-free branching requirement (see, e.g., [2]).

We first show that  $\text{HS}_{lp}$  can be translated into the well-formed fragment of hybrid  $\text{CTL}_{lp}^*$  (resp., well-formed fragment of finitary hybrid  $\text{CTL}_{lp}^*$ ). Then we show that this fragment is subsumed by  $\text{CTL}^*$  (resp., finitary  $\text{CTL}^*$ ).

► **Proposition 20.** *Given a  $\text{HS}_{lp}$  formula  $\varphi$ , one can construct in linear-time an equivalent well-formed sentence of hybrid  $\text{CTL}_{lp}^*$  (resp., finitary hybrid  $\text{CTL}_{lp}^*$ ).*

**Proof.** We focus on the translation from  $\text{HS}_{lp}$  into the well-formed fragment of hybrid  $\text{CTL}_{lp}^*$ . The translation from  $\text{HS}_{lp}$  into the well-formed fragment of finitary hybrid  $\text{CTL}_{lp}^*$  is similar. Let  $\varphi$  be a  $\text{HS}_{lp}$  formula. The desired hybrid  $\text{CTL}_{lp}^*$  sentence is given by  $\downarrow x.G f(\varphi, x)$ , where the mapping  $f(\varphi, x)$  is homomorphic with respect to the Boolean connectives, and is defined for the atomic propositions and the other modalities as follows ( $y$  is a fresh variable):

$$\begin{aligned} f(p, x) &= G^-(F^-x \rightarrow p), \\ f(\langle B \rangle \psi, x) &= X^-F^-(f(\psi, x) \wedge F^-x), \\ f(\langle \bar{B} \rangle \psi, x) &= (F^-x) \wedge \exists(XF f(\psi, x)), \\ f(\langle E \rangle \psi, x) &= \downarrow y.F^-(x \wedge XF\downarrow x.F(y \wedge f(\psi, x))), \\ f(\langle \bar{E} \rangle \psi, x) &= \downarrow y.F^-(XFx \wedge \downarrow x.F(y \wedge f(\psi, x))). \end{aligned}$$

Clearly  $\downarrow x.G f(\varphi, x)$  is well-formed. Moreover, let  $\mathcal{X}$  be a Kripke structure,  $[h, i]$  be an interval of positions,  $g$  be a valuation assigning to the variable  $x$  the position  $h$ , and  $\pi$  be an initial infinite path. By a straightforward induction on the structure of  $\varphi$ , one can show that  $\mathcal{X}, \pi, g, i \models f(\varphi, x)$  if and only if  $\mathcal{C}(\mathcal{X}), \mathcal{C}(\pi, h, i) \models_{\text{st}} \varphi$ , where  $\mathcal{C}(\pi, h, i)$  denotes the track of the computation tree  $\mathcal{C}(\mathcal{X})$  starting from  $\pi[0, h]$  and leading to  $\pi[0, i]$ . Hence  $\mathcal{X}$  is a model of  $\downarrow x.G f(\varphi, x)$  if for each initial track  $\rho$  of  $\mathcal{C}(\mathcal{X})$  we have  $\mathcal{C}(\mathcal{X}), \rho \models_{\text{st}} \varphi$ . ◀

Let  $\text{LTL}_p$  be the past extension of  $\text{LTL}$ , obtained by adding the past modalities  $X^-$  and  $U^-$ . By exploiting the well-formedness requirement and the well-known separation theorem for  $\text{LTL}_p$  over finite and infinite words [9] (i.e., any  $\text{LTL}_p$  formula can be effectively converted into an equivalent Boolean combination of  $\text{LTL}$  formulas and pure past  $\text{LTL}_p$  formulas), and proceeding by induction on the nesting depth of path quantifiers, we establish the following result (the proof can be found in [5]).

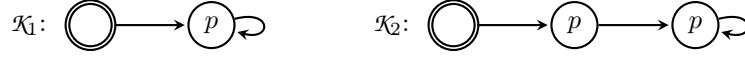
► **Proposition 21.** *The set of well-formed sentences of hybrid  $\text{CTL}_{lp}^*$  (resp., finitary hybrid  $\text{CTL}_{lp}^*$ ) has the same expressiveness as  $\text{CTL}^*$  (resp., finitary  $\text{CTL}^*$ ).*

By Corollary 19, and Propositions 20 and 21, we obtain the main result of Section 3.2.

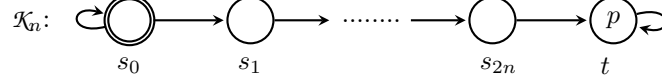
► **Theorem 22.**  $\text{CTL}^* \geq \text{HS}_{lp}$ . *Moreover,  $\text{HS}_{lp}$  is as expressive as finitary  $\text{CTL}^*$ .*

### 3.3 Expressiveness comparison of $\text{HS}_{in}$ , $\text{HS}_{st}$ and $\text{HS}_{lp}$

We first show that  $\text{HS}_{st}$  is *not* subsumed by  $\text{HS}_{lp}$ . As a matter of fact we show that  $\text{HS}_{st}$  is sensitive to unwinding, allowing us to discriminate finite Kripke structures having



■ **Figure 3** The Kripke structures  $\mathcal{K}_1$  and  $\mathcal{K}_2$ .



■ **Figure 4** The Kripke structure  $\mathcal{K}_n$  with  $n \geq 1$ .

the same computation tree (whereas they are indistinguishable by  $\text{HS}_{\text{lp}}$ ). In particular, let us consider the two finite Kripke structures  $\mathcal{K}_1$  and  $\mathcal{K}_2$  of Figure 3. Since  $\mathcal{K}_1$  and  $\mathcal{K}_2$  have the same computation tree, no HS formula  $\varphi$  under the computation-tree-based semantics can distinguish  $\mathcal{K}_1$  and  $\mathcal{K}_2$ , i.e.,  $\mathcal{K}_1 \models_{\text{lp}} \varphi$  iff  $\mathcal{K}_2 \models_{\text{lp}} \varphi$ . On the other hand, the requirement “each state reachable from the initial one where  $p$  holds has a predecessor where  $p$  holds as well” can be expressed, under the state-based semantics, by the HS formula  $\psi := \langle E \rangle (p \wedge \text{length}_1) \rightarrow \langle E \rangle (\text{length}_1 \wedge \langle \bar{A} \rangle (p \wedge \neg \text{length}_1))$ . Clearly  $\mathcal{K}_1 \models_{\text{st}} \psi$  but  $\mathcal{K}_2 \not\models_{\text{st}} \psi$ . Hence we obtain the following result.

► **Proposition 23.**  $\text{HS}_{\text{lp}} \not\cong \text{HS}_{\text{st}}$ .

Since  $\text{HS}_{\text{lp}}$  and finitary  $\text{CTL}^*$  have the same expressiveness (Theorem 22) and finitary  $\text{CTL}^*$  is subsumed by  $\text{HS}_{\text{st}}$  (Corollary 19), by Proposition 23 we obtain the following result.

► **Corollary 24.**  $\text{HS}_{\text{st}}$  is more expressive than  $\text{HS}_{\text{lp}}$ .

Let us now consider the CTL formula  $\forall G \exists F p$  asserting that from each state reachable from the initial one, it is possible to reach a state where  $p$  holds. It is well-known that this formula is not LTL-expressible. Thus, by Corollary 14, there is no equivalent HS formula under the trace-based semantics. On the other hand, the requirement  $\forall G \exists F p$  can be expressed under the state-based (resp., computation-tree-based) semantics by the HS formula  $\langle \bar{B} \rangle \langle E \rangle p$ .

► **Proposition 25.**  $\text{HS}_{\text{lin}} \not\cong \text{HS}_{\text{st}}$  and  $\text{HS}_{\text{lin}} \not\cong \text{HS}_{\text{lp}}$ .

Next we show that  $\text{HS}_{\text{lin}} \not\cong \text{HS}_{\text{st}}$  and  $\text{HS}_{\text{lin}} \not\cong \text{HS}_{\text{lp}}$ . To this end we establish the following.

► **Proposition 26.** The LTL formula  $Fp$  (equivalent to the CTL formula  $\forall Fp$ ) cannot be expressed in either  $\text{HS}_{\text{lp}}$  or  $\text{HS}_{\text{st}}$ .

We prove Proposition 26 for the state-based semantics (for the computation-tree-based semantics the proof is similar). We exhibit two families of Kripke structures  $(\mathcal{K}_n)_{n \geq 1}$  and  $(\mathcal{M}_n)_{n \geq 1}$  over  $\{p\}$  such that for all  $n \geq 1$  the LTL formula  $Fp$  distinguishes  $\mathcal{K}_n$  and  $\mathcal{M}_n$ , and for every HS formula  $\psi$  of size at most  $n$ ,  $\psi$  does *not* distinguish  $\mathcal{K}_n$  and  $\mathcal{M}_n$  under the state-based semantics. Hence the result follows. Fix  $n \geq 1$ . The Kripke structure  $\mathcal{K}_n$  is given in Figure 4. The Kripke structure  $\mathcal{M}_n$  is obtained from  $\mathcal{K}_n$  by setting as its initial state  $s_1$  instead of  $s_0$ . Formally,  $\mathcal{K}_n = (\{p\}, S_n, \delta_n, \mu_n, s_0)$  and  $\mathcal{M}_n = (\{p\}, S_n, \delta_n, \mu_n, s_1)$ , where  $S_n = \{s_0, s_1, \dots, s_{2n}, t\}$ ,  $\delta_n = \{(s_0, s_0), (s_0, s_1), \dots, (s_{2n-1}, s_{2n}), (s_{2n}, t), (t, t)\}$ ,  $\mu(s_i) = \emptyset$  for all  $0 \leq i \leq 2n$ , and  $\mu(t) = \{p\}$ . Clearly  $\mathcal{K}_n \not\models Fp$  and  $\mathcal{M}_n \models Fp$ .

We say that a HS formula  $\psi$  is *balanced* if, for each subformula  $\langle B \rangle \theta$  (resp.,  $\langle \bar{B} \rangle \theta$ ),  $\theta$  is of the form  $\theta_1 \wedge \theta_2$  with  $|\theta_1| = |\theta_2|$ . By using conjunctions of  $\top$ , one can trivially convert a HS formula  $\psi$  into a balanced HS formula which is equivalent to  $\psi$  under any of the considered HS semantic variants. Lemma 27 is proved in [5]: by such a lemma and the fact that, for each  $n \geq 1$ ,  $\mathcal{K}_n \not\models Fp$  and  $\mathcal{M}_n \models Fp$ , we get a proof of Proposition 26.

► **Lemma 27.** For all  $n \geq 1$  and balanced HS formulas  $\psi$  s.t.  $|\psi| \leq n$ ,  $\mathcal{K}_n \models_{\text{st}} \psi$  iff  $\mathcal{M}_n \models_{\text{st}} \psi$ .

By Propositions 25–26, we obtain the following result.

► **Corollary 28.**  $\text{HS}_{\text{lin}}$  and  $\text{HS}_{\text{st}}$  (resp.,  $\text{HS}_{\text{lp}}$ ) are expressively incomparable.

The proved results also allow us to establish the expressiveness relations between  $\text{HS}_{\text{st}}$ ,  $\text{HS}_{\text{lp}}$  and the standard branching temporal logics CTL and CTL\*.

► **Corollary 29.**

1.  $\text{HS}_{\text{st}}$  and CTL\* (resp., CTL) are expressively incomparable;
2.  $\text{HS}_{\text{lp}}$  and finitary CTL\* are less expressive than CTL\*;
3.  $\text{HS}_{\text{lp}}$  and CTL are expressively incomparable.

**Proof.** (Point 1) By Proposition 26 and the fact that CTL\* is not sensitive to unwinding. (Point 2) By Theorem 22,  $\text{HS}_{\text{lp}}$  is subsumed by CTL\*, and  $\text{HS}_{\text{lp}}$  and finitary CTL\* have the same expressiveness. Hence, by Proposition 26, the result follows. (Point 3) By Proposition 26, it suffices to show that there exists a  $\text{HS}_{\text{lp}}$  formula which cannot be expressed in CTL. Let us consider the CTL\* formula  $\varphi := \exists((p_1 \text{U} p_2) \vee (q_1 \text{U} q_2)) \text{U} r$  over the set of propositions  $\{p_1, p_2, q_1, q_2, r\}$ . It is shown in [8] that  $\varphi$  cannot be expressed in CTL. Clearly if we replace the path quantifier  $\exists$  in  $\varphi$  with the finitary path quantifier  $\exists_f$ , we obtain an equivalent formula of finitary CTL\*. Thus, since  $\text{HS}_{\text{lp}}$  and finitary CTL\* have the same expressiveness (Theorem 22), the result follows. ◀

## 4 Conclusions and future work

In this paper, we have studied three semantic variants, namely,  $\text{HS}_{\text{st}}$ ,  $\text{HS}_{\text{lp}}$ , and  $\text{HS}_{\text{lin}}$ , of the interval temporal logic HS, comparing their expressiveness to that of the standard temporal logics LTL, CTL, finitary CTL\*, and CTL\*. The reported results imply the decidability of the model checking problem for  $\text{HS}_{\text{lp}}$  and  $\text{HS}_{\text{lin}}$ ; the related complexity issues will be studied in the future work. Moreover, we shall investigate how the expressiveness changes when the homogeneity assumption is relaxed.

---

### References

- 1 J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. doi:10.1145/182.358434.
- 2 R. Alur, P. Cerný, and S. Zdancewic. Preserving secrecy under refinement. In *ICALP*, LNCS 4052, pages 107–118, 2006. doi:10.1007/11787006\_10.
- 3 P. Blackburn and J. Seligman. What are hybrid languages? In *AiML*, pages 41–62, 1998.
- 4 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Interval Temporal Logic Model Checking: the Border Between Good and Bad HS Fragments. In *IJCAR*, LNAI 9706, pages 389–405, 2016. doi:10.1007/978-3-319-40229-1\_27.
- 5 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Interval vs. point temporal logic model checking: an expressiveness comparison. Technical report, Univ. of Udine, Italy, 2016. URL: <http://www.uniud.it/it/ateneo-uniud/ateneo-uniud-organizzazione/dipartimenti/dima/ricerca/pubblicazioni/preprints/3.2016>.
- 6 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Model Checking the Logic of Allen’s Relations Meets and Started-by is  $\mathbf{P}^{\text{NP}}$ -Complete. In *GandALF*, pages 76–90, 2016. doi:10.4204/EPTCS.226.6.

- 7 D. Bresolin, D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. The dark side of interval temporal logic: marking the undecidability border. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):41–83, 2014. doi:10.1007/s10472-013-9376-4.
- 8 E. A. Emerson and J. Y. Halpern. “Sometimes” and “not never” revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986. doi:10.1145/4904.4999.
- 9 D. M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Temporal Logic in Specification*, LNCS 398, pages 409–448, 1987. doi:10.1007/3-540-51803-7\_36.
- 10 J. Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962, 1991. doi:10.1145/115234.115351.
- 11 H. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, 1968.
- 12 O. Kupferman, A. Pnueli, and M. Y. Vardi. Once and for all. *J. Comput. Syst. Sci.*, 78(3):981–996, 2012. doi:10.1016/j.jcss.2011.08.006.
- 13 K. Lodaya. Sharpening the undecidability of interval temporal logic. In *ASIAN*, LNCS 1961, pages 290–298, 2000. doi:10.1007/3-540-44464-5\_21.
- 14 A. Lomuscio and J. Michaliszyn. An epistemic Halpern-Shoham logic. In *IJCAI*, pages 1010–1016, 2013.
- 15 A. Lomuscio and J. Michaliszyn. Decidability of model checking multi-agent systems against a class of EHS specifications. In *ECAI*, pages 543–548, 2014. doi:10.3233/978-1-61499-419-0-543.
- 16 A. Lomuscio and J. Michaliszyn. Model checking multi-agent systems against epistemic HS specifications with regular expressions. In *KR*, pages 298–308, 2016.
- 17 J. Marcinkowski and J. Michaliszyn. The undecidability of the logic of subintervals. *Fundamenta Informaticae*, 131(2):217–240, 2014. doi:10.3233/FI-2014-1011.
- 18 A. Molinari, A. Montanari, A. Murano, G. Perelli, and A. Peron. Checking interval properties of computations. *Acta Informatica*, 53(6):587–619, 2016. doi:10.1007/s00236-015-0250-1.
- 19 A. Molinari, A. Montanari, and A. Peron. Complexity of ITL model checking: some well-behaved fragments of the interval logic HS. In *TIME*, pages 90–100, 2015. doi:10.1109/TIME.2015.12.
- 20 A. Molinari, A. Montanari, and A. Peron. A model checking procedure for interval temporal logics based on track representatives. In *CSL*, pages 193–210, 2015. doi:10.4230/LIPIcs.CSL.2015.193.
- 21 A. Molinari, A. Montanari, A. Peron, and P. Sala. Model Checking Well-Behaved Fragments of HS: the (Almost) Final Picture. In *KR*, pages 473–483, 2016.
- 22 B. Moszkowski. *Reasoning About Digital Circuits*. PhD thesis, Stanford University, CA, USA, 1983.
- 23 A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977. doi:10.1109/SFCS.1977.32.
- 24 I. Pratt-Hartmann. Temporal prepositions and their logic. *Artificial Intelligence*, 166(1-2):1–36, 2005. doi:10.1016/j.artint.2005.04.003.
- 25 P. Roeper. Intervals and tenses. *Journal of Philosophical Logic*, 9:451–469, 1980.
- 26 M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for concurrency*, pages 238–266. Springer, 1996. doi:10.1007/3-540-60915-6\_6.
- 27 Y. Venema. Expressiveness and completeness of an interval tense logic. *Notre Dame Journal of Formal Logic*, 31(4):529–547, 1990. doi:10.1305/ndjfl/1093635589.
- 28 T. Wilke. Classifying discrete temporal properties. In *STACS*, LNCS 1563, pages 32–46, 1999. doi:10.1007/3-540-49116-3\_3.

# Model Checking Population Protocols\*

Javier Esparza<sup>1</sup>, Pierre Ganty<sup>†2</sup>, Jérôme Leroux<sup>3</sup>, and  
Rupak Majumdar<sup>4</sup>

1 Technical University of Munich, Germany

2 IMDEA Software Institute, Madrid, Spain

3 LaBRI, CNRS, Univ. Bordeaux, Talence, France

4 MPI-SWS, Kaiserslautern, Germany

---

## Abstract

Population protocols are a model for parameterized systems in which a set of identical, anonymous, finite-state processes interact pairwise through rendezvous synchronization. In each step, the pair of interacting processes is chosen by a random scheduler. Angluin et al. (PODC 2004) studied population protocols as a distributed computation model. They characterized the computational power in the limit (semi-linear predicates) of a subclass of protocols (the well-specified ones). However, the modeling power of protocols goes beyond computation of semi-linear predicates and they can be used to study a wide range of distributed protocols, such as asynchronous leader election or consensus, stochastic evolutionary processes, or chemical reaction networks. Correspondingly, one is interested in checking specifications on these protocols that go beyond the well-specified computation of predicates.

In this paper, we characterize the decidability frontier for the model checking problem for population protocols against probabilistic linear-time specifications. We show that the model checking problem is decidable for qualitative objectives, but as hard as the reachability problem for Petri nets – a well-known hard problem without known elementary algorithms. On the other hand, model checking is undecidable for quantitative properties.

**1998 ACM Subject Classification** D.2.4 Software/Program Verification, F.1.1 Models of Computation

**Keywords and phrases** parameterized systems, population protocols, probabilistic model checking, probabilistic linear-time specifications, decidability

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.27

## 1 Introduction

Population protocols [3, 4, 6] are a model of distributed computation by anonymous, identical, finite-state agents interacting in pairs. Given an initial configuration – an initial distribution of agents specifying the number of agents at each state – a random scheduler repeatedly chooses a pair of processes and one of the interactions enabled by their current states. This naturally assigns to each initial configuration a semantics in terms of a finite Markov Chain. Thus, the protocol defines infinitely many finite Markov Chains.

Population protocols were originally introduced to study which predicates on an initial configuration of agents (like, for example, “does the configuration contain more processes

---

\* This research was sponsored in part by the ERC Synergy project ImPACT.

† Pierre Ganty has been supported by the Madrid Regional Government project S2013/ICE-2731, *N-Greens Software – Next-Generation Energy-Efficient Secure Software*, and the Spanish Ministry of Economy and Competitiveness project No. TIN2015-71819-P, *RISCO – RIGorous analysis of SOphisticated COncurrent and distributed systems*.





of type A than of type B”) can be computed by the agents themselves in a distributed way. For this, Angluin et al. [3] introduced a definition of computation by consensus: a configuration computes a value if all its agents eventually converge to that value (represented by a particular state or set of states) with probability 1.

Population protocols have been used to model systems beyond their initial motivation in distributed computing. In particular, they can also model trust propagation [15], evolutionary dynamics [23], or chemical reaction systems [24, 25]. These systems do not aim at the computation of predicates, or they do not compute in the way defined by Angluin et al. [3]. With more diverse applications of population protocols comes also new properties one would like to reason about. For instance, Delporte-Gallet et al. [14] studied privacy in population protocols. They proved (by hand) different properties of specific protocols, like “the system can reach a good configuration without any interaction involving a distinguished agent  $p_0$ ”. Another example is the work of Clement et al. [13] who use the probabilistic model checker PRISM to check properties including the aforementioned convergence. However, PRISM is a finite state model checker and therefore the verification was limited to a finite number of individual initial configurations.

In this paper, we solve the general model checking problem for population protocols against probabilistic linear-time (LTL) specifications. As opposed to previous work, we can reason fully automatically starting from sets of initial configurations which can be infinite as long as they are Presburger definable. We show that the qualitative problem (i.e., deciding if the formula holds with probability 1) is decidable, but as hard as the reachability problem for Petri nets (*reachability hard*). The quantitative problem (deciding if the property holds with at least a given probability) is undecidable. In particular, our proof of undecidability for quantitative LTL uses a novel simulation of counter machines by Petri nets with arbitrarily small error. Additionally, we prove (§4) undecidability for both the qualitative problem for broadcast protocols (a stronger model where interactions involve all processes, not just a fixed number) and a natural variant of LTL specifications where atomic propositions are defined on configurations rather than actions.

From the verification point of view, in this paper we study the decidability of parameterized verification for a class of probabilistic systems. Our work can be seen as an extension of the approach of German and Sistla [20] to probabilistic verification. Our results establish certain decidability frontiers for parameterized verification of probabilistic programs, an area of increasing interest [2, 10, 11, 1, 12]. The question whether all instances of a parameterized system satisfy a property with probability 1 was also studied by Pnueli and Zuck with different co-authors [26, 7] and also by Esparza et al. [17]. The emphasis in these papers was on deductive proof systems, and they do not contain decidability results.

## 2 Definitions and Examples

A *multiset* on a finite non-empty set  $E$  is a mapping  $M: E \rightarrow \mathbb{N}$ . Given  $e \in E$ , let  $M(e)$  denote the number of elements of type  $e$  in the multiset  $M$ . Operations on  $\mathbb{N}$ , like addition, subtraction, or comparison are implicitly defined on multisets by defining the operation componentwise. The set of all multisets over  $E$  is denoted  $\mathbb{N}^E$ . Given  $e \in E$ , we denote by  $\mathbf{e} \in \mathbb{N}^E$  the multiset consisting of one occurrence of element  $e$ , that is, the multiset satisfying  $\mathbf{e}(e) = 1$  and  $\mathbf{e}(e') = 0$  for every  $e' \neq e$ . We write  $\{x, y, y, z\}$  to denote the multiset  $\mathbf{x} + \mathbf{y} + \mathbf{y} + \mathbf{z}$ . Every set  $S$  on  $E$  is also a multiset which maps  $E$  into  $\{0, 1\}$ . The *size* of a multiset  $M \in \mathbb{N}^E$ , denoted  $|M|$ , is defined as  $\sum_{e \in E} M(e)$ .

A set of multisets  $\mathcal{M} \subseteq \mathbb{N}^E$  is said to be *Presburger* if it can be denoted by a formula in *Presburger arithmetic*, i.e., in the first-order theory of addition  $FO(\mathbb{N}, +)$ . A *population*  $P$



on  $E$  is a multiset on  $E$  with two or more elements:  $P \in \mathbb{N}^E$  and  $|P| \geq 2$ .<sup>1</sup> The set of all populations on  $E$  is denoted by  $\text{Pop}(E)$ .

## 2.1 Labeled Population Protocols

A (labeled) *protocol scheme*  $\mathcal{A} = (Q, \Sigma, \Delta)$  consists of a finite non-empty set  $Q$  of states, a finite set  $\Sigma$  of action labels, and a set  $\Delta \subseteq Q^2 \times \Sigma \times Q^2$ . If  $(q_1, q_2, a, q'_1, q'_2) \in \Delta$ , we write  $(q_1, q_2) \xrightarrow{a} (q'_1, q'_2)$  and call it an  $a$ -labeled *transition*. The populations of  $\text{Pop}(Q)$  are called *configurations*. Intuitively, a configuration  $C$  describes a collection of identical finite-state *agents* with  $Q$  as set of states, containing  $C(q)$  agents in state  $q$  for every  $q \in Q$ . Pairs of agents interact using labeled transitions from  $\Delta$ . Formally, given two configurations  $C$  and  $C'$  and a transition  $(q_1, q_2) \xrightarrow{a} (q'_1, q'_2)$ , we write  $C \xrightarrow{a} C'$  and call it a *step* if

$$C \geq (\mathbf{q}_1 + \mathbf{q}_2) \text{ holds, and } C' = C - (\mathbf{q}_1 + \mathbf{q}_2) + (\mathbf{q}'_1 + \mathbf{q}'_2) .$$

We write  $C \xrightarrow{w} C'$  for a sequence  $w = a_1 \dots a_k \in \Sigma^*$  of labels if there exists a sequence  $C_0, \dots, C_k$  of configurations satisfying  $C = C_0 \xrightarrow{a_1} C_1 \dots \xrightarrow{a_k} C_k = C'$ . We call this sequence a *finite execution*. The notation  $C \xrightarrow{w}$  for an infinite sequence  $w$  and the notion of an infinite execution are defined analogously. The  $\omega$ -language of  $\mathcal{A}$  from configuration  $C$ , denoted  $L(\mathcal{A}, C)$ , is the set  $\{w \in \Sigma^\omega \mid C \xrightarrow{w}\} \subseteq \Sigma^\omega$ .

In what follows we assume every protocol scheme has its set of states  $Q$  and transitions  $\Delta$  satisfying the following condition: for every  $q_1, q_2$  in  $Q$ , there exists  $q'_1, q'_2$  and label  $a$  such that  $(q_1, q_2, a, q'_1, q'_2) \in \Delta$ . It follows that every configuration enables a transition.

The *configuration graph* of a protocol scheme  $\mathcal{A}$  is the infinite labeled, directed graph  $(\text{Pop}(Q), \Sigma, \rightarrow)$  having the populations over  $Q$  as nodes and labeled edges  $(C, a, C')$  whenever  $C \xrightarrow{a} C'$  holds. Consider the partition  $\{\text{Pop}(Q)_i\}_{i \geq 2}$  of  $\text{Pop}(Q)$ , where  $\text{Pop}(Q)_i = \{C \in \text{Pop}(Q) \mid |C| = i\}$ . (Note that  $i$  starts at 2 because every population contains at least two agents.) Since interactions do not create or destroy agents, the set  $\{\rightarrow_i\}_{i \geq 2}$ , where  $\rightarrow_i = \rightarrow \cap \text{Pop}(Q)_i \times \Sigma \times \text{Pop}(Q)_i$ , is also a partition of  $\rightarrow$ . Therefore  $(\text{Pop}(Q), \rightarrow)$  consists of the infinitely many disjoint and finite subgraphs  $\{(\text{Pop}(Q)_i, \Sigma, \rightarrow_i)\}_{i \geq 2}$ .

A *strongly connected component* (SCC) of the configuration graph is a maximal set of mutually reachable configurations. An SCC is a *bottom SCC* if it is closed under reachability, i.e., if  $C$  belongs to the SCC and  $C'$  is reachable from  $C$ , then  $C'$  also belongs to the SCC. A configuration is a *bottom configuration* if it belongs to a bottom SCC of the graph.

We define a *population protocol* as a protocol scheme equipped with a possibly infinite Presburger set  $\mathcal{I}$  of *initial configurations* and denote it as a pair  $(\mathcal{A}, \mathcal{I})$ . The original paper [3] introducing population protocols considered, instead of Presburger sets of initial configurations, a restricted class called simple sets defined by means of *input variables* – a subset of the states of the protocol scheme. Given a set  $S \subseteq Q$  of input variables, the set  $\mathcal{I}$  of initial configurations thereof is given by  $\{C \in \text{Pop}(Q) \mid \forall q \in Q: q \in S \vee C(q) = 0\}$ . When  $\mathcal{I}$  is definable using *input variables* then  $\mathcal{I}$  is said to be *simple*.<sup>2</sup>

The complexity of parameterized verification can differ based on the presence or absence of a leader, a specific agent starting in a given state. Simple sets of initial configurations are essential to define *leaderless protocols*: protocols with no distinguished leader agent. (If we want to have a distinguished leader we can design a protocol whose set of states is the disjoint

<sup>1</sup> Since, as we will see later, the atomic semantic step of population protocols is a pairwise interaction, we require at least two agents in every population.

<sup>2</sup> To sum up, unless we state  $\mathcal{I}$  is simple then it is assumed to be described by a Presburger formula.

union of two sets  $Q_l \cup Q_a$  of leader and agent states, and having a distinguished initial state  $q_0 \in Q_l$  for the leader. Sets  $\mathcal{I}$  containing only configurations  $C$  satisfying  $C(q_0) = 1$  ensure that there is a unique leader.)

We give a population protocol a semantics as an infinite family of finite-state Markov Chains, one for each initial configuration. We assume that, given a configuration  $C$ , a probabilistic scheduler picks a pair of agents of  $C$  uniformly at random, and then picks one of the transitions they can execute according to some fixed probability distribution satisfying two properties: the probability of a transition depends only on the current states of the agents, and every transition has nonzero probability. This associates with each step  $C \xrightarrow{a} C'$  a probability. Since  $C \xrightarrow{a} C'$  implies  $|C| = |C'|$ , the number of configurations reachable from any configuration  $C$  is finite. Thus, for every  $C$ , the Markov Chain rooted at  $C$  has finitely many states.

► **Example 1.** Consider a protocol with two states  $q_1, q_2$  and a configuration  $C = (C(q_1), C(q_2)) = (1, 4)$ . The scheduler picks two agents at state  $q_1$  with probability 0, two agents at states  $q_1$  and  $q_2$  with probability  $2/5$ , and two agents at state  $q_2$  with probability  $3/5$ . If the protocol has three transitions  $(q_1, q_2) \xrightarrow{a} (q_2, q_2)$ ,  $(q_1, q_2) \xrightarrow{a} (q_1, q_1)$ ,  $(q_1, q_2) \xrightarrow{b} (q_2, q_2)$ , each with probability  $1/3$ , then the steps  $(1, 4) \xrightarrow{a} (0, 5)$ ,  $(1, 4) \xrightarrow{a} (2, 3)$  and  $(1, 4) \xrightarrow{b} (0, 5)$  have probability  $2/15$  each. The probability of doing an  $a$  from  $(1, 4)$  is  $4/15$ , and the probability of moving from  $(1, 4)$  to  $(0, 5)$  by means of some action is also  $4/15$ .

Once the set of initial configurations  $\mathcal{I}$  is fixed, we can talk about the probability of a measurable set of infinite paths of a Markov Chain rooted at some  $C \in \mathcal{I}$ . We write  $\Pr[\mathcal{A}, C \models \mathcal{E}]$  to denote the probability that the stochastic process assigns to an event  $\mathcal{E}$  for some  $C \in \mathcal{I}$ . Later, events will correspond to formulas of the linear temporal logic.

► **Example 2 (Computation by consensus).** Angluin et al. [3] study protocols as a computation model. In their model, each state has an *output*, either 0 or 1. A protocol is well-specified if for every initial configuration, all agents eventually output the same value  $b = 0, 1$  and stay committed to this value forever. Being well-specified further requires that the commitment of a population of agents to a value exclusively depends on their initial distribution. A well-specified protocol computes a predicate over its input variables: for each initial configuration, the predicate has value 0 (resp. 1) iff agents commit to 0 (resp. output 1) with probability 1.

► **Example 3 (Birth-death processes).** A Moran process [23] is a stochastic process for evolution in a population with  $N - 1$  residents and a mutant. In each step, one agent is randomly chosen for reproduction and one agent is randomly chosen for death. In the next step, the agent who dies is replaced with a copy of the agent that reproduces, therefore keeping the size of the population constant. In this setting, an interesting question is fixation: whether a single mutant can eventually replace all residents.

### Probability vs. fairness

Instead of the probabilistic semantics, the semantics of population protocols is usually defined using fairness [3, 6]. An execution is *fair* if it is infinite and for every configuration  $C$  appearing in it infinitely often, and for every possible labelled step  $C \xrightarrow{a} C'$ , the step also appears infinitely often in the execution. We show later in Proposition 7 that the fair semantics and the probability semantics are indistinguishable for the questions we are studying: For every initial configuration  $C$  and every property  $\varphi$  expressible in LTL, some fair execution starting at  $C$  satisfies  $\varphi$  iff the set of executions of the protocol (fair or not) starting at  $C$  and satisfying  $\varphi$  has *positive* probability.

## 2.2 Probabilistic Linear Temporal Logic

Let  $\Sigma$  be a finite set of actions labels. The formulas of linear temporal logic (LTL) are defined by the grammar

$$\varphi ::= a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi \mid \varphi \mathbf{W}\varphi \quad \text{where } a \in \Sigma.$$

The semantics of LTL formulas are given in the usual way over traces [9]: an LTL formula  $\varphi$  defines an  $\omega$ -language  $L(\varphi) \subseteq \Sigma^\omega$ . We define the following derived symbols:  $\neg a \equiv \bigvee_{\sigma \in \Sigma \setminus \{a\}} \sigma$ ,  $\text{true} \equiv \bigvee_{\sigma \in \Sigma} \sigma$ ,  $\mathbf{F}\varphi \equiv \text{true}\mathbf{U}\varphi$  and  $\mathbf{G}\varphi \equiv \varphi\mathbf{W}\text{false}$ . The derived symbols can be eliminated with at most a polynomial cost in the size of the formula.

**Qualitative and quantitative model checking.** Let us now introduce the probabilistic interpretation for LTL. Given a configuration  $C$ , we say that  $(\mathcal{A}, C)$  *satisfies the LTL formula  $\varphi$  with probability  $p$*  if  $\Pr[\mathcal{A}, C \models \varphi] = p$ . The *(qualitative) model checking problem* consists of, given a population protocol  $(\mathcal{A}, \mathcal{I})$ , and an LTL formula  $\varphi$ , deciding if  $\Pr[\mathcal{A}, C \models \varphi] = 1$  for all  $C \in \mathcal{I}$ . We often work with the complement problem (deciding if  $\Pr[\mathcal{A}, C \models \neg\varphi] > 0$  for some  $C \in \mathcal{I}$ ). Abusing language, we also call it the *model checking problem*; the context should determine which problem we refer to. The *quantitative model checking problem* has an additional input  $p$  between 0 and 1 and asks whether  $\Pr[\mathcal{A}, C \models \varphi] \geq p$  for all  $C \in \mathcal{I}$ .

► **Example 4** (Cont'd from Ex. 2). Let  $\Sigma$  be the set of all actions and let  $\Sigma_{ij}$ , for  $i, j \in \{0, 1\}$ , be the set of all transitions in which the output of the first process is  $i$  and the second process is  $j$ . Then, if the property  $\Pr[\mathcal{A}, C \models \mathbf{F}(\mathbf{G}\Sigma_{00} \vee \mathbf{G}\Sigma_{11})] = 1$  holds for all  $C \in \mathcal{I}$ , we have that from any initial configuration, an execution of the protocol stabilizes to an output (0 or 1) with probability 1. Being well-specified actually requires more: all the executions starting at a configuration must converge *to the same value* with probability 1. This property can be expressed as a LTL formula but on a *modified* protocol instead of the original one. Intuitively, the modified protocol is the result of running two copies of the protocol side-by-side as we explained in a previous work [18].

Delporte-Gallet et al. [14] study which predicates can be computed by privacy-preserving protocols that do not reveal information on the initial configuration to a curious adversary. They identify a sufficient condition for a protocol to be privacy preserving, expressed as the conjunction of two properties (plus two other minor conditions). The first one is the existence from each initial configuration of an execution leading to a given set of configurations  $\mathcal{G}$ , and containing no interaction involving a distinguished agent. For the cases studied in the paper  $\mathcal{G}$  can be expressed by an LTL-formula  $\varphi$ , and the property can be reduced to the model-checking problem for the formula  $(\bigvee_{a \in A} a)\mathbf{U}\varphi$  for a suitable  $A \subseteq \Sigma$ . The second one, called  *$\mathcal{G}$ -imitability*, requires that for every action  $a$  of the distinguished agent and for every configuration of  $\mathcal{G}$  there is a finite execution leading to a configuration of  $\mathcal{G}$  and containing exactly one occurrence of the action. This can be verified by taking  $\mathcal{G}$  as set of initial configurations and checking the formula  $\bigwedge_{a \in \Sigma} (\neg a \mathbf{U}(a \wedge \mathbf{X}(\neg a \mathbf{U}\varphi)))$ .

► **Example 5.** For the Moran process, the property that a mutant takes over the population is  $\mathbf{F}\mathbf{G}\Sigma_0$ , where  $\Sigma_0$  is the set of actions where a mutant reproduces.

## 2.3 Deterministic Rabin Automata

A *deterministic Rabin automaton* (DRA)  $\mathcal{R} = (Q, \Sigma, \delta, q_0, \mathcal{F})$  consists of a finite set  $Q$  of states including an initial state  $q_0$ , an alphabet  $\Sigma$ , a transition function  $\delta : Q \times \Sigma \rightarrow Q$ , and an acceptance condition  $\mathcal{F} \subseteq 2^Q \times 2^Q$ .

An input for  $\mathcal{R}$  is an infinite string in  $\Sigma^\omega$ . The run of  $\mathcal{R}$  on  $w = w_0w_1\dots \in \Sigma^\omega$  is an infinite sequence  $\rho = r_0r_1\dots$  of states in  $Q$  such that  $r_0 = q_0$ , the initial state, and for each  $i \geq 0$ , we have  $r_{i+1} = \delta(r_i, w_i)$ . Since  $\mathcal{R}$  is deterministic, we can extend the transition function  $\delta$  to finite words as follows:  $\delta^*(q, \varepsilon) = q$  for every  $q \in Q$  and  $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$ . Let  $\text{Inf}(\rho)$  be the set of states that appear infinitely often in  $\rho$ .

Let  $\mathcal{F} = \{\langle F_1, G_1 \rangle, \dots, \langle F_k, G_k \rangle\}$  be the set of *Rabin pairs*. A run  $\rho$  is accepting if there exists an  $i \in \{1, \dots, k\}$  such that  $\text{Inf}(\rho) \cap F_i = \emptyset$  and  $\text{Inf}(\rho) \cap G_i \neq \emptyset$ ; that is, no state from  $F_i$  is seen infinitely often and some state from  $G_i$  is seen infinitely often. A word  $w$  is accepted by  $\mathcal{R}$  if the unique run of  $\mathcal{R}$  on  $w$  is accepting. The language of  $\mathcal{R}$ , written  $L(\mathcal{R})$ , is the set of all words accepted by  $\mathcal{R}$ .

For each LTL formula  $\varphi$ , it is well-known that there is a DRA  $\mathcal{R}_\varphi$  of size at most doubly exponential in the size of  $\varphi$  such that  $L(\varphi) = L(\mathcal{R}_\varphi)$ .

## 2.4 Labeled Petri Nets

A *labeled Petri net*  $N = (P, T, F, \Sigma, \lambda)$  consists of a finite set  $P$  of *places*, a finite set  $T$  of *transitions*, a *flow function*  $F: (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ , an alphabet  $\Sigma$  of *actions* and a *labeling function*  $\lambda: T \rightarrow \Sigma$ . Abusing language, we also use  $\lambda$  to denote the homomorphic extension of the labeling function to  $T^* \cup T^\omega \rightarrow \Sigma^* \cup \Sigma^\omega$ . Given a transition  $t \in T$ , the multiset  $\bullet t$  of *input places* of  $t$  is defined by  $\bullet t(p) = F(p, t)$ , and the multiset  $t^\bullet$  of *output places* by  $t^\bullet(p) = F(t, p)$ . A Petri net is a labeled Petri net without a labeling function and alphabet of actions.

A *marking*  $M$  of a net  $N$  is a multiset of places. Given a place  $p$ , we say that  $M$  puts  $M(p)$  *tokens* in  $p$ . A transition  $t \in T$  is *enabled at* a marking  $M$ , written  $M[t]$ , if  $\bullet t \leq M$ . A transition  $t$  enabled at  $M$  can *fire*, yielding the marking  $M' = M - \bullet t + t^\bullet$ . We write this fact as  $M[t]M'$ . We extend enabledness and firing to sequences of transitions as follows. Let  $\sigma = t_1\dots t_k$  be a finite sequence of transitions  $t_j \in T$ . We write  $M[\sigma]M'$  and call it a *firing sequence* if there exists a sequence  $M_0, \dots, M_k$  of markings such that  $M = M_0[t_1]M_1 \dots [t_k]M_k = M'$ . In that case, we say that  $M'$  is *reachable from*  $M$  and denote by  $\text{Reach}(N, M)$  the set of markings reachable from  $M$ . Given an infinite sequence  $\sigma = t_1t_2\dots$ , we write  $M[\sigma]$  if, and only if, there exists an infinite sequence  $M_0, M_1, \dots$  of markings such that  $M = M_0$  and  $M_i[t_{i+1}]M_{i+1}$  for every  $i \geq 0$ . Finally, given  $w \in \Sigma^* \cup \Sigma^\omega$ , we write  $M[w]$  and  $M[w]M'$  if  $M[\sigma]$  and  $M[\sigma]M'$  for some sequence  $\sigma$  of transitions such that  $\lambda(\sigma) = w$ .

## 3 Model-checking LTL

In Section 3.1 we first show that the qualitative model checking problem is decidable. We then prove that it is as hard as the reachability problem for Petri nets, even for simple sets of initial configurations (that is, for leaderless protocols) and for the formula  $\mathbf{FG}a$ . Finally, in Section 3.2, we show the quantitative version is undecidable.

### 3.1 The Model Checking Problem Is Decidable but Reachability Hard

Our solution to the model checking problem is based on a product construction that, given a protocol scheme  $\mathcal{A} = (Q, \Sigma, \Delta)$  and a DRA  $\mathcal{R} = (Q', \Sigma, \delta, q'_0, \mathcal{F})$  such that  $Q \cap Q' = \emptyset$ , produces a labeled Petri net  $N(\mathcal{A}, \mathcal{R}) = (P, T, F, \Sigma, \lambda)$ , defined as follows:

- $P = Q \cup Q'$ .
- $T$  contains a transition  $t_{\delta_{\mathcal{A}, q}}$  for each  $\delta_{\mathcal{A}} = (q_1, q_2) \xrightarrow{a} (q_3, q_4) \in \Delta$  and  $q \in Q'$ .

- For each  $t_{\delta_{\mathcal{A}},q} \in T$  with  $\delta_{\mathcal{A}} = (q_1, q_2) \xrightarrow{a} (q_3, q_4) \in \Delta$ :  $\bullet(t_{\delta_{\mathcal{A}},q}) = \{q_1, q_2, q\}$  and  $(t_{\delta_{\mathcal{A}},q})^\bullet = \{q_3, q_4, \delta(q, a)\}$ . Further,  $\lambda(t_{\delta_{\mathcal{A}},q}) = a$ .

It follows immediately from the definition of  $N(\mathcal{A}, \mathcal{R})$  that  $(C+\mathbf{q}') [w] M$  for some marking  $M$  and word  $w \in \Sigma^*$  iff  $M = C' + \mathbf{q}''$ ,  $C \xrightarrow{w} C'$ , and  $\delta^*(q', w) = q''$ , that is,  $N(\mathcal{A}, \mathcal{R})$  captures the action-based synchronized product of  $\mathcal{A}$  and  $\mathcal{R}$ .

A marking  $M$  of  $N(\mathcal{A}, \mathcal{R})$  is *proper* if  $M = C + \mathbf{q}$  where  $C$  is a configuration of  $\mathcal{A}$  and  $q$  is a state of  $\mathcal{R}$ . In particular, every marking reachable from a proper marking is proper. The *proper reachability graph* of  $N(\mathcal{A}, \mathcal{R})$  contains the proper markings of  $N(\mathcal{A}, \mathcal{R})$  as nodes and the steps  $(C+\mathbf{q}') [a] (C'+\mathbf{q}'')$  for  $a \in \Sigma$  as edges. We say that an SCC  $\mathcal{S}$  of the reachability graph of  $N(\mathcal{A}, \mathcal{R})$  is *accepting* if there is a Rabin pair  $\langle F, G \rangle$  of  $\mathcal{R}$  such that  $q' \in F$  for no marking  $(C+\mathbf{q}') \in \mathcal{S}$ , and  $q' \in G$  for some marking  $(C+\mathbf{q}') \in \mathcal{S}$ .

Next, Proposition 6 reduces the qualitative model checking problem to a topological problem about the (typically infinitely many) SCCs of  $N(\mathcal{A}, \mathcal{R})$ .

► **Proposition 6.** *Given a configuration  $C$  of  $\mathcal{A}$  and a Rabin automaton  $\mathcal{R}_\varphi$  for a LTL formula  $\varphi$ :  $\Pr[\mathcal{A}, C \models \varphi] > 0$  iff some bottom SCC of the proper reachability graph of  $N(\mathcal{A}, \mathcal{R}_\varphi)$  is accepting and reachable from  $(C+\mathbf{q}'_0)$ .*

**Proof.** For simplicity, we conduct the proof for the special case in which every transition of  $\mathcal{A}$  is labeled with a different action. The extension to the general case is straightforward.

Observe that, since the interactions on  $\mathcal{A}$  do not change the size of a configuration, for every two markings  $(C_1+\mathbf{q}_1), (C_2+\mathbf{q}_2)$  of the proper reachability graph of  $N(\mathcal{A}, \mathcal{R}_\varphi)$  we have  $|C_1| = |C_2|$ . Since  $\mathcal{R}_\varphi$  has finitely many states, the number of markings reachable from  $(C_1+\mathbf{q}_1)$  is at most  $K(C_1) := (n^{|C_1|+1}) \cdot m$ , where  $n$  and  $m$  are the number of states of  $\mathcal{A}$  and  $\mathcal{R}_\varphi$ , respectively.

We introduce the following notation. Let  $\tau = C \xrightarrow{a_1 \dots a_n} C_n$  be a finite execution of  $\mathcal{A}$ . As usual, the *cylinder*  $Cyl(\tau)$  denotes all the infinite executions of  $\mathcal{A}$  starting with  $\tau$ , and its probability  $\Pr(Cyl(\tau))$  is the product of the probabilities of the steps  $C \xrightarrow{a_1} C_1, \dots, C_{n-1} \xrightarrow{a_n} C_n$ . Since  $\mathcal{R}_\varphi$  is complete and deterministic, the unique state  $q'_n = \delta^*(q'_0, a_1 \dots a_n)$  is such that  $(C+\mathbf{q}'_0) [a_1 \dots a_n] (C_n+\mathbf{q}'_n)$  is a firing sequence of  $N(\mathcal{A}, \mathcal{R}_\varphi)$ . We call the run of  $\mathcal{R}_\varphi$  the *matching run* of  $\tau$ , and denote it by  $\bar{\tau}$ . Further, we denote the firing sequence by  $(\tau, \bar{\tau})$ . We extend the notation to infinite executions, runs, and firing sequences.

We first prove a preliminary claim. An infinite execution  $\sigma$  of  $\mathcal{A}$  starting at a configuration  $C$  satisfies the following property with probability 1: the infinite firing sequence  $(\sigma, \bar{\sigma})$  of  $N(\mathcal{A}, \mathcal{R}_\varphi)$  from the marking  $(C+\mathbf{q}'_0)$  eventually reaches a bottom SCC of the proper reachability graph of  $N(\mathcal{A}, \mathcal{R}_\varphi)$ , and visits all its markings infinitely often.

For the first part, observe that for every finite prefix  $\sigma_1$  of  $\sigma$  there is, by definition, a finite execution  $\sigma_2$  of length at most  $K(C)$  such that the marking reached by  $(\sigma_1\sigma_2, \bar{\sigma}_1\bar{\sigma}_2)$  belongs to a bottom SCC of the proper reachability graph of  $N(\mathcal{A}, \mathcal{R}_\varphi)$ . Therefore, there is a bound  $p(C) > 0$ , depending only on  $\sigma_2$  and in particular on  $C$ , such that the probability that for  $\sigma \in Cyl(\sigma_1)$  the firing sequence  $(\sigma, \bar{\sigma})$  reaches a bottom marking is at least  $p(C)$ . By elementary probability theory, the probability of the infinite execution  $\sigma$  such that  $(\sigma, \bar{\sigma})$  eventually visits a bottom SCC is equal to 1. For the second part, assume that  $\sigma$  has a prefix  $\tau$  such that  $(\tau, \bar{\tau})$  leads to a bottom SCC, say  $\mathcal{S}$ , and let  $(C'+\mathbf{q}')$  be an arbitrary marking of  $\mathcal{S}$ . Then, for every finite execution  $\tau\tau_1$  of  $\mathcal{A}$ , there is  $\tau_2$  of length at most  $K(C)$  such that the firing sequence  $(\tau\tau_1\tau_2, \bar{\tau}\bar{\tau}_1\bar{\tau}_2)$  leads to  $(C'+\mathbf{q}')$ . So the probability that an infinite execution of  $Cyl(\tau\tau_1)$  eventually reaches  $(C'+\mathbf{q}')$  is equal to 1. This concludes the proof of the claim.

We now proceed to prove the proposition. Assume that some bottom SCC  $\mathcal{S}$  of the proper reachability graph of  $N(\mathcal{A}, \mathcal{R}_\varphi)$  is accepting for some Rabin pair  $\langle F, G \rangle$ , and is also reachable

from  $(C + \mathbf{q}'_0)$ . Let  $(\sigma, \bar{\sigma})$  be a firing sequence leading to some marking of  $\mathcal{S}$ , and let  $(C_G + \mathbf{q}_G)$  be a marking of  $\mathcal{S}$ . Because  $\mathcal{S}$  is accepting we have  $q_G \in G$ . For the same reason, no marking of  $\mathcal{S}$  is of the form  $(C_F + \mathbf{q}_F)$  with  $q_F \in F$ . By the claim, an infinite execution  $\tau \in \text{Cyl}(\sigma)$  satisfies with probability 1 that the infinite firing sequence  $(\tau, \bar{\tau})$  visits  $(C_G + \mathbf{q}_G)$  infinitely often and visits no marking  $(C_F + \mathbf{q}_F)$  with  $q_F \in F$ . So  $\Pr[\mathcal{A}, C \models \varphi] \geq \Pr(\text{Cyl}(\sigma)) > 0$ .

Assume now that no bottom SCC of the proper reachability graph of  $N(\mathcal{A}, \mathcal{R}_\varphi)$  is accepting. Then, for every Rabin pair  $\langle F, G \rangle$  and every bottom SCC  $\mathcal{S}$ , either  $\mathcal{S}$  contains a marking  $(C' + \mathbf{q}')$  such that  $q' \in F$ , or it contains no marking  $(C' + \mathbf{q}')$  such that  $q' \in G$ . By the claim, for every infinite execution  $\tau$  of  $\mathcal{A}$  starting at  $C$ , the infinite firing sequence  $(\tau, \bar{\tau})$  gets eventually trapped in a bottom SCC, say  $\mathcal{S}$ , with probability 1. If  $\mathcal{S}$  contains a marking  $(C' + \mathbf{q}')$  such that  $q' \in F$ , then, again by the claim,  $(\tau, \bar{\tau})$  visits  $(C' + \mathbf{q}')$  infinitely often with probability 1, and so it is non-accepting with probability 1. If  $\mathcal{S}$  contains no marking  $(C' + \mathbf{q}')$  such that  $q' \in G$ , then with probability 1 it visits configurations  $(C' + \mathbf{q}')$  such that  $q' \in G$  only finitely often, and so it is not accepting with probability 1. So  $C$  satisfies  $\varphi$  with probability 0.  $\blacktriangleleft$

Using this result we now proceed to prove the indistinguishability of the fair and the probability semantics we announced in Section 2.1:

► **Proposition 7.** *Let  $(\mathcal{A}, \mathcal{I})$  be a population protocol,  $C \in \mathcal{I}$ , and let  $\varphi$  be an LTL formula. We have:  $\Pr[\mathcal{A}, C \models \varphi] > 0$  iff some fair execution of  $\mathcal{A}$  starting at  $C$  satisfies  $\varphi$ .*

**Proof.** It is easy to show (see Esparza et al. [18, 19]) that every fair execution of  $\mathcal{A}$  starting at a configuration  $C$  gets eventually trapped in a bottom SCC of the configuration graph of  $\mathcal{A}$ , and crosses all its edges infinitely often. Using the same arguments as in Proposition 6, we show that a fair execution starting at  $C$  satisfies  $\varphi$  iff its unique counterpart firing sequence reaches an accepting bottom SCC of the proper reachability graph of  $N(\mathcal{A}, \mathcal{R}_\varphi)$ . So, by Proposition 6, some fair executions starting at  $C$  satisfy  $\varphi$  iff  $\Pr[\mathcal{A}, C \models \varphi] > 0$ .  $\blacktriangleleft$

We need the following fact about Petri nets from [22] to prove decidability.

► **Lemma 8** ([22]). *Let  $N$  be a Petri net. The set of pairs of markings  $(M, M')$  such that  $M$  and  $M'$  are mutually reachable (i.e.,  $M'$  is reachable from  $M$  and  $M$  is reachable from  $M'$ ) is Presburger, and a Presburger formula  $\text{MR}(M, M')$  denoting it can be effectively constructed.*

► **Theorem 9.** *Let  $(\mathcal{A}, \mathcal{I})$  be a population protocol, and let  $\varphi$  be an LTL formula. The problems whether there exists a configuration  $C \in \mathcal{I}$  satisfying  $\Pr[\mathcal{A}, C \models \varphi] > 0$  and  $\Pr[\mathcal{A}, C \models \varphi] < 1$  can be reduced to the reachability problem for Petri nets.*

**Proof.** Let  $\mathcal{R}_\varphi$  be a DRA for  $\varphi$ . Assume for simplicity that  $\mathcal{R}_\varphi$  has only one Rabin pair  $\langle F, G \rangle$  (the generalization to multiple pairs is straightforward). We first show that the set of markings of  $N(\mathcal{A}, \mathcal{R}_\varphi)$  that belong to accepting bottom SCCs of the proper reachability graph is Presburger, and that a formula  $\text{BA}(M)$  denoting it can be constructed.

Let  $\text{P}(M)$  be the Presburger formula characterizing the proper markings of  $N(\mathcal{A}, \mathcal{R}_\varphi)$ . Further, let  $\text{FO}(M, M')$  be a Presburger formula that holds if  $M'$  can be reached from  $M$  by firing one transition. Then we can take for  $\text{BA}(M)$  the formula

$$\begin{aligned} \text{P}(M) \quad &\wedge \quad \forall M', M'' : (\text{MR}(M, M') \wedge \text{FO}(M', M'')) \Rightarrow \text{MR}(M, M'') \\ &\wedge \quad \forall M' : \text{MR}(M, M') \Rightarrow \left( \bigwedge_{q \in F} M'(q) = 0 \right) \\ &\wedge \quad \exists M' : \text{MR}(M, M') \wedge \bigvee_{q \in G} M' \geq \mathbf{q} \end{aligned}$$

where  $\text{MR}(M, M')$  is obtained following Lemma 8. Similarly, we obtain a formula  $\text{BR}(M)$  for the markings that belong to a non-accepting bottom SCCs of  $N(\mathcal{A}, \mathcal{R}_\varphi)$ . By Proposition 6, the

problem whether some configuration  $C \in \mathcal{I}$  satisfies  $\Pr[\mathcal{A}, C \models \varphi] > 0$  reduces to deciding whether some marking  $M'$  satisfying  $\text{BA}(M')$  is reachable from some proper marking  $M$  in the Presburger set given by  $\{(C + \mathbf{q}'_0) \mid C \in \mathcal{I}\}$ . Similarly,  $\Pr[\mathcal{A}, C \models \varphi] < 1$  reduces to reachability of  $\text{BR}(M')$ . ◀

We show that the problems of Theorem 9 are *reachability hard*, i.e., at least as hard as the reachability problem of Petri nets. In previous works [18, 19], we proved reachability hardness for the well-specification problem (whether a given protocol computes a predicate). However, the protocols given by the reduction from the reachability problem always had a leader (formally, they always had a state such that at all initial configurations that state was inhabited by *exactly* one agent, and this was crucial for the proof). When population protocols are used to compute by consensus (see Example 2), leaderless protocols turn out to have the same computational power as protocols with leader; the only difference is that the latter can be faster [3, 5]. Therefore, the question arises whether verification problems have lower complexity for the special case of leaderless protocols. A positive answer would mean that one can trade-off efficiency for ease of verification, without losing computational power. We now show that, unfortunately, this is not the case: the qualitative model-checking problem for the basic liveness property **FGa** is reachability hard for leaderless protocols. The same technique also proves hardness of the well-specification problem.

► **Theorem 10.** *The reachability problem for Petri nets can be reduced in linear time to the following problem: given a population protocol  $(\mathcal{A}, \mathcal{I})$  where  $\mathcal{I}$  is simple, decide if some configuration  $C \in \mathcal{I}$  satisfies  $\Pr[\mathcal{A}, C \models \text{FGa}] > 0$ .*

**Sketch of Proof.** The proof constructs a sequence of reductions from the Petri net reachability problem. Each step in the sequence transform a problem on Petri net into an equivalent problem closer to the model of population protocols. We first use a result of Hack [21] that reduces the reachability problem to the problem of deciding for a given Petri net  $N$  and a marking  $M_0$  with a distinguished place  $\hat{p}$  if some marking  $M \in \text{Reach}(N, M_0)$  satisfies  $M(\hat{p}) = 0$ . Then we introduce a “normal form” for nets: a net  $N = (P, T, F)$  is in normal form if  $F(x, y) \in \{0, 1\}$  for every  $x, y \in (P \times T) \cup (T \times P)$ , and every transition  $t$  satisfies  $1 \leq |\bullet t| \leq 2$  and  $1 \leq |t \bullet| \leq 2$ . Transitions of Petri nets in normal form can be simulated by transitions of population protocols, which only involve two agents. We prove that the reachability problem reduces to: given a net  $N$  in normal form, a place  $p_0$  and a set of places  $\hat{P}$ , decide if some marking  $M \in \text{Reach}(N, \mathbf{p}_0)$  satisfies  $M(\hat{P}) = 0$ . The next step applies a simple but key observation: for any two markings  $M, M'$  of a net,  $M'$  is reachable from  $M$  iff  $M$  is reachable from  $M'$  in the *reverse* net obtained by reversing the arcs. This allows to reduce the reachability problem to: given a net  $N$  in normal form, a place  $p_0$  and a set of places  $\hat{P}$ , decide if  $\mathbf{p}_0 \in \text{Reach}(N, M)$  for some marking  $M$  satisfying  $M(\hat{P}) = 0$ . The crucial point is that this set of markings corresponds to a *simple* set of initial configurations of the protocol simulating the net. So, starting from this simple set, the protocol can reach a certain configuration iff  $\mathbf{p}_0$  is reachable. It still remains to ensure that the protocol satisfies  $\Pr[\mathcal{A}, C \models \text{FGa}] > 0$  for some initial configuration iff the marking  $\mathbf{p}_0$  is reachable. This is achieved by adding “probing” transitions to the protocol, labeled by an action different from  $b$ , ensuring that the protocol can always do a  $b$  as long as it has not reached  $\mathbf{p}_0$ . ◀

### 3.2 Quantitative Model Checking Is Undecidable

We prove that, contrary to the qualitative case in the previous section, the quantitative model checking problem is undecidable. Our proof uses the simulation of deterministic two counter machines with arbitrarily small error.



Recall that a counter machine is a triple  $M = (L, Co, In)$  where  $L$  is a finite set of program labels,  $Co$  is a finite set of counters, and  $In$  is a set of program instructions, one for each label. The program instruction for label  $\ell$  is of one of the following types:  $\ell: c := c + 1; \text{ goto } \ell'$  (increment),  $\ell: c := c - 1; \text{ goto } \ell'$  (decrement),  $\ell: \text{ if } c = 0 \text{ then goto } \ell' \text{ else goto } \ell''$  (zero-test), or  $\ell: \text{ halt}$  (termination). Only one label  $\ell_h$  has an instruction of the last type, and there is also a distinguished initial label  $\ell_0$ . The *termination problem for counter machines* consists of deciding if a given machine, starting at  $\ell_0$  with all counters initially set to 0, eventually halts, i.e., reaches  $\ell_h$ .

► **Lemma 11.** *Given a counter machine  $M$ , we can construct in polynomial time a protocol scheme  $\mathcal{A}$  with a distinguished state  $q_h$ , and a set  $\mathcal{I}$  of initial configurations such that  $M$  halts iff some configuration  $C \in \mathcal{I}$  satisfies the following property: starting at  $C$ , the protocol eventually reaches a configuration with one agent in  $q_h$  with probability at least  $1/2$ .*

**Proof.** It is convenient to define first the set of states of  $\mathcal{A}$ , then the set  $\mathcal{I}$  of initial configurations, and then the transitions of  $\mathcal{A}$ .

- $\mathcal{A}$  has a state for each label and for each counter of  $M$ , three distinguished states *Store*,  $D$  (for *Dummy*), and *Stop*, and two auxiliary states  $\ell_1, \ell_2$  for each zero-test label  $\ell$ . We set  $q_h := \ell_h$ , i.e., choose the state  $q_h$  as the one corresponding to the halting label.
- $\mathcal{I}$  contains the configurations that put one agent in the initial label  $\ell_0$ , one agent in  $D$ , arbitrarily many agents in *Store*, and no agent elsewhere.

Before defining the transitions of  $\mathcal{A}$  we give some intuition. First, the transitions guarantee that every reachable configuration puts one agent in exactly one of the states corresponding to the programs labels  $L$ . This models that the next instruction executed by the machine is the one with label  $\ell$ . We call this agent the *control agent*. The number of agents at a state  $c$  models the current value of the counter. The transitions also guarantee that the one agent at  $D$  never moves elsewhere (its role is only to enable some transitions).

Increasing and decreasing a counter  $c$  is modeled by transitions that transfer an agent from *Store* to  $c$ , and from  $c$  to *Store*, respectively. Therefore, if an initial configuration puts, say,  $K$  agents in *Store*, then from that configuration the protocol cannot always simulate the complete execution of the machine, only the prefix during which the sum of the values of all counters does not exceed  $K$ .  $\mathcal{A}$  has the following transitions:

- For each instruction  $\ell: c := c + 1; \text{ goto } \ell'$ , a transition  $(\ell, \text{Store}) \xrightarrow{\text{inc}} (\ell', c)$ .
- For each instruction  $\ell: c := c - 1; \text{ goto } \ell'$ , a transition  $(\ell, c) \xrightarrow{\text{dec}} (\ell', \text{Store})$ .
- For each instruction  $\ell: \text{ if } c = 0 \text{ then goto } \ell' \text{ else goto } \ell''$ , the following transitions:
  - $(\ell, D) \xrightarrow{\text{go}} (\ell_1, D)$ ,  $(\ell, c) \xrightarrow{\text{nonzero}} (\ell'', c)$
  - $(\ell_1, \text{Store}) \xrightarrow{\text{back}} (\ell, \text{Store})$ ,  $(\ell_1, D) \xrightarrow{\text{zero}} (\ell_2, D)$
  - $(\ell_2, D) \xrightarrow{\text{zero}' } (\ell', D)$  and  $(\ell_2, c) \xrightarrow{\text{game over}} (\text{Stop}, c)$ .

It is convenient to think of the *go* and *back* transitions as a loop moving an agent from  $\ell$  to  $\ell_1$  and back, and of the transitions *zero* and *nonzero* as the two possible exits of this loop.

Before proving the lemma, we make an observation. Assume that the control agent of a configuration  $C$  is at the state of a zero-test label  $\ell: \text{ if } c = 0 \text{ then goto } \ell' \text{ else goto } \ell''$ , and assume further  $C(\text{Store}) \geq 1$ . We consider two cases. In the first case,  $C$  puts no agents in counter  $c$ . Then the *nonzero* exit of the loop is not enabled at  $C$ . Therefore, the scheduler will eventually move the control agent to  $\ell_2$  with probability 1 (after executing the “loop” *go back* a number of times). Further, since the *game over* action is also not enabled, the scheduler will eventually move the control agent to  $\ell'$  with probability 1.



In the second case,  $C$  puts at least one agent in  $c$ . Then both exits are enabled at  $C$ , and the scheduler eventually chooses one of them with probabilities  $p_z$  and  $p_{nz}$ , respectively. The key point is that these probabilities depend on the number of agents in  $Store$  and in  $c$ . Indeed, for every value of  $c$ , increasing the number  $N$  of agents in  $Store$  also increases  $p_{nz}$ , since it makes it more likely that the scheduler picks agents at  $Store$ . In particular, we have  $p_{nz} \rightarrow 1$  when  $N \rightarrow \infty$ . So the scheduler eventually moves the control agent to  $\ell''$  with probability that tends to 1 when  $N$  tends to infinity.

We prove the left-to-right direction of the lemma. Assume that  $M$  halts. We show that there is a configuration  $C \in \mathcal{I}$  from which the protocol eventually reaches a configuration with the control agent in  $q_h$  with probability at least  $1/2$ .

Let  $k$  be the length of the halting computation of  $M$ . Clearly, during the computation no counter ever has a value larger than  $k$ . So the probability of the protocol taking the wrong exit when the control agent visits a zero-test label is always bounded by a value  $p(N)$  that tends to 0 as  $N$  tends to infinity. Since the computation of  $M$  visits zero-test labels at most  $k$  times, the probability that at all these visits the protocol chooses the right exit is at least  $(1 - p(N))^k$ , which tends to 1 as  $N$  tends to infinity. So, by making  $N$  sufficiently large, we obtain an initial configuration for which the protocol faithfully simulates the computation of  $M$  with probability at least  $1/2$ . Since  $M$  halts, that computation visits  $q_h$ , and we are done.

We now prove the converse direction, for which we need the *game over* actions, which have played no role so far. Assume that  $M$  does not halt. We prove that for every configuration  $C \in \mathcal{I}$  the probability that, starting at  $C$ , the protocol eventually reaches a configuration with the control agent in  $q_h$  is at most  $1/2$ .

We say that the protocol “cheats” during the simulation of  $M$  if, after reaching a configuration with the control agent at a zero-test label  $\ell$  and a strictly positive number of agents at  $c$ , the protocol moves the control agent to  $\ell'$ , and not to  $\ell''$ , as indicated by the instruction.

Let  $C$  be an arbitrary initial configuration and let the protocol produce an execution. Since  $M$  does not halt, in order to move the control agent to  $q_h$  the protocol must cheat at least once. For this, the scheduler must move the control agent to  $\ell'$  at a moment at which there is at least one agent in counter  $c$ . But then exactly one pair of agents can move the control agent to  $\ell'$  – namely the agents at  $\ell_2$  and  $D$  – and at least one pair of agents can move it to  $Stop$  (the agent at  $\ell_2$  and one of the agents at  $c$ ). Since after reaching  $Stop$  the protocol cannot reach  $q_h$  anymore, the probability that after moving to  $\ell_2$  the control agent eventually reaches  $q_h$  is at most  $1/2$ . So the probability of reaching  $q_h$  is bounded from above by  $1/2$  times the probability that an execution cheats at least once. Hence, in particular, the probability is at most  $1/2$ , and we are done. ◀

► **Proposition 12.** *The quantitative model checking problem for population protocols is already undecidable for specifications of the form  $\mathbf{G}(\bigvee_{a \in A} a)$  for some set  $A$  of action labels.*

**Proof.** We show that the problem is already undecidable for a formula of the form  $\mathbf{G}(\bigvee_{a \in A} a)$  for some set  $A$  of action labels, and the probability bound  $1/2$ . We proceed by reduction from the non-termination problem for counter machines. Given a machine  $M$ , we construct a protocol  $\mathcal{A}$  and a set of initial configurations  $\mathcal{I}$  such that  $M$  halts iff  $\Pr[\mathcal{A}, C \models \mathbf{G}(\bigvee_{a \in A} a)] \geq 1/2$  for every  $C \in \mathcal{I}$ . Almost all the work has been done in Lemma 11. Consider the protocol  $\mathcal{A}$  and the set  $\mathcal{I}$  defined there, and add a transition  $(q_h, D) \xrightarrow{halt} (q_h, D)$ . Then, an execution of  $\mathcal{A}$  satisfies  $\mathbf{F} halt$  iff it eventually moves the control agent to state  $q_h$ . Applying the lemma, we obtain that  $M$  does not halt iff  $\Pr[\mathcal{A}, C \models \mathbf{F} halt] < 1/2$  for every  $C \in \mathcal{I}$ . Taking  $A$  as the set of all actions of  $\mathcal{A}$  but  $halt$ , we get:  $M$  does not halt iff  $\Pr[\mathcal{A}, C \models \mathbf{G}(\bigvee_{a \in A} a)] \geq 1/2$  for every  $C \in \mathcal{I}$ . ◀

In fact, Lemma 11 also implies undecidability of the problem whether the probability of a property can be made arbitrarily close to 1 by increasing the number of agents. Indeed, a look at the proof of the lemma shows that the reduction produces a protocol satisfying the following property: the counter machine halts iff there a bound  $\rho < 1$  (in the lemma,  $\rho = 1/2 + \varepsilon$ ) such that  $\Pr[\mathcal{A}, C \models \varphi] \leq \rho$  for every  $C \in \mathcal{I}$ .

#### 4 Discussion and Further Undecidability Results

We have characterized the decidability frontier for LTL model checking of population protocols: qualitative model checking is decidable, and quantitative is undecidable. We have also shown that, though decidable, qualitative model checking is as hard as the reachability problem for Petri nets (for which no primitive-recursive algorithm is known) even for leaderless protocols and for the simple formula **FGa**. Essentially the same proof shows that the well-specification problem is also as hard as the Petri net reachability problem even in the leaderless case, removing the assumption of a leader from our previous hardness proof [19].

In the rest of the section we briefly discuss other undecidability results showing that Theorem 9 is rather close to the “decidability border.”

**LTL on Configurations.** Note that we have defined LTL on actions. An alternate definition could take the set of configurations as atomic propositions. Configuration-based LTL model checking is known to be decidable for Well-Structured Transition Systems (WSTS) – a general class which includes population protocols and much more – provided the reasoning can be restricted to upward-closed sets. This is the key idea used by Baier et al. [8] who prove that a state-based fragment of  $\mu$ -calculus is decidable for all WSTS. It is not known whether this result can be made more general when focusing on population protocols instead of the whole class of WSTS. Unfortunately, the model checking problem for population protocols is undecidable even for very simple classes of atomic propositions.

Given a state  $q$  of a protocol, let  $q_{\geq 1}$  denote the atomic proposition that holds for a configuration  $C$  if  $C(q) \geq 1$ . We call  $q_{\geq 1}$  a *flag*, since it flags that state  $q$  is inhabited.

► **Proposition 13.** *The qualitative model checking problem for population protocols and LTL specifications over flags is undecidable.*

**Proof.** Instead of using the construction of Lemma 11 for zero-tests, we enable the zero-test transition always, and use an LTL formula over configurations to catch “cheating”, i.e., the population protocol taking a zero-test when it should not. Indeed, every zero-test instruction  $\ell$ : `if  $c = 0$  then goto  $\ell'$  else goto  $\ell''$`  yields the formula:

$$\mathbf{G}(q_{\geq 1} \wedge c_{\geq 1} \Rightarrow q_{\geq 1} \mathbf{U} q''_{\geq 1}) \quad (1)$$

where  $q, q''$  are the states modelling the locations  $\ell, \ell'$  and  $c$  the state modelling the counter. The final formula is the conjunction of the formulas given at (1) (one conjunct for each zero-test) together with  $\mathbf{F}(q_h)_{\geq 1}$ . If the counter machine halts, then this formula holds with nonzero probability from an initial configuration with exactly one agent in the control location and sufficiently many agents in the *Store* location. If the counter machine rejects, then the probability of the formula is zero for every initial configuration. ◀

**Broadcast Protocols.** Adding broadcasts makes the qualitative model checking undecidable as well. Consider an extension of population protocols with *broadcast actions* [16] where, in addition to a set  $\Delta$  of transitions involving two agents, also a set  $\Delta^* \subseteq Q \times \Sigma \times Q \times 2^{Q \times Q}$  of

“broadcast” transitions is allowed. Given a broadcast transition  $(q, a, q', \delta)$  and a configuration  $C$  satisfying  $C(q) > 0$ , if the scheduler picks an agent in state  $q$ , then the agent changes its state to  $q'$  and simultaneously, *all* other agents update their states according to the function  $\delta$ . The qualitative model checking problem for this model is undecidable, because the model can weakly simulate counter machines by slightly modifying the proof of [16, Theorem 5.1].

---

## References

- 1 Parosh Aziz Abdulla, Radu Ciobanu, Richard Mayr, Arnaud Sangnier, and Jeremy Sproston. Qualitative analysis of VASS-induced MDPs. In *FoSSaCS'16*, LNCS, pages 319–334. Springer, 2016. doi:10.1007/978-3-662-49630-5\_19.
- 2 Parosh Aziz Abdulla, Noomene Ben Henda, and Richard Mayr. Decisive Markov chains. *Logical Methods in Computer Science*, 3(4), 2007. doi:10.2168/LMCS-3(4:7)2007.
- 3 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC'04*, pages 290–299. ACM, 2004. doi:10.1145/1011767.1011810.
- 4 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *PODC'06*, pages 292–299. ACM, 2006. doi:10.1145/1146381.1146425.
- 5 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, 2008. doi:10.1007/s00446-008-0067-z.
- 6 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007. doi:10.1007/s00446-007-0040-2.
- 7 Tamarah Arons, Amir Pnueli, and Lenore D. Zuck. Parameterized verification by probabilistic abstraction. In *FOSSACS'03*, volume 2620 of LNCS, pages 87–102. Springer, 2003. doi:10.1007/3-540-36576-1\_6.
- 8 Christel Baier, Nathalie Bertrand, and Philippe Schnoebelen. On computing fixpoints in well-structured regular model checking, with applications to lossy channel systems. In *LPAR'06*, volume 4246 of LNCS, pages 347–361. Springer, 2006. doi:10.1007/11916277\_24.
- 9 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- 10 Nathalie Bertrand and Paulin Fournier. Parameterized verification of many identical probabilistic timed processes. In *FSTTCS'13*, volume 24 of *LIPICs*, pages 501–513. Schloss Dagstuhl, 2013. doi:10.4230/LIPICs.FSTTCS.2013.501.
- 11 Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Playing with probabilities in reconfigurable broadcast networks. In *FoSSaCS'14*, volume 8412 of LNCS, pages 134–148. Springer, 2014. doi:10.1007/978-3-642-54830-7\_9.
- 12 Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. In *ICALP'16*, volume 55 of *LIPICs*, pages 106:1–106:14. Schloss Dagstuhl, 2016. doi:10.4230/LIPICs.ICALP.2016.106.
- 13 J. Clement, C. Delporte-Gallet, H. Fauconnier, and M. Sighireanu. Guidelines for the verification of population protocols. In *ICDCS'11*, pages 215–224, 2011. doi:10.1109/ICDCS.2011.36.
- 14 Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. Secretive birds: Privacy in population protocols. In *OPODIS'07*, volume 4878 of LNCS, pages 329–342. Springer, 2007. doi:10.1007/978-3-540-77096-1\_24.

- 15 Zoë Diamadi and Michael J. Fischer. A simple game for the study of trust in distributed systems. *Wuhan University Journal of Natural Sciences*, 6(1):72–82, 2001. doi:10.1007/BF03160228.
- 16 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *LICS'99*, pages 352–359. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782630.
- 17 Javier Esparza, Andreas Gaiser, and Stefan Kiefer. Proving termination of probabilistic programs using patterns. In *CAV'12*, volume 7358 of *LNCS*, pages 123–138. Springer, 2012. doi:10.1007/978-3-642-31424-7\_14.
- 18 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. In *CONCUR'15*, volume 42 of *LIPICs*, pages 470–482. Schloss Dagstuhl, 2015. doi:10.4230/LIPICs.CONCUR.2015.470.
- 19 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, pages 1–25, 2016. doi:10.1007/s00236-016-0272-3.
- 20 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992. doi:10.1145/146637.146681.
- 21 Michel Henri Théodore Hack. Decidability questions for Petri nets. Technical Report 161, MIT, 1976.
- 22 Jérôme Leroux. Vector addition system reversible reachability problem. In *CONCUR'11*, volume 6901 of *LNCS*, pages 327–341. Springer, 2011. doi:10.1007/978-3-642-23217-6\_22.
- 23 P. A. P. Moran. Random processes in genetics. *Mathematical Proceedings of the Cambridge Philosophical Society*, 54:60–71, 1 1958. doi:10.1017/S0305004100033193.
- 24 Saket Navlakha and Ziv Bar-Joseph. Distributed information processing in biological and computational systems. *Commun. ACM*, 58(1):94–102, December 2014. doi:10.1145/2678280.
- 25 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008. doi:10.1007/s11047-008-9067-y.
- 26 Lenore D. Zuck, Amir Pnueli, and Yonit Kesten. Automatic verification of probabilistic free choice. In *VMCAI'02*, volume 2294 of *LNCS*, pages 208–224. Springer, 2002. doi:10.1007/3-540-47813-2\_15.

# Visibly Linear Dynamic Logic\*

Alexander Weinert<sup>1</sup> and Martin Zimmermann<sup>2</sup>

1 Reactive Systems Group, Saarland University, Saarbrücken, Germany  
weinert@react.uni-saarland.de

2 Reactive Systems Group, Saarland University, Saarbrücken, Germany  
zimmermann@react.uni-saarland.de

---

## Abstract

We introduce Visibly Linear Dynamic Logic (VLDL), which extends Linear Temporal Logic (LTL) by temporal operators that are guarded by visibly pushdown languages over finite words. In VLDL one can, e.g., express that a function resets a variable to its original value after its execution, even in the presence of an unbounded number of intermediate recursive calls. We prove that VLDL describes exactly the  $\omega$ -visibly pushdown languages. Thus it is strictly more expressive than LTL and able to express recursive properties of programs with unbounded call stacks.

The main technical contribution of this work is a translation of VLDL into  $\omega$ -visibly pushdown automata of exponential size via one-way alternating jumping automata. This translation yields exponential-time algorithms for satisfiability, validity, and model checking. We also show that visibly pushdown games with VLDL winning conditions are solvable in triply-exponential time. We prove all these problems to be complete for their respective complexity classes.

**1998 ACM Subject Classification** F.4.1 Mathematical Logic

**Keywords and phrases** Temporal Logic, Visibly Pushdown Languages, Satisfiability, Model Checking, Infinite Games

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.28

## 1 Introduction

Linear Temporal Logic (LTL) [9] is widely used for the specification of non-terminating systems. Its popularity is owed to its simple syntax and intuitive semantics, as well as to the exponential compilation property, i.e., for each LTL formula there exists an equivalent Büchi automaton of exponential size. Due to the latter property, there exist algorithms for model checking in polynomial space and for solving infinite games in doubly-exponential time.

While LTL suffices to express properties of circuits and non-recursive programs with bounded memory, its application to real-life programs is hindered by its inability to express recursive properties. In fact, LTL is too weak to even express all  $\omega$ -regular properties. There are several approaches to address the latter shortcoming, by augmenting LTL, for example, with regular expressions [7, 10], finite automata on infinite words [11], and right-linear grammars [13]. We concentrate on the approach of Linear Dynamic Logic (LDL) [10], which guards the globally- and eventually-operators of LTL with regular expressions. While the LTL-formula  $\mathbf{F}\psi$  simply means “either now, or at some point in the future,  $\psi$  holds”, the corresponding LDL operator  $\langle r \rangle \psi$  means “There exists an infix matching the regular expression  $r$  starting at the current position, and  $\psi$  holds true after that infix”.

---

\* Supported by the projects “TriCS” (ZI 1516/1-1) and “AVACS” (SFB/TR 14) of the German Research Foundation (DFG).



The logic LDL captures the  $\omega$ -regular languages. In spite of its greater expressive power, LDL still enjoys the exponential compilation property, hence there exist algorithms for model checking and solving infinite games in polynomial space and doubly-exponential time, respectively.

While the expressive power of LDL is sufficient for many specifications, it is still not able to reason about recursive properties of systems. In order to address this shortcoming, we replace the regular languages guarding the temporal operators with visibly pushdown languages (VPLs) [2]. We consider VPLs specified by visibly pushdown automata (VPAs) [2] in this work.

A VPA is a pushdown automaton that operates over a fixed partition of the input alphabet into calls, returns and local actions. In contrast to traditional pushdown automata, VPAs may only push symbols onto the stack when reading calls and may only pop symbols off the stack when reading returns. Moreover, they may not even inspect the topmost symbol of the stack when not reading returns. Thus, the height of the stack after reading a word is known in advance for all VPAs using the same partition of the input alphabet. Due to this, VPAs are closed under union and intersection, as well as complementation. The class of languages accepted by VPAs is known as visibly pushdown languages.

The class of such languages over infinite words, i.e.,  $\omega$ -visibly pushdown languages, are known to allow for the specification of many important properties in program verification such as “there are infinitely many positions at which at most two functions are active”, which expresses repeated returns to a main-loop, or “every time the program enters a module  $m$  while  $p$  holds true,  $p$  holds true upon exiting  $m$ ” [2]. The extension of VPAs to their variant operating on infinite words is, however, not well-suited to the specification of such properties in practice, as Boolean operations on such automata do not preserve the logical structure of the original automata. By guarding its temporal operators with VPAs, VLDDL allows for modular specification of recursive properties while capturing  $\omega$ -VPAs.

## 1.1 Our contributions

We introduce VLDDL and study its expressiveness and algorithmic properties.

Firstly, we provide translations from VLDDL to VPAs over infinite words, so-called  $\omega$ -VPAs, and vice versa. For the direction from logic to automata we translate VLDDL formulas into one-way alternating jumping automata (1-AJA), which are known to be translatable into  $\omega$ -VPAs of exponential size due to Bozzelli [4]. For the direction from automata to logic we use a translation of  $\omega$ -VPAs into deterministic parity stair automata (PSA) by Löding et al. [8], which we then translate into VLDDL formulas.

Secondly, we prove the satisfiability problem and the validity problem for VLDDL to be EXPTIME-complete. Membership in EXPTIME follows from the previously mentioned constructions, while we show EXPTIME-hardness of the problems by a reduction from the word problem for polynomially space-bounded alternating Turing machines adapting a similar reduction by Bouajjani et al. [3].

As a third result, we show that model checking visibly pushdown systems against VLDDL specifications is EXPTIME-complete as well. Membership in EXPTIME follows from EXPTIME-membership of the model checking problem for 1-AJAs against visibly pushdown systems. EXPTIME-hardness follows from EXPTIME-hardness of the validity problem for VLDDL.

Finally, solving visibly pushdown games with VLDDL winning conditions is proven to be 3EXPTIME-complete. Membership in 3EXPTIME follows from the exponential translation of VLDDL formulas into  $\omega$ -VPAs and the membership of solving pushdown games against  $\omega$ -VPA winning conditions in 2EXPTIME due to Löding et al. [8]. 3EXPTIME-hardness is

due to a reduction from solving pushdown games against LTL specifications, again due to Löding et al. [8].

Our results show that VLDL allows for the concise specification of important properties in a logic with intuitive semantics. In the case of satisfiability and model checking, the complexity jumps from PSPACE-completeness for LDL to EXPTIME-completeness. For solving infinite games, we gain an exponent moving from 2EXPTIME-completeness to 3EXPTIME-completeness.

We choose VPAs for the specification of guards in order to simplify arguing about the expressive power of VLDL. In order to simplify the modeling of  $\omega$ -VPLs, other formalisms that capture VPLs over finite words may be used. We discuss one such formalism in the conclusion.

All proofs omitted due to space restrictions can be found in the full version [12].

## 1.2 Related Work

The need for specification languages able to express recursive properties has been identified before and there exist other approaches to using visibly pushdown languages over infinite words for specifications, most notably CaRet [1], and, more recently, VLTL [5]. While VLTL captures the class of  $\omega$ -visibly pushdown languages, CaRet captures only a strict subset of it. For both logics there exist exponential translations into  $\omega$ -VPAs. In this work, we provide exponential translations from VLDL to  $\omega$ -VPAs and vice versa. Hence, CaRet is strictly less powerful than VLDL, but every CaRet formula can be translated into an equivalent VLDL formula, albeit with a doubly-exponential blowup. Similarly, every VLTL formula can be translated into an equivalent VLDL formula and vice versa, with doubly-exponential blowup in both directions.

In contrast to VLTL, which introduces substitution operators to regular expressions (replacing occurrences of local actions by well-matched words), VLDL instead extends the concepts introduced for LTL and LDL with visibly pushdown automata. Hence, specifications written in VLDL are modular and have an intuitive semantics, in particular for practitioners already used to LTL.

Other logical characterizations of visibly pushdown languages include characterizations by a fixed-point logic [4] and by monadic second order logic augmented with a binary matching predicate (MSO $_{\mu}$ ) [2]. Even though these logics also capture the class of visibly pushdown languages, they feature neither an intuitive syntax nor intuitive semantics and thus are less applicable than VLDL in a practical setting.

## 2 Preliminaries

In this section we introduce the basic notions used in the remainder of this work. A pushdown alphabet  $\tilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$  is a finite set  $\Sigma$  that is partitioned into calls  $\Sigma_c$ , returns  $\Sigma_r$  and local actions  $\Sigma_l$ . We write  $w = w_0 \cdots w_n$  and  $\alpha = \alpha_0 \alpha_1 \alpha_2 \cdots$  for finite and infinite words, respectively. The stack height  $sh(w)$  reached after reading  $w$  is defined inductively as  $sh(\varepsilon) = 0$ ,  $sh(wc) = sh(w) + 1$  for  $c \in \Sigma_c$ ,  $sh(wr) = \max\{0, sh(w) - 1\}$  for  $r \in \Sigma_r$ , and  $sh(wl) = sh(w)$  for  $l \in \Sigma_l$ . A call  $c \in \Sigma_c$  at some position  $k$  of a word  $w$  is matched if there exists a  $k' > k$  with  $w_{k'} \in \Sigma_r$  and  $sh(w_0 \cdots w_k) - 1 = sh(w_0 \cdots w_{k'})$ . The return at the smallest such position  $k'$  is the matching return of  $c$ . We define  $steps(\alpha) := \{k \in \mathbb{N} \mid \forall k' \geq k. sh(\alpha_0 \cdots \alpha_{k'}) \geq sh(\alpha_0 \cdots \alpha_k)\}$  as the positions reaching a lower bound on the stack height. Note that  $0 \in steps(\alpha)$  and that  $steps(\alpha)$  is infinite for infinite words  $\alpha$ .



**Visibly Pushdown Systems.** A visibly pushdown system (VPS)  $\mathcal{S} = (Q, \tilde{\Sigma}, \Gamma, \Delta)$  consists of a finite set  $Q$  of states, a pushdown alphabet  $\tilde{\Sigma}$ , a stack alphabet  $\Gamma$ , which contains a stack-bottom marker  $\perp$ , and a transition relation

$$\Delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\perp\})) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_l \times Q).$$

A configuration  $(q, \gamma)$  of  $\mathcal{S}$  is a pair of a state  $q \in Q$  and a stack content  $\gamma \in \Gamma_c = (\Gamma \setminus \{\perp\})^* \cdot \perp$ . The VPS  $\mathcal{S}$  induces the configuration graph  $G_{\mathcal{S}} = (Q \times \Gamma_c, E)$  with  $E \subseteq ((Q \times \Gamma_c) \times \Sigma \times (Q \times \Gamma_c))$  and  $((q, \gamma), a, (q', \gamma')) \in E$  if, and only if, either (i)  $a \in \Sigma_c$ ,  $(q, a, q', A) \in \Delta$ , and  $A\gamma = \gamma'$ , (ii)  $a \in \Sigma_r$ ,  $(q, a, \perp, q') \in \Delta$ , and  $\gamma = \gamma' = \perp$ , (iii)  $a \in \Sigma_r$ ,  $(q, a, A, q') \in \Delta$ ,  $A \neq \perp$ , and  $\gamma = A\gamma'$ , or (iv)  $a \in \Sigma_l$ ,  $(q, a, q') \in \Delta$ , and  $\gamma = \gamma'$ . For an edge  $e = ((q, \gamma), a, (q', \gamma'))$ ,  $a$  is the label of  $e$ . A run  $\pi = (q_0, \gamma_0) \cdots (q_n, \gamma_n)$  of  $\mathcal{S}$  on  $w = w_0 \cdots w_{n-1}$  is a sequence of configurations where  $((q_i, \gamma_i), w_i, (q_{i+1}, \gamma_{i+1})) \in E$  in  $G_{\mathcal{S}}$  for all  $i \in [0; n-1]$ .

The VPS  $\mathcal{S}$  is deterministic if for each vertex  $(q, \gamma)$  in  $G_{\mathcal{S}}$  and each  $a \in \Sigma$  there exists at most one outgoing  $a$ -labeled edge from  $(q, \gamma)$ . In figures, we write  $\downarrow A$ ,  $\uparrow A$  and  $\rightarrow$  to denote pushing and popping  $A$  onto and off the stack, and local actions, respectively.

**(Büchi) Visibly Pushdown Automata.** A visibly pushdown automaton (VPA) [2] is a six-tuple  $\mathfrak{A} = (Q, \tilde{\Sigma}, \Gamma, \Delta, I, F)$ , where  $(Q, \tilde{\Sigma}, \Gamma, \Delta)$  is a VPS and  $I, F \subseteq Q$  are sets of initial and final states. A run  $(q_0, \gamma_0)(q_1, \gamma_1)(q_2, \gamma_2) \cdots$  of  $\mathfrak{A}$  is initial if  $(q_0, \gamma_0) = (q_I, \perp)$  for some  $q_I \in I$ . A finite run  $\pi = (q_0, \gamma_0) \cdots (q_n, \gamma_n)$  is accepting if  $q_n \in F$ . A Büchi VPA (BVPA) is syntactically identical to a VPA, but we only consider runs over infinite words. An infinite run is Büchi-accepting if it visits states in  $F$  infinitely often. A (B)VPA  $\mathfrak{A}$  accepts a word  $w$  (an infinite word  $\alpha$ ) if there exists an initial (Büchi-)accepting run of  $\mathfrak{A}$  on  $w$  ( $\alpha$ ). The family of languages accepted by (B)VPA is denoted by  $(\omega\text{-})\text{VPL}$ .

### 3 Visibly Linear Dynamic Logic

We fix a finite set  $P$  of atomic propositions and a partition  $\tilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$  of  $2^P$  throughout this work. The syntax of VLDL is defined by the grammar

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle \mathfrak{A} \rangle \varphi \mid [\mathfrak{A}] \varphi,$$

where  $p \in P$  and  $\mathfrak{A}$  ranges over testing visibly pushdown automata (TVPA) over  $\tilde{\Sigma}$ . A TVPA  $\mathfrak{A} = (Q, \tilde{\Sigma}, \Gamma, \Delta, I, F, t)$  consists of a VPA  $(Q, \tilde{\Sigma}, \Gamma, \Delta, I, F)$  and a partial function  $t$  mapping states to VLDL formulas over  $\tilde{\Sigma}$ .<sup>1</sup> Such an automaton accepts an infix  $\alpha_i \cdots \alpha_j$  of an infinite word  $\alpha_0 \alpha_1 \alpha_2 \cdots$  if the embedded VPA has an initial accepting run  $(q_i, \gamma_i) \cdots (q_{j+1}, \gamma_{j+1})$  on  $\alpha_i \cdots \alpha_j$  such that, if  $q_{i+k}$  is marked with  $\varphi$  by  $t$ , then  $\alpha_{i+k} \alpha_{i+k+1} \alpha_{i+k+2} \cdots$  satisfies  $\varphi$ .

We define the size of  $\varphi$  as the sum of the number of subformulas (including those contained as tests in automata and their subformulas) and of the numbers of states of the automata contained in  $\varphi$ . As shorthands, we use  $\mathbf{tt} := p \vee \neg p$  and  $\mathbf{ff} := p \wedge \neg p$  for some atomic proposition  $p$ . Even though the testing function  $t$  is defined as a partial function, we generally assume it is total by setting  $t: q \mapsto \mathbf{tt}$  if  $q \notin \text{domain}(t)$ .

Let  $\alpha = \alpha_0 \alpha_1 \alpha_2 \cdots$  be an infinite word over  $2^P$  and let  $k \in \mathbb{N}$  be a position in  $\alpha$ . We define the semantics of VLDL inductively via

<sup>1</sup> Obviously, there are some restrictions on the nesting of tests into automata. More formally, we require the subformula relation to be acyclic as usual.



- $(\alpha, k) \models p$  if, and only if,  $p \in \alpha_k$ ,
  - $(\alpha, k) \models \neg\varphi$  if, and only if,  $(\alpha, k) \not\models \varphi$ ,
  - $(\alpha, k) \models \varphi_0 \wedge \varphi_1$  if, and only if,  $(\alpha, k) \models \varphi_0$  and  $(\alpha, k) \models \varphi_1$ , and dually for  $\varphi_0 \vee \varphi_1$ ,
  - $(\alpha, k) \models \langle \mathfrak{A} \rangle \varphi$  if, and only if, there exists  $l \geq k$  s.t.  $(k, l) \in \mathcal{R}_{\mathfrak{A}}(\alpha)$  and  $(\alpha, l) \models \varphi$ ,
  - $(\alpha, k) \models [\mathfrak{A}] \varphi$  if, and only if, for all  $l \geq k$ ,  $(k, l) \in \mathcal{R}_{\mathfrak{A}}(\alpha)$  implies  $(\alpha, l) \models \varphi$ ,
- where  $\mathcal{R}_{\mathfrak{A}}(\alpha)$  contains all  $(k, l)$  such that  $\mathfrak{A}$  accepts  $\alpha_k \cdots \alpha_{l-1}$ . Formally, we define

$$\mathcal{R}_{\mathfrak{A}}(\alpha) := \{(k, l) \in \mathbb{N} \times \mathbb{N} \mid \exists \text{ init. acc. run } (q_k, \sigma_k) \cdots (q_l, \sigma_l) \text{ of } \mathfrak{A} \text{ on } \alpha_k \cdots \alpha_{l-1} \\ \text{and } \forall m \in \{k, \dots, l\}. (\alpha, m) \models t(q_m)\}.$$

We write  $\alpha \models \varphi$  as a shorthand for  $(\alpha, 0) \models \varphi$  and say that  $\alpha$  is a model of  $\varphi$  in this case. The language of  $\varphi$  is defined as  $L(\varphi) := \{\alpha \in (2^P)^\omega \mid \alpha \models \varphi\}$ . As usual, disjunction and conjunction are dual, as well as the  $\langle \mathfrak{A} \rangle$ -operator and the  $[\mathfrak{A}]$ -operator, which can be dualized using De Morgan's law and the logical identity  $[\mathfrak{A}] \varphi \equiv \neg \langle \mathfrak{A} \rangle \neg \varphi$ , respectively. Note that the latter identity only dualizes the temporal operator, but does not require complementation of the automaton guarding the operator. We additionally allow the use of derived boolean operators such as  $\rightarrow$  and  $\leftrightarrow$ , as they can easily be reduced to the basic operators  $\wedge$ ,  $\vee$  and  $\neg$ .

The logic VLDL combines the expressive power of visibly pushdown automata with the intuitive temporal operators of LDL. Thus, it allows for concise and intuitive specifications of many important properties in program verification [2]. In particular, VLDL allows for the specification of recursive properties, which makes it more expressive than both LDL [10] and LTL [9]. In fact, we can embed LDL in VLDL in linear time.

► **Lemma 1.** *For any LDL formula  $\psi$  over  $P$  we can effectively construct a VLDL formula  $\varphi$  over  $\tilde{\Sigma} := (\emptyset, \emptyset, 2^P)$  in linear time such that  $L(\psi) = L(\varphi)$ .*

**Proof.** We define  $\varphi$  by structural induction over  $\psi$ . The only interesting case is  $\psi = \langle r \rangle \psi'$ , since all other cases follow from closure properties and duality. We obtain the VLDL formula  $\varphi'$  over  $\tilde{\Sigma}$  equivalent to  $\psi'$  by induction and construct the finite automaton  $\mathfrak{A}_r$  from  $r$  using the construction from [6]. The automaton  $\mathfrak{A}_r$  contains tests, but is not equipped with a stack. Since  $\tilde{\Sigma} = (\emptyset, \emptyset, 2^P)$ , we can interpret  $\mathfrak{A}_r$  as a TVPA without changing the language it recognizes. We call the TVPA  $\mathfrak{A}'_r$  and define  $\varphi = \langle \mathfrak{A}'_r \rangle \varphi'$ . ◀

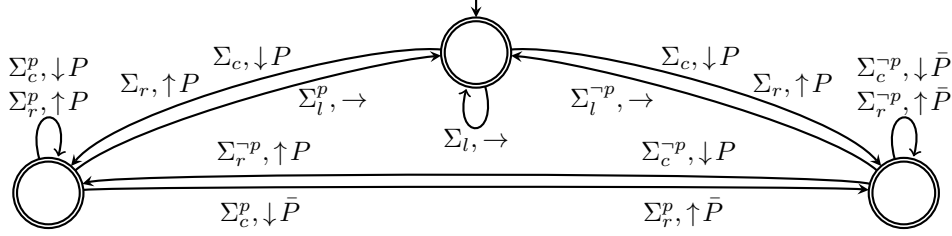
Since LTL can be in turn embedded in LDL in linear time, Lemma 1 directly implies the embeddability of LTL in VLDL in linear time. Note that this proof motivates the use of TVPAs instead of VPAs without tests as guards in order to obtain a concise formalism. We later show that removing tests from these automata does not change the expressiveness of VLDL. It is, however, open whether it is possible to translate even LTL formulas into VLDL formulas without tests in polynomial time.

► **Example 2.** Assume that we have a program that may call some module  $m$  and has the observable atomic propositions  $P := \{c, r, p, q\}$ , where  $c$  and  $r$  denote calls to and returns from  $m$ , and  $p$  and  $q$  are arbitrary propositions.

We now construct a formula that describes the condition “If  $p$  holds true immediately after entering  $m$ , it shall hold immediately after the corresponding return from  $m$  as well” [1]. For the sake of readability, we assume that the program never emits both  $c$  and  $r$  in the same step. Moreover, we assume that the program emits at least one atomic proposition in each step. Since we want to count the calls and returns occurring in the program using the stack, we pick the pushdown alphabet  $\tilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$  such that  $P' \subseteq P$  is in  $\Sigma_c$  if  $c \in P'$ ,  $P' \in \Sigma_r$  if  $r \in P'$ , but  $c \notin P'$ , and  $P' \in \Sigma_l$  otherwise.



■ **Figure 1** The automata  $\mathfrak{A}_c$  and  $\mathfrak{A}_r$  for Example 2.



■ **Figure 2** A BVPA  $\mathfrak{A}$  specifying the same language as  $\varphi$  from Example 2.

The formula  $\varphi := [\mathfrak{A}_c](p \rightarrow \langle \mathfrak{A}_r \rangle p)$  then captures the condition, with  $\mathfrak{A}_c$  and  $\mathfrak{A}_r$  as shown in Figure 1. The automaton  $\mathfrak{A}_c$  accepts all finite words ending with a call to  $m$ , whereas the automaton  $\mathfrak{A}_r$  accepts all words ending with a single unmatched return.

Figure 2 shows a BVPA  $\mathfrak{A}$  describing the same specification as  $\varphi$ . Here, we use  $\Sigma_x^p = \{P' \in \Sigma_x \mid p \in P'\}$  and  $\Sigma_x^{\bar{p}} = \{P' \in \Sigma_x \mid p \notin P'\}$  for  $x \in \{c, r, l\}$ . In contrast to  $\varphi$ , which uses only a single stack symbol, namely  $A$ , the BVPA  $\mathfrak{A}$  has to rely on the two stack symbols  $P$  and  $\bar{P}$  to track whether or not  $p$  held true after entering the module  $m$ . Moreover, there is no direct correlation between the logical structure of the specification and the structure of the BVPA, which exemplifies the difficulty of maintaining specifications given as BVPA.

In order to abstain from using automata, it would also be possible to formalize the specification using a VLTL formula [5] that describes the same language as  $\varphi$ . One such formula would be  $\psi := (\alpha; \tau\tau) \mid \alpha \rangle \text{ff}$ , where the visibly rational expression  $\alpha$  is defined as

$$\alpha := [(p \cup q)^* c [(q \square) \cup (p \square p)] r (p \cup q)^*]^{\circ \square} \curvearrowright (p \cup q)^*$$

that uses the additional local action  $\square$ . Again, the conditional nature of the specification is lost in the translation to VLTL. Moreover, the temporal nature is not well visible in the formal specification due to use of the non-standard future weak power operator  $\psi \mid \alpha \rangle \psi$ .

In contrast to these two alternative formal specifications, VLTL offers a readable and intuitive formalism that combines the well-known standard acceptors for visibly pushdown languages with guarded versions of the widely used temporal operators of LTL and the readability of classical logical operators.

#### 4 VLTL Captures $\omega$ -VPL

In this section we show that VLTL characterizes  $\omega$ -VPL. Recall that a language is in  $\omega$ -VPL if, and only if, there exists a BVPA recognizing it. We provide effective constructions for transforming BVPA into equivalent VLTL formulas and vice versa.

► **Theorem 3.** *For any language of infinite words  $L \subseteq \Sigma^\omega$  there exists a BVPA  $\mathfrak{A}$  with  $L(\mathfrak{A}) = L$  if, and only if, there exists a VLTL formula  $\varphi$  with  $L(\varphi) = L$ . There exist effective translations for both directions.*

In Section 4.1 we show the construction of VLDL formulas from BVPAs via deterministic parity stair automata. In Section 4.2 we construct one-way alternating jumping automata from VLDL formulas. These automata are known to be translatable into equivalent BVPAs. Both constructions incur an exponential blowup in size. In the construction of BVPAs from VLDL formulas, this blowup is shown to be unavoidable. It remains open whether the blowup can be avoided in the construction for the other direction.

#### 4.1 From Stair Automata to VLDL

In this section we construct a VLDL formula of exponential size that is equivalent to a given BVPA  $\mathfrak{A}$ . To this end, we first transform  $\mathfrak{A}$  into an equivalent deterministic parity stair automaton (DPSA) [8] in order to simplify the translation. A PSA  $\mathfrak{A} = (Q, \tilde{\Sigma}, \Gamma, \Delta, I, \Omega)$  consists of a VPS  $\mathcal{S} = (Q, \tilde{\Sigma}, \Gamma, \Delta)$ , a set of initial states  $I$ , and a coloring  $\Omega: Q \rightarrow \mathbb{N}$ . The automaton  $\mathfrak{A}$  is deterministic if  $\mathcal{S}$  is deterministic and  $|I| = 1$ .

A run  $\rho$  of  $\mathfrak{A}$  on a word  $\alpha$  is a run of the VPS  $\mathcal{S}$  on  $\alpha$ . Recall that a step is a position at which the stack height reaches a lower bound for the remainder of the word. A stair automaton only evaluates the parity condition at the steps of the word. Formally, a run  $\rho_\alpha = (q_0, \sigma_0)(q_1, \sigma_1)(q_2, \sigma_2) \cdots$  on the word  $\alpha$  induces a sequence of colors  $\Omega(\rho_\alpha) := \Omega(q_{k_0})\Omega(q_{k_1})\Omega(q_{k_2}) \cdots$ , where  $k_0 < k_1 < k_2 \cdots$  is the ordered enumeration of the steps of  $\alpha$ . A DPSA  $\mathfrak{A}$  accepts an infinite word  $\alpha$  if there exists an initial run  $\rho$  of  $\mathfrak{A}$  on  $\alpha$  such that the largest color appearing infinitely often in  $\Omega(\rho)$  is even. The language  $L(\mathfrak{A})$  of a parity stair automaton  $\mathfrak{A}$  is the set of all words  $\alpha$  that are accepted by  $\mathfrak{A}$ .

► **Lemma 4** ([8]). *For every BVPA  $\mathfrak{A}$  there exists an effectively constructible equivalent DPSA  $\mathfrak{A}_{st}$  with  $|\mathfrak{A}_{st}| \in \mathcal{O}(2^{|\mathfrak{A}|})$ .*

Since the stair automaton  $\mathfrak{A}_{st}$  equivalent to a BVPA  $\mathfrak{A}$  is deterministic, the acceptance condition collapses to the requirement that the unique run of  $\mathfrak{A}_{st}$  on  $\alpha$  must be accepting. Another important observation is that every time  $\mathfrak{A}_{st}$  reaches a step of  $\alpha$ , the stack may be cleared. Since the topmost element of the stack will never be popped after reaching a step, and since VPAs cannot inspect the top of the stack, neither this symbol, nor the ones below it have any influence on the remainder of the run.

Thus, the formula equivalent to  $\mathfrak{A}_{st}$  has to specify the following constraints: There must exist some state  $q$  of even color such that the stair automaton visits  $q$  at a step, afterwards the automaton may never visit a higher color again at a step, and each visit to  $q$  at a step must be followed by another visit to  $q$  at a step. All of these conditions can be specified by VLDL formulas in a straightforward way, since  $\mathfrak{A}_{st}$  is deterministic and since there is only a finite number of colors in  $\mathfrak{A}_{st}$ .

► **Lemma 5.** *For each DPSA  $\mathfrak{A}$  there exists an effectively constructible equivalent VLDL formula  $\varphi_{\mathfrak{A}}$  with  $|\varphi_{\mathfrak{A}}| \in \mathcal{O}(|\mathfrak{A}|^2)$ .*

**Proof.** We first construct a formula  $\varphi_{st}$  such that  $(\alpha, k) \models \varphi_{st}$  if, and only if,  $k \in \text{steps}(\alpha)$ : Let  $\mathfrak{A}_{st}$  be a VPA that accepts upon reading an unmatched return, constructed similarly to  $\mathfrak{A}_r$  from Example 2. Then we can define  $\varphi_{st} := [\mathfrak{A}_{st}] \mathbf{ff}$ , i.e., we demand that the stack height never drops below the current level by disallowing  $\mathfrak{A}_{st}$  to ever accept.

In the remainder of this proof, we write  ${}_{I'}\mathfrak{A}_{F'}$  to denote the TVPA that we obtain from combining the VPS of  $\mathfrak{A}$  with the sets  $I'$  and  $F'$  of initial and final states. Additionally, we require that  ${}_{I'}\mathfrak{A}_{F'}$  does not accept the empty word. This is trivially true if the intersection of  $I'$  and  $F'$  is empty, and easily achieved by adding a new initial state if it is not. Furthermore, we define  $Q_{\text{even}} := \{q \in Q \mid \Omega(q) \text{ is even}\}$  and  $Q_{>q} := \{q' \in Q \mid \Omega(q') > \Omega(q)\}$ .

Recall that  $\mathfrak{A}$  accepts a word  $\alpha$  if the largest color seen infinitely often at a step during the run of  $\mathfrak{A}$  on  $\alpha$  is even. This is equivalent to the existence of a state  $q$  as described above. These conditions are formalized as

$$\varphi_1(q) := \langle I\mathfrak{A}_{\{q\}} \rangle (\varphi_{st} \wedge [\{q\}\mathfrak{A}_{Q>q}] \neg \varphi_{st})$$

and

$$\varphi_2(q) := [I\mathfrak{A}_{\{q\}}] (\varphi_{st} \rightarrow \langle \{q\}\mathfrak{A}_{\{q\}} \rangle \varphi_{st}),$$

respectively. We obtain  $\varphi_{\mathfrak{A}} := \bigvee_{q \in Q_{\text{even}}} (\varphi_1(q) \wedge \varphi_2(q))$ .

The construction of  $\varphi_2(q)$  relies heavily on the determinism of the DPSA  $\mathfrak{A}$ . If  $\mathfrak{A}$  were not deterministic, the universal quantification over all runs ending in  $q$  at a step would also capture eventually rejecting partial runs. Since there only exists a single run of  $\mathfrak{A}$  on the input word, however,  $\varphi_{\mathfrak{A}}$  has the intended meaning. Furthermore, both  $\varphi_1(q)$  and  $\varphi_2(q)$  use the observation that we are able to clear the stack every time that we reach a step. Thus, although the stack contents are not carried over between the different automata, the concatenation of the automata does not change the resulting run. Hence, we have  $\alpha \in L(\mathfrak{A})$  if, and only if,  $(\alpha, 0) \models \varphi_{\mathfrak{A}}$  and thus  $L(\mathfrak{A}) = L(\varphi_{\mathfrak{A}})$ .  $\blacktriangleleft$

Combining Lemmas 4 and 5 yields that VLDL is at least as expressive as BVPA. The construction inherits an exponential blowup from the construction of DPSAs from BVPAs. This shows one direction of Theorem 3.

In the next section we show that each VLDL formula  $\varphi$  can be transformed into an equivalent VPA with exponential blowup. Thus, the construction from the proof of Lemma 5 yields a normal form for VLDL formulas. In particular, formulas in this normal form only use temporal operators up to nesting depth three.

**► Proposition 6.** *Let  $\varphi$  be a VLDL formula. There exists an equivalent formula  $\varphi' = \bigvee_{i=1}^n (\langle \mathfrak{A}_{i,1} \rangle (\varphi_{st} \wedge [\mathfrak{A}_{i,2}] \neg \varphi_{st}) \wedge [\mathfrak{A}_{i,1}] (\varphi_{st} \rightarrow \langle \mathfrak{A}_{i,3} \rangle \varphi_{st}))$ , for some  $n$  that is doubly-exponential in  $|\varphi|$ , where all  $\mathfrak{A}_{i,j}$  share the same underlying VPS,  $\varphi_{st}$  is fixed over all  $\varphi$ , and neither  $\mathfrak{A}_{i,j}$  nor  $\varphi_{st}$  contain tests.*

Proposition 6 shows that tests are syntactic sugar but removing them incurs a doubly-exponential blowup. It remains open whether this blowup can be avoided.

## 4.2 From VLDL to 1-AJA

We now construct, for a given VLDL formula  $\varphi$ , an equivalent BVPA  $\mathfrak{A}_{\varphi}$ . A direct construction would incur a non-elementary blowup due to the unavoidable exponential blowup of complementing BVPAs. Moreover, it would be difficult to handle runs of the VPAs over finite words and their embedded tests, which run in parallel. Thus, we extend a construction from [6], where a similar challenge was addressed using alternating automata. Instead of alternating visibly pushdown automata, however, we use one-way alternating jumping automata (1-AJA) [4], which can be translated into equivalent BVPAs of exponential size.

A 1-AJA  $\mathfrak{A} = (Q, \tilde{\Sigma}, \delta, I, \Omega)$  consists of a finite state set  $Q$ , a visibly pushdown alphabet  $\tilde{\Sigma}$ , a set  $I \subseteq Q$  of initial states, a transition function  $\delta: Q \times \Sigma \rightarrow \mathcal{B}^+(\text{Comms}_Q)$ , with  $\text{Comms}_Q := \{\rightarrow, \rightarrow_a\} \times Q \times Q$ , where  $\mathcal{B}^+(\text{Comms}_Q)$  denotes the set of positive Boolean formulas over  $\text{Comms}_Q$ , and a coloring  $\Omega: Q \rightarrow \mathbb{N}$ . We define  $|\mathfrak{A}| = |Q|$ . Intuitively, when the automaton is in state  $q$  at position  $k$  of the word  $\alpha = \alpha_0\alpha_1\alpha_2 \dots$  it guesses a set of commands  $R \subseteq \text{Comms}_Q$  that is a model of  $\delta(q, \alpha_k)$ . It then spawns one copy of itself for

$\alpha$	c	l	c	r	r	c	c	l	r	l	l	...	
$q$	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$	$q_7$	$q_8$	$q_9$	$q_{10}$	$q_{11}$	...
$\gamma$	$\perp$	$A$	$A$	$A$	$A$	$\perp$	$A$	$A$	$A$	$A$	$A$	$A$	...
	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	...

■ **Figure 3** Run of a VPA  $\mathfrak{A}$  on the word *clrrrclrl*.

each command  $(d, q, q') \in R$  and executes the command with that copy. If  $d = \rightarrow_a$  and if  $\alpha_k$  is a matched call, the copy jumps to the position of the matching return of  $\alpha_k$  and transitions to state  $q'$ . Otherwise the automaton advances to position  $k + 1$  and transitions to state  $q$ . All copies of  $\mathfrak{A}$  continue in parallel. A single copy of  $\mathfrak{A}$  is successful if the highest color visited infinitely often is even. A 1-AJA accepts  $\alpha$  if all of its copies are successful.

► **Lemma 7** ([4]). *For every 1-AJA  $\mathfrak{A}$  there exists an effectively constructible equivalent BVPA  $\mathfrak{A}_{vp}$  with  $|\mathfrak{A}_{vp}| \in \mathcal{O}(2^{|\mathfrak{A}|})$ .*

For a given VLDL formula  $\varphi$  we now inductively construct a 1-AJA that recognizes the same language as  $\varphi$ . The main difficulty lies in the translation of formulas of the form  $\langle \mathfrak{A} \rangle \varphi$ , since these require us to translate TVPAs over finite words into 1-AJAs over infinite words. We do so by adapting the idea for the translation from BVPAs to 1-AJAs from [4] and by combining it with the bottom-up translation from LDL into alternating automata in [6].

► **Lemma 8.** *For any VLDL formula  $\varphi$  there exists an effectively constructible equivalent 1-AJA  $\mathfrak{A}_\varphi$  with  $|\mathfrak{A}_\varphi| \in \mathcal{O}(|\varphi|^2)$ .*

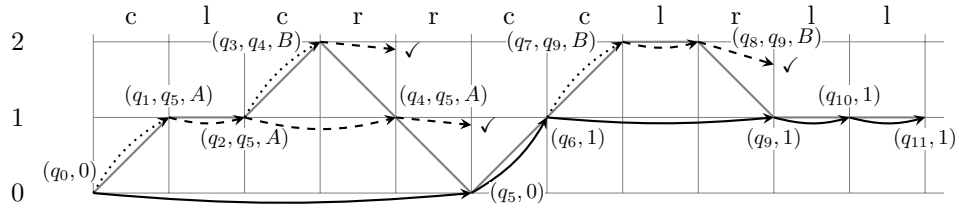
**Proof.** We construct the automaton inductively over the structure of  $\varphi$ . The case  $\varphi = p$  is trivial. For Boolean operations, we obtain  $\mathfrak{A}_\varphi$  by closure of 1-AJAs under these operations [4]. If  $\varphi = [\mathfrak{A}]\varphi'$  we use the identity  $[\mathfrak{A}]\varphi' \equiv \neg(\mathfrak{A})\neg\varphi'$  and construct  $\mathfrak{A}_{\neg(\mathfrak{A})\neg\varphi'}$  instead.

We now consider  $\varphi = \langle \mathfrak{A} \rangle \varphi'$ , where  $\mathfrak{A}$  is some TVPA and sketch the construction of  $\mathfrak{A}_\varphi$ . By induction we obtain a 1-AJA  $\mathfrak{A}'$  equivalent to  $\varphi'$ .  $\mathfrak{A}_\varphi$  simulates a run of  $\mathfrak{A}$  on a prefix of  $\alpha$  and, upon acceptance, nondeterministically transitions into  $\mathfrak{A}'$ .

Consider an initial run of  $\mathfrak{A}$  on such a prefix  $w$ . Since  $w$  is finite,  $steps(w)$  is finite as well. Hence, each stack height may only be encountered finitely often at a step. At the last visit to a step of a given height,  $\mathfrak{A}$  either accepts, or it reads a call action. The symbol pushed onto the stack in that case does not influence the remainder of the run. Such a run on the word *clrrrclrl* is shown in Figure 3, where  $c$  is a call,  $r$  is a return, and  $l$  is a local action.

The idea for the simulation of the run of  $\mathfrak{A}$  is to have a main copy of  $\mathfrak{A}_\varphi$  that jumps along the steps of the input word. When  $\mathfrak{A}_\varphi$  encounters a call  $c \in \Sigma_c$  it guesses whether or not  $\mathfrak{A}$  encounters the current stack height again. If it does, then  $\mathfrak{A}_\varphi$  guesses  $q', q'' \in Q$  and  $A \in \Gamma$  such that  $(q, c, q', A)$  is a transition of  $\mathfrak{A}$ , it jumps to the matching return of  $c$  in state  $q''$  and spawns a copy that verifies that  $\mathfrak{A}$  can go from the configuration  $(q', A)$  to the configuration  $(q'', \perp)$ . If  $\mathfrak{A}$  never returns to the current stack height, then  $\mathfrak{A}_\varphi$  only guesses  $q' \in Q$  and  $A \in \Gamma$  such that  $(q, c, q', A)$  is a transition of  $\mathfrak{A}$ , moves to state  $q'$ , and stores in its state space that it may not read any returns anymore. This is repeated until the main copy guesses that  $\mathfrak{A}'$  accepts.

The run of such a 1-AJA corresponding to the run of  $\mathfrak{A}$  shown in Figure 3 is shown in Figure 4. The gray line indicates the stack height, while the solid and dashed black paths represent the run of the main automaton and of the verifying automata, respectively. Dotted lines indicate spawning a verifying automaton. For readability, the figure does not include copies of the automata that are spawned to verify that the tests of  $\mathfrak{A}$  hold true. States of



■ **Figure 4** Behavior of 1-AJA on the word *clrrcclrl*.

the form  $(q, 0)$  denote the main copy of the automaton that has not yet ignored any call actions, while states of the form  $(q, 1)$  denote copies that have done so. The states  $(q, q', A)$  denote verification copies that verify  $\mathfrak{A}$ 's capability to move from the configuration  $(q, A)$  to the configuration  $(q', \perp)$ . The verification automata work similarly to the main automaton, except that they assume that all pushed symbols to be eventually popped and reject if they encounter an unmatched call. Details can be found in the full version [12]. ◀

By combining Lemmas 7 and 8 we see that BVPAs are at least as expressive as VLDL formulas. This proves the direction from logic to automata of Theorem 3. The construction via 1-AJAs yields automata of exponential size in the number of states. This blowup is unavoidable, which can be shown by relying on the analogous lower bound for translating LTL into Büchi automata, obtained by encoding an exponentially bounded counter in LTL.

► **Lemma 9.** *There exists a pushdown alphabet  $\tilde{\Sigma}$  such that for all  $n \in \mathbb{N}$  there exists a language  $L_n$  that is defined by a VLDL formula over  $\tilde{\Sigma}$  of polynomial size in  $n$ , but every BVPA over  $\tilde{\Sigma}$  recognizing  $L_n$  has at least exponentially many states in  $n$ .*

After having shown that VLDL has the same expressiveness as BVPAs, we now turn our attention to several decision problems for this logic. Namely, we study the satisfiability and the validity problem, as well as the model checking problem. Moreover, we consider the problem of solving visibly pushdown games with VLDL winning conditions.

## 5 Satisfiability and Validity are ExpTime-complete

We say that a VLDL formula  $\varphi$  is satisfiable if it has a model. Dually, we say that  $\varphi$  is valid if all words are models of  $\varphi$ . Instances of the satisfiability and validity problem consist of a VLDL formula  $\varphi$  and ask whether  $\varphi$  is satisfiable and valid, respectively. Both problems are decidable in exponential time. We also show both problems to be EXPTIME-hard.

► **Theorem 10.** *The satisfiability and the validity problem for VLDL are EXPTIME-complete.*

**Proof.** Due to duality, we only show EXPTIME-completeness of the satisfiability problem. Membership follows from the 1-AJA-emptiness-problem being in EXPTIME [4] and Lemma 8.

We show EXPTIME-hardness by a polynomial-time reduction from the word problem for polynomially space-bounded alternating Turing machines. Our proof is based on the reduction of this problem to the problem of model checking pushdown systems against LTL specifications from the full version of [3]. In that reduction, a run of an alternating Turing machine is encoded as a pair of a pushdown system, which checks the general format of the encoding using its stack, and an LTL specification, which checks additional properties without using a stack. We adapt this proof by checking the properties asserted by the pushdown system with a visibly pushdown automaton. Also, we encode the specification of the general format in a VLDL formula. Technical details can be found in the full version [12]. ◀

## 6 Model Checking is ExpTime-complete

We now consider the model checking problem for VLDL. An instance of the model checking problem consists of a VPS  $\mathcal{S}$ , an initial state  $q_I$  of  $\mathcal{S}$ , and a VLDL formula  $\varphi$  and asks whether  $\text{traces}(\mathcal{S}, q_I) \subseteq L(\varphi)$  holds true, where  $\text{traces}(\mathcal{S}, q_I)$  denotes the set obtained by mapping each run of  $\mathcal{S}$  starting in  $q_I$  to its sequence of labels. This problem is decidable in exponential time due to Lemma 8 and an exponential-time model checking algorithm for 1-AJAs [4]. Moreover, the problem is EXPTIME-hard, as it subsumes the validity problem.

► **Theorem 11.** *Model checking VLDL specifications against VPS's is EXPTIME-complete.*

**Proof.** Membership in EXPTIME follows from Lemma 8 and the membership of the problem of checking visibly pushdown systems against 1-AJA specifications in EXPTIME [4]. Moreover, since the validity problem for VLDL is EXPTIME-hard and since validity of  $\varphi$  is equivalent to  $\text{traces}(\mathcal{S}_{univ}) \subseteq \varphi$ , where  $\mathcal{S}_{univ}$  with  $\text{traces}(\mathcal{S}_{univ}) = \Sigma^\omega$  is effectively constructible in constant time, the model checking problem for VLDL is EXPTIME-hard as well. ◀

## 7 Solving VLDL Games is 3ExpTime-complete

In this section we investigate visibly pushdown games with winning conditions given by VLDL formulas. We consider games with two players, called Player 0 and Player 1, respectively.

A two-player game with VLDL winning condition  $\mathcal{G} = (V_0, V_1, \Sigma, E, v_I, \ell, \varphi)$  consists of two disjoint, at most countably infinite sets  $V_0$  and  $V_1$  of vertices, where we define  $V := V_0 \cup V_1$ , a finite alphabet  $\Sigma$ , an initial state  $v_I \in V$ , a set of edges  $E \subseteq V \times V$ , a labeling  $\ell: V \rightarrow \Sigma$ , and a VLDL formula  $\varphi$  over some partition of  $\Sigma$ , called the winning condition.

A play  $\pi = v_0 v_1 v_2 \dots$  of  $\mathcal{G}$  is an infinite sequence of vertices of  $\mathcal{G}$  with  $(v_i, v_{i+1}) \in E$  for all  $i \geq 0$ . The play  $\pi$  is initial if  $v_0 = v_I$ . It is winning for Player 0 if  $\ell(v_1)\ell(v_2)\ell(v_3)\dots$ <sup>2</sup> is a model of  $\varphi$ . Otherwise  $\pi$  is winning for Player 1.

A strategy for Player  $i$  is a function  $\sigma: V^*V_i \rightarrow V$ , such that  $(v, \sigma(w \cdot v)) \in E$  for all  $v \in V_i, w \in V^*$ . We call a play  $\pi = v_0 v_1 v_2 \dots$  consistent with  $\sigma$  if  $\sigma(\pi') = v_{n+1}$  for all finite prefixes  $\pi' = v_0 \dots v_n$  of  $\pi$  with  $v_n \in V_i$ . A strategy  $\sigma$  is winning for Player  $i$  if all initial plays that are consistent with  $\sigma$  are winning for that player. We say that Player  $i$  wins  $\mathcal{G}$  if she has a winning strategy. If either player wins  $\mathcal{G}$ , we say that  $\mathcal{G}$  is determined.

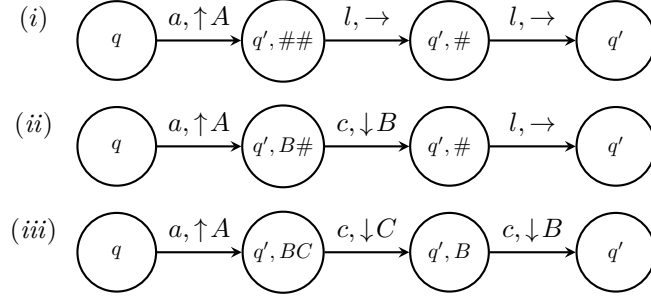
A visibly pushdown game (VPG) with a VLDL winning condition  $\mathcal{H} = (\mathcal{S}, Q_0, Q_1, q_I, \varphi)$  consists of a VPS  $\mathcal{S} = (Q, \tilde{\Sigma}, \Gamma, \Delta)$ , a partition of  $Q$  into  $Q_0$  and  $Q_1$ , an initial state  $q_I \in Q$ , and a VLDL formula  $\varphi$  over  $\tilde{\Sigma}$ . The VPG  $\mathcal{H}$  then defines the two-player game  $\mathcal{G}_{\mathcal{H}} = (V_0, V_1, \Sigma, E, v_I, \ell, \varphi)$  with  $V_i := Q_i \times ((\Gamma \setminus \{\perp\})^* \cdot \perp) \times \Sigma$ ,  $v_I = (q_I, \perp, a)$  for some  $a \in \Sigma$  (recall that the trace disregards the label of the initial vertex),  $((q, \gamma, a), (q', \gamma', a')) \in E$  if there is an  $a'$ -labeled edge from  $(q, \gamma)$  to  $(q, \gamma')$  in the configuration graph  $G_{\mathcal{S}}$ , and  $\ell: (q, \gamma, a) \mapsto a$ . Solving a VPG  $\mathcal{H}$  means deciding whether Player 0 wins  $\mathcal{G}_{\mathcal{H}}$ .

► **Proposition 12.** *VPGs with VLDL winning conditions are determined.*

**Proof.** Since each VLDL formula defines a language in  $\omega$ -VPL due to Theorem 3, each VPG with VLDL winning condition is equivalent to a VPG with an  $\omega$ -VPL winning condition. These are known to be determined [8]. ◀

<sup>2</sup> Note that the sequence of labels trace omits the label of the first vertex for technical reasons.





■ **Figure 5** Construction of a VPG from a pushdown game for transitions of the forms (i)  $(q, a, A, q', \varepsilon)$ , (ii)  $(q, a, A, q', B)$ , and (iii)  $(q, a, A, q', BC)$ .

We show that solving VPGs with winning conditions specified in VLDL is harder than solving VPGs with winning conditions specified by BVPAs, i.e., they can be solved in triply exponential time. Moreover, they are complete for this complexity class.

► **Theorem 13.** *Solving VPGs with VLDL winning conditions is 3EXPTIME-complete.*

**Proof.** We solve VPGs with VLDL winning conditions by constructing a BVPA  $\mathfrak{A}_\varphi$  from the winning condition  $\varphi$  and by then solving the visibly pushdown game with a BVPA winning condition [8]. This approach takes triply-exponential time in  $|\varphi|$  and exponential time in  $|\mathcal{S}|$ .

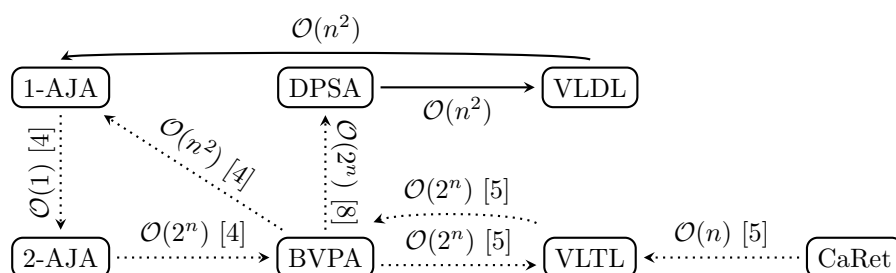
We show 3EXPTIME-hardness of the problem by a reduction from solving pushdown games with LTL winning conditions, which is known to be 3EXPTIME-complete [8]. Instead of, e.g., popping one symbol off the stack and pushing two others onto it, the resulting VPG splits these operations into individual pop- and push-operations, which are then carried out sequentially. The actions that still have to be carried out can be tracked using additional vertices. Since each stack operation can be split into at most three individual operations, this incurs only a linear blowup in the size of both the game and the winning condition.

A pushdown game with an LTL winning condition  $\mathcal{H} = (\mathcal{S}, V_I, V_O, \psi)$  is defined similarly to a VPG, except for the relaxation that  $\mathcal{S}$  may now be a traditional pushdown system instead of a visibly pushdown system. Specifically, we have  $\Delta \subseteq (Q \times \Gamma \times \Sigma \times Q \times \Gamma^{\leq 2})$ , where  $\Gamma^{\leq 2}$  denotes the set of all words over  $\Gamma$  of at most two letters. Stack symbols are popped off the stack using transitions of the form  $(q, A, a, q', \varepsilon)$ , the top of the stack can be tested and changed with transitions of the form  $(q, A, a, q', B)$ , and pushes are realized with transitions of the form  $(q, A, a, q', BC)$ . Additionally, the winning condition is given as an LTL formula instead of a VLDL formula. The two-player game  $\mathcal{G}_\mathcal{H}$  is defined analogously to the visibly pushdown game.

Since the pushdown game admits transitions such as  $(q, A, a, q', BC)$ , which pop  $A$  off the stack and push  $B$  and  $C$  onto it, we need to split such transitions into several transitions in the visibly pushdown game. We modify the original game such that every transition of the original game is modeled by three transitions in the visibly pushdown game, up to two of which may be dummy actions that do not change the stack. As each transition may perform at most three operations on the stack, we can keep track of the list of changes still to be performed in the state space. We perform these actions using dummy letters  $c$  and  $l$ , which we add to  $\Sigma$  and read while performing the required actions on the stack. We choose the vertices  $V'_X = V_X \cup (V_X \times (\Gamma \cup \{\#\})^{\leq 2})$  and the alphabet  $\tilde{\Sigma} = (\{c\}, \Sigma, \{l\})$ .

We transform  $\mathcal{H}$  as shown in Figure 5 and obtain the VPG  $\mathcal{H}'$ . Moreover, we transform the winning condition  $\psi$  of  $\mathcal{H}$  into  $\psi'$  by inductively replacing each occurrence of  $\mathbf{X}\psi$  by  $\mathbf{X}^3\psi'$  and each occurrence of  $\psi_1 \mathbf{U} \psi_2$  by  $(\psi'_1 \vee c \vee l) \mathbf{U} (\psi'_2 \wedge \neg c \wedge \neg l)$ . We subsequently translate





■ **Figure 6** Formalisms capturing (subsets of)  $\omega$ -VPL and translations between them.

the resulting LTL formula  $\psi'$  into an equivalent VLDL formula  $\varphi$  using Lemma 1. The input player wins  $\mathcal{H}'$  with the winning condition  $\varphi$  if and only if he wins  $\mathcal{H}$  with the winning condition  $\psi$ . Hence, solving VPGs with VLDL winning conditions is 3EXPTIME-hard. ◀

## 8 Conclusion

We have introduced Visibly Linear Dynamic Logic (VLDL) which strengthens Linear Dynamic Logic (LDL) by replacing the regular languages used as guards in the latter logic with visibly pushdown languages. VLDL captures precisely the class of  $\omega$ -visibly pushdown languages. We have provided effective translations from VLDL to BVPA and vice versa with an exponential blowup in size in both directions. From automata to logic, this blowup cannot be avoided while it remains open whether or not it can be avoided in the other direction.

Figure 6 gives an overview over the known formalisms that capture  $\omega$ -VPL and the translations between them. Our constructions are marked by solid lines, all others by dotted lines. All constructions are annotated with the blowup they incur.

In particular, there exist translations between VLTL and VLDL via BVPA that incur a doubly-exponential blowup in both directions, as shown in Figure 6. In spite of this blowup the satisfiability problem and the model checking problem for both logics are EXPTIME-complete. It remains open whether there exist efficient translations between the two logics.

We showed the satisfiability and the emptiness problem for VLDL, as well as model checking visibly pushdown systems against VLDL specifications, to be EXPTIME-complete. Also, we proved that solving visibly pushdown games with VLDL winning conditions is 3EXPTIME-complete.

Extending VLDL by replacing the guards with a more expressive family of languages quickly yields undecidable decision problems. In fact, using deterministic pushdown languages as guards already renders all decision problems discussed in this work undecidable [12].

In contrast to LDL [10] and VLTL [5], VLDL uses automata to define guards instead of regular or visibly rational expressions. We are currently investigating a variant of VLDL where the VPAs guarding the temporal operators are replaced by visibly rational expressions (with tests), which is closer in spirit to LDL.

**Acknowledgments.** The authors would like to thank Laura Bozzelli for providing the full version of [4] and Christof Löding for pointing out the 3EXPTIME-hardness of solving infinite games for visibly pushdown games against LTL specifications.

---

**References**

---

- 1 Rajeev Alur, Kousha Ettesami, and Parthasarathy Madhusudan. A temporal logic of nested calls and returns. In *TACAS 2004*, volume 2988 of *LNCS*, pages 467–481. Springer, 2004.
- 2 Rajeev Alur and Parthasarathy Madhusudan. Visibly Pushdown Languages. In *STOC 2004*, pages 202–211. ACM, 2004.
- 3 Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR 1997*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997. Full version available at <http://www.liafa.univ-paris-diderot.fr/~abou/BEM97.pdf>.
- 4 Laura Bozzelli. Alternating Automata and a Temporal Fixpoint Calculus for Visibly Pushdown Languages. In L. Caires and V. T. Vasconcelos, editors, *CONCUR 2007*, volume 4703 of *LNCS*, pages 476–491. Springer, 2007.
- 5 Laura Bozzelli and César Sánchez. Visibly linear temporal logic. In *IJCAR 2014*, volume 8562 of *LNCS*, pages 418–483, 2014.
- 6 Peter Faymonville and Martin Zimmermann. Parametric Linear Dynamic Logic. *Inf. Comput.*, 2016. Article in press.
- 7 Martin Leucker and César Sánchez. Regular linear temporal logic. In C. B. Jones, Z. Liu, and J. Woodcock, editors, *ICTAC 2007*, number 4711 in *LNCS*, pages 291–305, 2007.
- 8 Christof Löding, Parthasarathy Madhusudan, and Olivier Serre. Visibly Pushdown Games. In L. Lodaya and M. Mahajan, editors, *FSTTCS 2004*, volume 3328 of *LNCS*, pages 408–420. Springer, 2005.
- 9 Amir Pnueli. The temporal logic of programs. In *FOCS 1977*, pages 46–57. IEEE, 1977.
- 10 Moshe Vardi. The Rise and Fall of LTL. In G. D’Agostino and S. La Torre, editors, *EPTCS 54*, 2011.
- 11 Moshe Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. and Comp.*, 115:1–37, 1994.
- 12 Alexander Weinert and Martin Zimmermann. Visibly linear dynamic logic. *arXiv*, 1512.05177, 2015.
- 13 Pierre Wolper. Temporal logic can be more expressive. *Inf. and Cont.*, 56:72–99, 1983.

# Stable Matching Games: Manipulation via Subgraph Isomorphism

Sushmita Gupta<sup>1</sup> and Sanjukta Roy<sup>2</sup>

1 Institute for Informatics, University of Bergen, Norway  
Sushmita.Gupta@uib.no, sushmita.gupta@gmail.com

2 The Institute of Mathematical Sciences, HBNI, Chennai, India  
sanjukta@imsc.res.in

---

## Abstract

In this paper we consider a problem that arises from a strategic issue in the stable matching model (with complete preference lists) from the viewpoint of exact-exponential time algorithms. Specifically, we study the STABLE EXTENSION OF PARTIAL MATCHING (SEOPM) problem, where the input consists of the complete preference lists of men, and a partial matching. The objective is to find (if one exists) a set of preference lists of women, such that the men-optimal Gale Shapley algorithm outputs a perfect matching that *contains* the given partial matching. Kobayashi and Matsui [*Algorithmica*, 2010] proved this problem is NP-complete. In this article, we give an exact-exponential algorithm for SEOPM running in time  $2^{\mathcal{O}(n)}$ , where  $n$  denotes the number of men/women. We complement our algorithmic finding by showing that unless Exponential Time Hypothesis (ETH) fails, our algorithm is asymptotically optimal. That is, unless ETH fails, there is no algorithm for SEOPM running in time  $2^{\mathcal{O}(n)}$ . Our algorithm is a non-trivial combination of a parameterized algorithm for SUBGRAPH ISOMORPHISM, a relationship between stable matching and finding an out-branching in an appropriate graph and enumerating non-isomorphic out-branchings.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** stable matching, Gale-Shapley algorithm, suitor graph, subgraph isomorphism, exact-exponential time algorithms

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.29

## 1 Introduction

STABLE MATCHING together with its in numerous variants are among the most well-studied problems in matching theory, driven by applications to economics, business, engineering, and more recently medical sciences. In the two-sided STABLE MATCHING problem (also called the STABLE MARRIAGE problem), we are given two sets of agents of equal size, known as *men* and *women*, where each person submits a ranked list of all the members of the opposite sex. In this setting, a *matching* is a set of man-woman pairs (called matching partners), no two of which share a common member. A **stable matching** is a matching for which there does not exist a *blocking pair*: a man and a woman, who are not part of a matching pair, but prefer each other to their respective matching partners.

Ever since the theoretical framework for STABLE MATCHING was laid down by Gale and Shapley [9] to study the then current heuristic used to assign medical residents to hospitals in New England, the topic has received considerable attention from theoreticians and practitioners alike. In particular, it is one of the foundational problems in social choice theory, where a matching is viewed as an *allocation* or *assignment* of resources to



© Sushmita Gupta and Sanjukta Roy;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 29; pp. 29:1–29:14



Leibniz International Proceedings in Informatics

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

relevant agents, whereby the nature of the assignment can vary greatly depending on the scenario/marketplace they are modelling. We refer the reader to books [10, 18, 14] for an in-depth introduction to stable matching and its variants.

Gale and Shapley [9] showed that every instance of the STABLE MATCHING problem admits a stable matching. In other words, given any set of preference lists of men and women there exists at least one stable matching. In fact, they gave a polynomial time algorithm to find a stable matching. This algorithm is widely used in both practice and theory, and it exists in two versions: the *men-optimal* and the *women-optimal*, so named to emphasise the fact that one side prefers one over the other. Both variants are defined analogously. As the name suggests, the men-optimal stable matching is a stable matching that is no worse than any other stable matching, in terms of the preferences of the men. In other words, there does not exist a stable matching such that each man prefers his partner in that matching to his partner in the men-optimal stable matching. The algorithm that yields the men-optimal stable matching is called the *men-proposing* (resp. *women-proposing*) Gale-Shapley algorithm. The men-proposing version of the algorithm works as follows. A man who is not yet matched to a woman, *proposes* to the woman who is at the top of his current list, which is obtained by removing from his original preference list, all the women who have rejected him at an earlier step. On the woman's side, when a woman  $w$  receives a proposal from a man  $m$ , she accepts the proposal if it is her first proposal, or if she prefers  $m$  to her current partner. If  $w$  prefers her current partner to  $m$ , then  $w$  *rejects*  $m$ . If  $m$  is rejected by  $w$ , then  $m$  removes  $w$  from his list. This process continues until there is no unmatched man. The output of this algorithm is the men-optimal stable matching. For more details, see [10]. It has been customary to use the men-proposing version of the algorithm, and our analysis here will stick to that convention. Henceforth, unless explicitly stated otherwise, any mention of a stable matching should be interpreted by the reader as such. We will use  $(\mathcal{L}^M, \mathcal{L}^W)$  to denote the set of preference lists of men and women, and the men-optimal matching with respect to these lists is denoted by  $\text{GS}(\mathcal{L}^M, \mathcal{L}^W)$ .

## 1.1 Our problem and motivation

Kobayashi and Matsui [12, 13] studied manipulation in the stable matching model, where agents are manipulating with the goal of attaining a specific matching target. Formally speaking, they considered the following class of problems. An input consists of two sets  $M$  and  $W$ , (each of size  $n$ ) of *men* and *women*, respectively; along with the preference list of every man (expressed as a strict ordering on the set of women) (denoted by  $\mathcal{L}^M$ ) and a matching on  $(M, W)$ . The said matching can either be *perfect* (if it contains  $n$  pairs), or *partial* (possibly, fewer than  $n$  pairs). Furthermore, for a couple of problems, we are given a set of preference lists of women,  $\mathcal{L}^{W'}$ , where  $W' \subseteq W$ . The goal is to decide if there exists a set of preference lists of women,  $\mathcal{L}^W$ , containing  $\mathcal{L}^{W'}$ , such that when used in conjunction with  $\mathcal{L}^M$  with the men-optimal stable matching algorithm, yields a matching that contains all the pairs in the stated matching. Of these problems, two are directly related to our work in this paper. Let us consider the following two problems, and compare and contrast their computational complexity.

ATTAINABLE STABLE MATCHING (ASM)

**Input:** A set of preference lists  $\mathcal{L}^M$  of men over women  $W$ , and a perfect matching  $\mu$  on  $(M, W)$ .

**Question:** Does there exist a set of preference lists of women  $\mathcal{L}^W$ , such that  $\text{GS}(\mathcal{L}^M, \mathcal{L}^W) = \mu$ ?

Kobayashi and Matsui in [12, 13] showed that ASM is polynomial time solvable, and exhibited an  $\mathcal{O}(n^2)$  algorithm that computes the set  $\mathcal{L}^W$ , if it exists. Or else, reports “none exists”. The following problem is identical to the above, except in one key aspect: *the target matching need not be perfect*. The authors show that this problem is NP-complete.

STABLE EXTENSION OF PARTIAL MATCHING (SEOPM)

**Input:** A set of preference lists  $\mathcal{L}^M$  of men  $M$  over women  $W$ , and a partial matching  $\mu'$  on  $(M, W)$ .

**Question:** Does there exist preferences of women  $\mathcal{L}^W$ , such that  $\mu' \subseteq \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$ ?

These two problems and their differing computational complexities represent a dichotomy with respect to the size of target matching. Kobayashi and Matsui solve ASM by designing a novel combinatorial structure called the *suitor graph*, which encodes enough information about the men’s preferences and the matching pairs, that it allows an efficient search of the possible preference lists of women, which are  $n \cdot n!$  in number. The same approach falls short when the stated matching is partial.

Our work in this paper falls thematically within the area of strategic results relating to the stable matching problem. There is a long history of results centred around the question as to whether an individual agent, or a coalition of agents can misstate their true preference lists (either by truncating, or by permuting the list), with the objective of obtaining a better partner (assessed in terms of the true preferences of the manipulating agents) than would otherwise be possible under the men-optimal stable matching algorithm. SEOPM is to be viewed as a manipulation game in which a coalition of agents (in this case the subset of women who are matched under the partial matching) have decided upon a specific partner. These agents are colluding, with co-operation from the other women who are not matched, to produce a perfect matching, which gives each of the manipulating agents their target partners. There exists a strategy to attain this objective if and only if there exists a set of preference list of women that yields a perfect matching that contains the partial matching.

Since SEOPM has been shown to be NP-complete, it is natural to study this problem in computational paradigms that are meant to cope with NP-hardness. We attempt such a study in the area of exact exponential time algorithms. Manipulation and strategic issues in voting have been well-studied in the field of exact algorithms and parameterized complexity; see the survey [3] for an overview. But one can not say the same regarding the strategic issues in the stable matching model. These problems hold a lot of promise and remain hitherto unexplored in the light of exact algorithms and parameterized complexity, with exceptions that are few and far between [15, 16].

To the best of our knowledge, Cseh and Manlove [4] initiated this type of analysis by studying an NP-hard variant of the stable marriage and *stable roommate* problems<sup>1</sup>, where the input consists of each of the preference lists, as well two subsets of (not necessarily pairwise disjoint) pairs of agents, representing the *forbidden pairs* and the *forced pairs*. The goal is to find a matching that does not contain any of the forbidden pairs, and contains each of the forced pairs, while simultaneously minimizing the number of blocking pairs.

<sup>1</sup> In the stable roommate problem, the matching market consists of agents of the same type, as opposed to the market modelled the stable marriage problem that consists of agents of two types, men and women. Roommate assignments in college housing facilities is a real world application of the stable roommate problem.

## 1.2 Our Contributions

Throughout the article,  $n$  is used to denote  $n = |M| = |W|$ . The most basic algorithm for SEOPM would be to guess the permutation of all women (that is, the set of preferences of women,  $\mathcal{L}^W$ ) and check whether  $\mu' \subseteq \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$ . However, this algorithm will take  $(n!)^n n^2 = 2^{\mathcal{O}(n^2 \log n)}$ . One can obtain an improvement over this naïve algorithm by using the polynomial time algorithm for ASM [13]. That is, using the algorithm for ASM, which given a matching  $\mu$  can check in polynomial time whether there exists  $\mathcal{L}^W$  such that  $\mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$ . The faster algorithm for SEOPM, using the algorithm for ASM, tries all possible extensions of the partial matching  $\mu \supseteq \mu'$  and checks in polynomial time whether there exists  $\mathcal{L}^W$  such that  $\mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$ . Thus, if the size of the partial matching is  $k$ , this algorithm would have to try  $(n - k)!$  possibilities. In the worst case this can take  $(n!)n^{\mathcal{O}(1)} = 2^{\mathcal{O}(n \log n)}$ .

In this article we give a  $2^{\mathcal{O}(n)}$  algorithm, which not only breaks the naïve bound, but also uses an idea which connects SEOPM to the problem of COLORED SUBGRAPH ISOMORPHISM (given two graphs  $G$  and  $H$ , the objective is to test whether  $H$  is isomorphic to some subgraph of  $G$ ). We establish this connection by introducing a combinatorial tool, the *universal suitor graph* that extends the notion of the rooted *suitor graph* devised by Kobayashi and Matsui in [12, 13], to solve ASM. It is shown in [13] that an input instance  $(\mathcal{L}^M, \mu)$  of ASM is a YES-instance if and only if the corresponding rooted suitor graph has an *out-branching*: a spanning subgraph in which every vertex has at most one in-coming arc, and is reachable from the root. The universal suitor graph satisfies the property that  $(\mathcal{L}^M, \mu')$ , an instance of SEOPM is a YES-instance if and only if the corresponding universal suitor graph contains a subgraph that is isomorphic to the out-branching corresponding to  $(\mathcal{L}^M, \mu)$  where  $\mu$  is the perfect matching that “extends”  $\mu'$ . Thus, the universal suitor graph succinctly encodes all “possible suitor graphs” and is only polynomially larger than the size of a suitor graph. That is, the size of universal suitor graph is  $\mathcal{O}(n^2)$ . This is our main conceptual contribution and we believe that the concept of the universal suitor graph is likely to be of independent interests, useful in characterizing existence of strategies in other manipulation games.

Using ideas from exact exponential algorithms and parameterized complexity; in particular by using as a subroutine the algorithm that enumerates all non-isomorphic out-branchings in a (given) rooted directed graph [2, 17], and a parameterized algorithm for COLORED SUBGRAPH ISOMORPHISM [1, 7, 8], we can search for a subgraph in the universal suitor graph that is isomorphic to an out-branching corresponding to an extension of  $\mu'$ . We complement our algorithmic finding by showing that unless Exponential Time Hypothesis (ETH) fails, our algorithm is asymptotically optimal. That is, unless ETH fails, there is no algorithm for SEOPM running in time  $2^{\mathcal{O}(n)}$ . We refer to the following books for further reading regarding exact algorithms [6] and parameterized complexity [5].

## 2 Preliminaries

For a positive integer  $n$ , we will use  $[n]$  to denote the set  $\{1, 2, \dots, n\}$ . As introduced earlier,  $M$  and  $W$  denote the set of men and women, respectively, and we assume that  $|M| = |W| = n$ . Each  $m \in M$  has a preference list, denoted by  $P(m)$ , which is a total ordering of  $W$ . The set of preference lists of all men is denoted by  $\mathcal{L}^M$ . Similarly, each  $w \in W$  has a preference list, denoted by  $P(w)$  which is a total ordering of  $M$ . The set of preference lists of all women is denoted by  $\mathcal{L}^W$ . It is helpful to view  $(M, W)$  as the bipartitions of a complete bipartite graph, and a perfect matching in  $(M, W)$  as a set of vertex disjoint edges that matches every vertex in  $M \cup W$ . Similarly, a partial matching in  $(M, W)$  can be viewed as a set of vertex disjoint edges that does not necessarily match every vertex in  $M \cup W$ .

Given a matching  $\mu$  (perfect or partial), and a vertex  $v \in M \cup W$ ,  $\mu(v)$  denotes the matched partner of the man/woman  $v$ . We note that for a perfect matching  $\mu$ :  $m \in M$  if and only if  $\mu(m) \in W$ , and similarly  $w \in W$  if and only if  $\mu(w) \in M$ . But, when we have a partial matching,  $\mu$ , it may be that some vertices (male or female) are not matched under it, we denote that symbolically as  $\mu(v) = v$  for any man/woman  $v \in M \cup W$  who is not matched in  $\mu$ . A matching  $\mu$  is said to be an **extension** of a matching  $\mu'$  if  $\mu' \subseteq \mu$ , that is  $\mu$  contains the set of edges in  $\mu'$ . For any matching  $\mu$ , and a man  $m$  matched in  $\mu$ , we define  $\delta^+(m) = \{w \in W \mid m \text{ strictly prefers } w \text{ to } \mu(m)\}$ , and conversely for any woman  $w \in W$  (not necessarily matched in  $\mu$ ) we define  $\delta^-(w) = \{m \in M \mid m \text{ strictly prefers } w \text{ to } \mu(m)\}$ ; all preferences are in terms of lists in  $\mathcal{L}^M$ .

Throughout the paper, we use the standard notations about directed graphs. Given a directed graph  $D$ , and a vertex  $v \in V(D)$ , we use  $N^-(v)$  to denote the set of vertices that are in-neighbors of  $v$ :  $N^-(v) = \{u \mid (u, v) \in E(D)\}$ . Similarly, we use  $N^+(v)$  to denote the set of vertices that are out-neighbors of  $v$ :  $N^+(v) = \{u \mid (v, u) \in E(D)\}$ . Following the usual notations, a source is a vertex  $v$  such that  $N^-(v) = \emptyset$  and a sink is a vertex  $v$  such that  $N^+(v) = \emptyset$ . An **out-branching** is a directed graph with a special vertex, called the *root*, where each vertex is reachable from the root by exactly one directed path. Essentially, this is a rooted tree with all arcs oriented away from the root. For any directed edge or an arc, *tail* is the vertex from where the arc originates and the *head* is the vertex at which it ends.

### 3 Generalization of Suitor Graph

The main tool we use to obtain our exact exponential time algorithm is the notion of a *universal suitor graph* – a generalization of the *suitor graph* introduced by Kobayashi and Matsui [13]. We start the section by introducing the definition of a suitor graph, followed by the definition of a universal suitor graph.

**Suitor Graph and Rooted Suitor Graph.** Given a set of preference lists  $\mathcal{L}^M$  of men over set of women  $W$  and a partial matching  $\mu'$ ,  $G(\mathcal{L}^M, \mu')$  denotes a directed bipartite graph, called a *suitor graph*, where  $V(G) = M \cup W$  and a set of directed arcs  $E(G)$  defined as follows,

$$E(G) = \left\{ (w, \mu'(w)) \in W \times M \mid w \text{ is matched in } \mu' \right\} \\ \cup \left\{ (m, w) \in M \times W \mid m \text{ is matched in } \mu', w \in \delta^+(m) \right\}.$$

Observe that the arcs for which a woman is the tail are the (only) arcs that correspond to the matched pairs in  $\mu'$ .

For a given suitor graph  $G(\mathcal{L}^M, \mu')$ , the associated *rooted suitor graph* is a directed graph  $\overline{G}(\mathcal{L}^M, \mu')$  defined as follows. We introduce an artificial vertex  $r$ , called the *root*, to  $G(\mathcal{L}^M, \mu')$  and add arcs  $(r, w)$  for every vertex  $w \in W$  that has no incoming arcs in  $G(\mathcal{L}^M, \mu')$ . That is, we add arcs from  $r$  to all the vertices that are *sources* in  $G(\mathcal{L}^M, \mu')$ . We give an example of a suitor graph and a rooted suitor graph. Figure 1 shows the suitor graph and the rooted suitor graph for the preference lists given in Table 1 and the partial matching  $\{(A, 1), (B, 2), (C, 3)\}$ . The vertex marked as  $r$  is the root vertex.

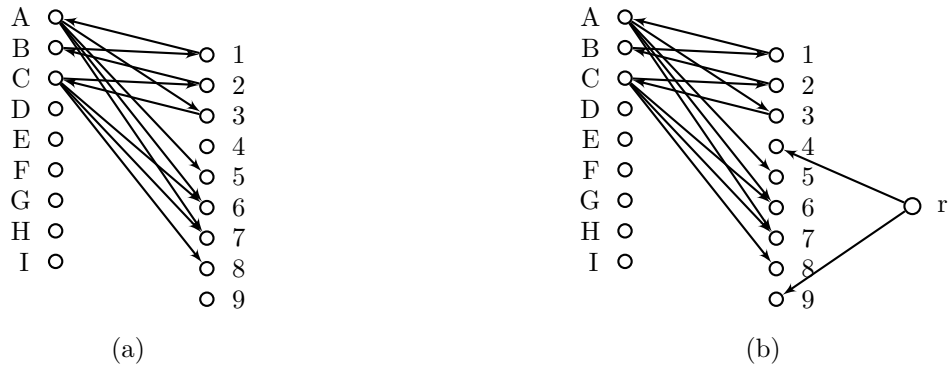
Our main motivation for suitor graph and its generalization is the following result proved in [13, Theorem 2].

► **Proposition 1** ([13]). *Let  $\mathcal{L}^M$  be a set of preference lists for  $M$ , and  $\mu$  be a perfect matching between  $(M, W)$ , then the following holds. There exists  $\mathcal{L}^W$ , a set of preference lists for*



■ **Table 1** Example: Preference List of Men over Women.

Man	Preference over women								
A	3	7	6	5	1	9	8	4	2
B	1	2	4	3	9	5	8	7	6
C	2	7	6	8	3	4	9	1	5
D	2	7	6	8	3	4	9	1	5
E	3	7	6	5	1	9	8	4	2
F	1	2	4	3	9	5	8	7	6
G	3	7	6	5	1	9	8	4	2
H	1	2	4	3	9	5	8	7	6
I	2	7	6	8	3	4	9	1	5



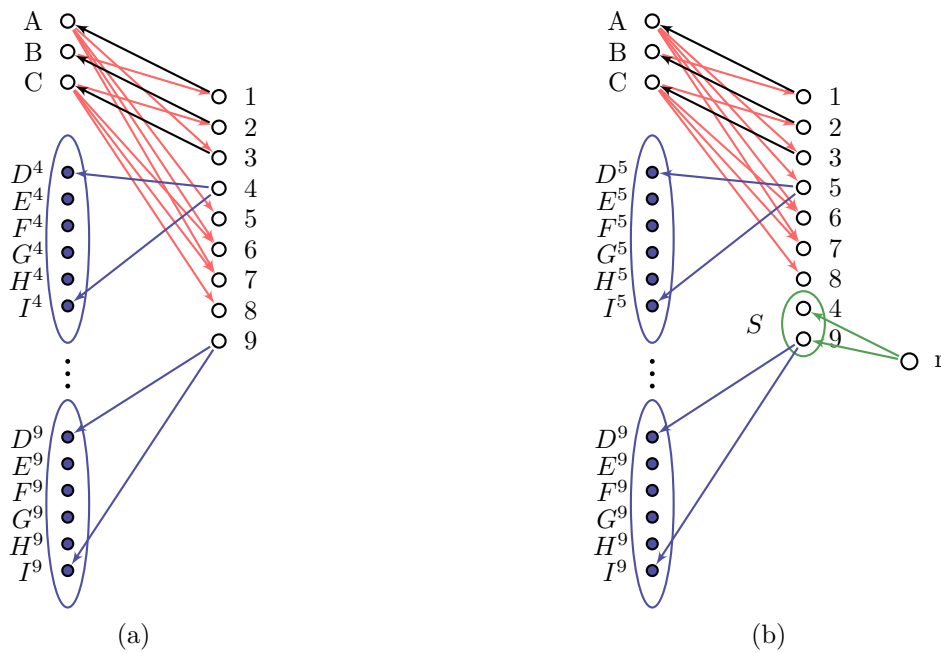
■ **Figure 1** (a) Suitor Graph, (b) Rooted Suitor Graph.

$W$  such that  $\text{GS}(\mathcal{L}^M, \mathcal{L}^W) = \mu$  if and only if the rooted suitor graph  $\bar{G}(\mathcal{L}^M, \mu)$  has an out-branching.

There exists a polynomial time algorithm that takes as input  $(\mathcal{L}^M, \mu)$  and outputs  $\mathcal{L}^W$  (if one exists) such that  $\text{GS}(\mathcal{L}^M, \mathcal{L}^W) = \mu$ . Otherwise, it reports “none exists”. We will be using this as a subroutine in our algorithm which will be presented in a later section.

**Universal Suitor Graph.** Next we define *universal suitor graph (USG)*. The idea is to construct a graph that given a set of preference lists  $\mathcal{L}^M$  of men over women captures all possible suitor graphs succinctly. Then we make use of this to solve our problem. Formally, given a set of preference lists  $\mathcal{L}^M$  of men over women, universal suitor graph,  $U(\mathcal{L}^M)$ , is defined as follows. We make  $n$  different copies of each man  $m_i \in M$ , denoted by  $M_i = \{m_i^1, \dots, m_i^n\}$ . Recall that for every  $m_i \in M$ , the preference list  $P(m_i) \in \mathcal{L}^M$  is given. We define  $P(m_i^j) = P(m_i)$ , for  $1 \leq j \leq n$ . Thus, the vertex set of the graph is  $V(U(\mathcal{L}^M)) = \biguplus_{i=1}^n M_i \cup W$ . The arc set,  $E(U(\mathcal{L}^M))$ , is defined as follows. For every  $w_i \in W$ , the graph contains arcs  $(w_i, m_j^i)$  for all  $1 \leq j \leq n$ . Additionally, the graph contains the arc  $(m_i^j, w_k)$  if  $m_i$  prefers  $w_k$  to  $w_j$  in  $P(m_i)$ ,  $w_k, w_j \in W$ . This condition is depicted notationally as  $w_k >_{m_i} w_j$ . The intuition behind the construction is the following: given any matching  $\mu$ , if a man  $m_j$  is matched with woman  $w_k$  then we imitate that by matching  $w_k$  to the  $k^{\text{th}}$  copy of  $m_j$ . Furthermore, using other copies of  $m_j$  we imitate connections with women whom he prefers to  $w_k$ . In particular, the  $i^{\text{th}}$  copy of every man is “paired” to  $w_i$ , i.e.,  $N^+(w_i) = \{m_1^i, m_2^i, \dots, m_n^i\}$ . This idea of pairing is captured by the fact that every male vertex in USG (consider  $m_k^i$ ) has a unique in-neighbor (the female vertex  $w_i$ ).





■ **Figure 2** (a) Universal suitor graph, (b) Rooted universal suitor graph [described later in section 4.2] for the partial matching  $\mu = \{(A, 1), (B, 2), (C, 3)\}$  with sources in  $\{4, 9\}$  (shows edges partially). Black edges represent the matching edges in  $\mu$ , red edges represent the preferences of men matched in  $\mu$ , while the blue edges represent edges from an unmatched woman to her own copies of the unmatched men. The green ellipse represents the set of source vertices,  $\{4, 9\}$ , that are connected from the root (not shown).

**Universal Suitor Graph for a Partial Matching.** For a given partial matching  $\mu$  on the set  $(M, W)$ , we define the graph,  $U(\mathcal{L}^M, \mu)$ , as follows. A man  $m \in M$  is matched under  $\mu$  if and only if  $\mu(m) \in W$ , and analogously for a woman  $w \in W$ ,  $w$  is matched under  $\mu$  if and only if  $\mu(w) \in M$ . We refer to the following set of operations collectively as the **pruning** of  $U(\mathcal{L}^M)$  w.r.t.  $\mu$ .

**Matched Women:** Let  $\mu(w_i) = m_j$ . Then delete vertices  $\{m_k^i \mid 1 \leq k \leq n, k \neq j\}$ , from the graph. This ensures that every matched female vertex  $w_i$ , has a unique out-going arc to the  $i^{th}$  copy of the man  $\mu(w_i)$ . In other words, only the arc  $(w_i, m_j^i)$ , where  $\mu(w_i) = m_j$ , survives.

**Unmatched Women:** Let  $\mu(w_i) \notin M$ . Then delete vertices  $\{m_k^i \mid 1 \leq k \leq n, \mu(m_k) \in W\}$ . That is, delete the  $i^{th}$  copy of a man who is matched under  $\mu$ . This ensures that in the subgraph, every unmatched female vertex  $w_i$  has out-going arcs to the vertices in the set  $\{m_k^i \mid m_k \text{ is unmatched in } \mu\}$ .

This completes the description of the pruning operations. Thus, to obtain the graph  $U(\mathcal{L}^M, \mu)$  we start with  $U(\mathcal{L}^M)$  and apply the pruning operations defined above with respect to the matching  $\mu$ . Edges in  $U(\mathcal{L}^M)$  that are not deleted during the above pruning operations, are said to have *survived pruning* w.r.t.  $\mu$ . We give an example of a universal suitor graph for a partial matching. Figure 2 shows the universal suitor graph and the rooted universal suitor graph [described later in section 4.2] for the preference lists given in Table 1, and for the partial matching  $\mu = \{(A, 1), (B, 2), (C, 3)\}$ . To keep the figure clear, we only show the copies of male vertices for women 4 and 9. The edges going out of these copies of the male vertices are omitted.

We conclude this discussion with a useful lemma that will be invoked in several arguments.

► **Lemma 2.** *Let  $\mu$  denote a partial matching. If a male vertex  $m_j^i$  survives pruning w.r.t.  $\mu$ , then either  $\mu(m_j) = w_i$ , or else both  $m_j$  and  $w_i$  are unmatched in  $\mu$ . Furthermore, the out-going arcs from  $m_j^i$  are also not deleted during pruning operations.*

**Proof.** We begin by noting that if  $w_i$  is matched to someone other than  $m_j$ , then the vertex  $m_j^i$  must be deleted during the pruning step; this is a contradiction. Suppose that  $w_i$  is unmatched, and  $\mu(m_j) = w_\ell$ . Then, the arc  $(w_\ell, m_j^i)$  survives, but the vertex  $m_j^i$  must be deleted, again a contradiction. Hence, the fact that  $m_j^i$  survives pruning w.r.t.  $\mu$ , implies that either  $\mu(m_j) = w_i$ , or both  $m_j, w_i$  are unmatched.

Additionally, we note that if  $\mu(m_j) = w_i$ , then  $m_j^i$  is the sole member of  $M_j$  that survives the pruning steps. Also note that regardless of whether  $m_j$  is matched or unmatched, the out-going arcs from  $m_j^i$  survive the pruning process. ◀

## 4 Exact Algorithm for SEOPM

In this section we design a moderately exponential time algorithm for SEOPM. Towards this we will combine the following three ingredients:

- the notion of a universal suitor graph defined in the previous section;
- a parameterized algorithm for SUBGRAPH ISOMORPHISM when the pattern graph has bounded treewidth; and
- the fact that the number of non-isomorphic (i.e. unlabelled) trees on  $n$  vertices is at most  $2.956^n n^{\mathcal{O}(1)}$ .

We start this section by giving an overview of our algorithms. Towards this we first give the relevant notions and definitions.

► **Definition 3.** Two digraphs  $G_1$  and  $G_2$  are said to be isomorphic if there is a function  $f : V(G_1) \rightarrow V(G_2)$  that satisfies the following properties:

1.  $f$  is a bijective function, i.e.,  $f^{-1}$  is a function from  $V(G_2)$  to  $V(G_1)$ ;
2. for every edge  $(u, v) \in E(G_1)$ , we have  $(f(u), f(v)) \in E(G_2)$ .

A function such as  $f$  is called an *isomorphism function*. This function can be extended to sets of vertices analogously. That is, for all  $V_1 \subseteq V(G_1)$ ,  $f(V_1) = \{f(v) \mid v \in V_1\} \subseteq V(G_2)$ . We write  $G_1 \simeq G_2$  to denote the two graphs are isomorphic.

Now we are ready to define the COLORED SUBGRAPH ISOMORPHISM problem. The COLORED SUBGRAPH ISOMORPHISM problem is formally defined as follows.

COLORED SUBGRAPH ISOMORPHISM (COL-SUB-ISO) **Parameter:**  $|V(H)|$   
**Input:** A host graph  $G$ , a pattern graph  $H$ , and a coloring  $\chi : V(G) \rightarrow \{1, 2, \dots, |V(H)|\}$ .  
**Question:** Is there a subgraph  $G'$  in  $G$  such that  $G' \simeq H$ , and the vertices of  $G'$  have distinct colors?

We obtain the desired algorithm by making  $2^{\mathcal{O}(n)}$  instances of the COL-SUB-ISO problem where the pattern graph has size  $2n + 1$  and treewidth 3, and the given instance of SEOPM is a YES instance if and only if one of the constructed instances is a YES instance of the COL-SUB-ISO problem. Our host graph will be a universal suitor graph corresponding to an instance of SEOPM. We refer the reader to [5] for definitions of treewidth and tree decomposition. To solve COL-SUB-ISO we will use known algorithms, in particular, the algorithm alluded to in the following result.

► **Proposition 4** ([1]). *Let  $G$  and  $H$  denote two graphs on  $n$  and  $q$  vertices, respectively such that the treewidth of  $H$  is at most  $t$ . Furthermore, there is a coloring  $\chi : V(G) \rightarrow [q]$  of  $G$ . Then there is a deterministic algorithm for COL-SUB-ISO that runs in time  $2^q(nt)^{t+\mathcal{O}(1)}$ , and outputs (if there exists one) a subgraph of  $G$  that has a distinct color on every vertex, and is isomorphic to  $H$ .*

To give the desired reduction to COL-SUB-ISO we essentially enumerate all non-isomorphic trees on  $2n + 1$  vertices. In the past, mainly rooted (undirected) trees have been studied, out-branchings not as much. However, every rooted tree can be made an out-branching by orienting every edge away from the root and every out-branching can be transformed into a rooted tree by disregarding all edge orientations. Thus, rooted trees and out-branchings are equivalent, and thus, the results obtained for the former are applicable to the latter. Otter [17] showed that the number of non-isomorphic out-branchings on  $n$  vertices is  $t_n = 2.956^n n^{\mathcal{O}(1)}$ . We can generate all non-isomorphic rooted trees on  $n$  vertices using the algorithm of Beyer and Hedetniemi [2] of runtime  $\mathcal{O}(t_n)$ . We summarize the above in the following result.

► **Proposition 5** ([2, 17]). *The number of non-isomorphic out-branchings on  $n$  vertices is  $t_n = 2.956^n n^{\mathcal{O}(1)}$ . Furthermore, we can enumerate all non-isomorphic rooted trees on  $n$  vertices in time  $\mathcal{O}(t_n)$ .*

## 4.1 Universality of Universal Suitor Graph

In this section we show the “universality” of the universal suitor graph. That is, how given a set of preference lists,  $\mathcal{L}^M$ , of men over women, universal suitor graph encodes all potential suitor graphs. Universal suitor graph for a partial matching encodes all suitor graphs of all potential extensions of the given partial matching. In particular, we show the following result.

► **Lemma 6.** *Let  $\mathcal{L}^M$  denote a set of preference lists of men over women and let  $\mu'$  denote a partial matching on the set  $(M, W)$ . If there exists a perfect matching  $\mu$  such that  $\mu' \subseteq \mu$  (as a set of edges), then  $U(\mathcal{L}^M, \mu)$  is a subgraph of  $U(\mathcal{L}^M, \mu')$ , and is isomorphic to the suitor graph  $G(\mathcal{L}^M, \mu)$ .*

**Proof.** Let  $M'$  and  $W'$  denote the subset of men and women who are matched under  $\mu'$ , respectively. Let  $\mu' \subseteq \mu$ , in terms of a subset of edges. We will refer to the suitor graphs  $G(\mathcal{L}^M, \mu')$  and  $G(\mathcal{L}^M, \mu)$  as simply suitor graphs for  $\mu'$  and  $\mu$ , respectively.

Consider the universal suitor graph for  $\mu$ , denoted by  $U(\mathcal{L}^M, \mu)$ , obtained from  $U(\mathcal{L}^M)$  by pruning w.r.t.  $\mu$ . Since  $\mu' \subseteq \mu$  for every  $w \in W'$  ( $m \in M'$ ) we have  $\mu(w) = \mu'(w)$  ( $\mu(m) = \mu'(m)$ ). Thus, it is easy to see that  $U(\mathcal{L}^M, \mu)$  is a subgraph of  $U(\mathcal{L}^M, \mu')$ , and can be obtained from the latter by applying the pruning operation to every female vertex  $w_i \in W \setminus W'$ . The next claim completes the proof, since it leads to the conclusion that the suitor graph  $G(\mathcal{L}^M, \mu)$  is isomorphic to the universal suitor graph  $U(\mathcal{L}^M, \mu)$ .

► **Claim 7.** *Suitor graph  $G(\mathcal{L}^M, \mu)$  is isomorphic to  $U(\mathcal{L}^M, \mu)$ .*

**Proof.** By the construction of  $G(\mathcal{L}^M, \mu)$ , we know the suitor graph of  $\mu$  has arcs  $(w, \mu(w))$  for every  $w \in W$ . Since  $U(\mathcal{L}^M, \mu)$  is obtained from  $U(\mathcal{L}^M)$  by pruning w.r.t.  $\mu$ , hence we know that  $U(\mathcal{L}^M, \mu)$  contains  $2|\mu|$  vertices

$$\biguplus_{w_i \in W} \{w_i, m_j^i \mid \mu(w_i) = m_j\}.$$

We use  $M^\mu$  to denote the male vertices in  $U(\mathcal{L}^M, \mu)$ .

Let  $\Psi_\mu : M \cup W \rightarrow M^\mu \cup W$  denote a function between the vertex sets of  $G(\mathcal{L}^M, \mu)$  and  $U(\mathcal{L}^M, \mu)$ . For every  $w_i \in W$ , we define  $\Psi_\mu(w_i) = w_i$ , and for every  $m_i \in M$ , we define  $\Psi_\mu(m_i) = m_i^j$ , where  $\mu(m_i) = w_j$ . We will prove that the map  $\Psi_\mu$  is an isomorphism.

We begin with the observation that both graphs are bipartite, with vertex set  $(M, W)$  and  $(M^\mu, W)$ . Thus, to prove that  $\Psi_\mu$  is an isomorphism, it is sufficient to prove that for every  $w \in W, m \in M$ ,  $(w, m)$  is an arc in  $G(\mathcal{L}^M, \mu)$  if and only if  $(w, \Psi_\mu(m))$  is an arc in  $U(\mathcal{L}^M, \mu)$ , and similarly  $(m, w)$  is an arc in  $G(\mathcal{L}^M, \mu)$  if and only if  $(\Psi_\mu(m), w)$  is an arc in  $U(\mathcal{L}^M, \mu)$ .

Let  $(w_i, m_j)$  be an arc in  $G(\mathcal{L}^M, \mu)$ . Thus, we have  $\mu(w_i) = m_j$ , and so  $\Psi_\mu(m_j) = m_j^i$ . The construction of  $U(\mathcal{L}^M, \mu)$  (that is pruning w.r.t.  $\mu$ ) ensures that  $(w_i, m_j^i)$  is an arc in  $U(\mathcal{L}^M, \mu)$ . Conversely, if  $(w_i, m_j^i)$  is an arc in  $U(\mathcal{L}^M, \mu)$  then since  $\mu$  is a perfect matching, by Lemma 2, we can conclude that  $\mu(w_i) = m_j$ , and so  $(w_i, m_j)$  is an arc in  $G(\mathcal{L}^M, \mu)$ . This completes the proof of the if and only if statement about female to male arcs.

Let  $(m_i, w_k)$  be an arc in  $G(\mathcal{L}^M, \mu)$  i.e.,  $\mu(m_i) = w_j$ . Thus,  $w_k >_{m_i} w_j$  ( $m_i$  prefers  $w_k$  to  $w_j$  in  $\mathcal{L}^M$ ). The vertex  $m_i^j = \Psi_\mu(m_i)$  and the arc  $(m_i^j, w_k)$  exists in the universal suitor graph  $U(\mathcal{L}^M)$ . If we can show that  $m_i^j$  exists in  $U(\mathcal{L}^M, \mu)$ , then by the additional condition of Lemma 2, we know that the arc  $(m_i^j, w_k)$  exists in  $U(\mathcal{L}^M, \mu)$ . We note that  $m_i^j$  must survive the pruning of  $U(\mathcal{L}^M)$  w.r.t.  $\mu$  because  $(w_j, m_i)$  is an arc in  $G(\mathcal{L}^M, \mu)$  and so from the earlier part we know that  $(w_j, m_i^j)$  is an arc in  $U(\mathcal{L}^M, \mu)$ . Hence,  $m_i^j$  must be a vertex in  $U(\mathcal{L}^M, \mu)$ , and so we conclude that  $(\Psi_\mu(m_i), w_k)$  is an arc in  $U(\mathcal{L}^M, \mu)$ . Conversely, if  $(m_i^j, w_k)$  is an arc in  $U(\mathcal{L}^M, \mu)$ , then the presence of  $m_i^j$  in the graph allows us to invoke Lemma 2 to conclude that  $\mu(m_i) = w_j$ . This implies that  $w_k >_{m_i} w_j$ , hence  $(m_i, w_k)$  must also be an arc in  $G(\mathcal{L}^M, \mu)$ . This completes the proof of the if and only if statement about male to female arcs. Hence, our proof is complete. ◀

Since  $U(\mathcal{L}^M, \mu)$  is a subgraph of  $U(\mathcal{L}^M, \mu')$ , hence by Claim 7 the latter contains a subgraph that is isomorphic to  $G(\mathcal{L}^M, \mu)$ . This completes the proof. ◀

## 4.2 Rooted Universal Suitor Graph and Valid Subgraphs

For a given universal suitor graph  $U(\mathcal{L}^M, \mu')$  and a subset  $S \subseteq W$ , we define the corresponding **rooted universal suitor graph with sources in  $S$** , as follows. For a vertex  $w \in S$ , if  $w$  is a source in  $U(\mathcal{L}^M, \mu')$  (i.e.  $N^-(w) = \emptyset$ ) then we add the arc  $(r, w)$ . Otherwise, we delete all the male vertices in  $N^-(w)$ , and add the arc  $(r, w)$ . The resulting graph is the rooted universal suitor graph with sources in  $S$ , and is denoted by  $\bar{U}(\mathcal{L}^M, \mu', S)$ . We refer the reader to Figure 2(b) for an example of a rooted universal suitor graph. The set of vertices marked as  $S$  is the set of source vertices that are connected to the root.

Recall that in a universal suitor graph for a partial matching there may be multiple copies of a male vertex, and that brings us to the notion of a valid subgraph. A subgraph of  $U(\mathcal{L}^M, \mu')$  is said to be a **valid subgraph** if it contains every female vertex, and exactly one copy of every male vertex. The definition can be extended to the rooted subgraphs of  $\bar{U}(\mathcal{L}^M, \mu', S)$ , where  $S \subseteq W$ , and a valid rooted subgraph contains the root, every female vertex and exactly one copy of every male vertex.

Consider a rooted tree, such that the root is considered to be in layer 0. A vertex  $v$  is said to be in layer  $i$  in the tree, if the (unique) path from the root to  $v$  contains  $i$  arcs. A rooted tree is called a **matching tree** if every vertex in an odd layer has a unique child in the tree. If a matching tree is a valid subgraph of  $U(\mathcal{L}^M, \mu')$  then it is called a **valid matching tree**. We note that a matching tree is also an out-branching.

Given a matching tree  $T$ , we construct the **triangular matching tree**  $T^\Delta$ , by adding two new vertices  $r^1$  and  $r^2$  to  $T$  and adding the arcs  $(r, r^1)$ ,  $(r^1, r^2)$  and  $(r^2, r)$ . Similarly, for any given rooted universal suitor graph  $\bar{U}(\mathcal{L}^M, \mu', S)$ , we construct the **triangular rooted universal suitor graph**,  $\bar{U}^\Delta(\mathcal{L}^M, \mu', S)$ , by adding two new vertices  $r^1$  and  $r^2$  to  $T$  and adding the arcs  $(r, r^1)$ ,  $(r^1, r^2)$  and  $(r^2, r)$ .

Finally, we define the special coloring  $\chi_{sp}$  used to color the vertices of a triangular rooted universal suitor graph:  $\chi_{sp}$  uses  $2n + 3$  colors, giving distinct colors to  $r, r^1, r^2, w_1, \dots, w_n$ , and using the remaining  $n$  colors such that the subset of copies of the same male vertex gets a distinct color. That is, for each  $i$  ( $1 \leq i \leq n$ ) the subset of  $\{m_i^1, \dots, m_i^n\}$  that exists in the universal suitor graph gets the  $n + 3 + i^{th}$  color.

### 4.3 $2^{\mathcal{O}(n)}$ Algorithm for SEOPM

In this section we combine all the results we have developed so far and design our algorithm.

**Overview of Algorithm 4.1:** Let  $(\mathcal{L}^M, \mu')$  be an input instance of SEOPM. If  $\mu'$  can be extended to  $\mu$ , then (by Lemma 6), we know that  $G(\mathcal{L}^M, \mu)$  is isomorphic to a subgraph in  $U(\mathcal{L}^M, \mu')$ . If  $\mu'$  cannot be extended, then by Proposition 1 we know that for any perfect matching  $\mu \supseteq \mu'$ , the graph  $\bar{G}(\mathcal{L}^M, \mu)$  does not contain an out-branching rooted at  $r$ . In other words, there exists a vertex  $v$  that is not reachable from  $r$  in the graph  $\bar{G}(\mathcal{L}^M, \mu)$ . Consequently, to “solve” SEOPM on  $(\mathcal{L}^M, \mu')$ , it is necessary and sufficient to look for a *valid out-branching or matching tree in the universal suitor graph*  $U(\mathcal{L}^M, \mu')$ . If the algorithm finds one, we can conclude that  $\mu'$  can be extended, else it answers that  $\mu'$  cannot be extended. We implement these ideas by constructing an appropriate instance of COL-SUB-ISO.

The algorithm works as follows. Assume that we have a stable matching  $\mu$  that extends  $\mu'$ . Then consider the graph  $G(\mathcal{L}^M, \mu)$  and let  $S$  denote the subset of female vertices that are sources in the graph. Our algorithm implements this by enumerating all subsets  $S$  of  $W$  in the first loop. Furthermore, by Proposition 1 there is a matching tree,  $T$ , rooted at  $r$  in  $\bar{G}(\mathcal{L}^M, \mu)$ . To “guess” the tree  $T$ , we enumerate all non-isomorphic out-branchings on  $2n + 1$  vertices and first check whether it is a matching tree. If the enumerated tree is a matching tree then we create an instance of COL-SUB-ISO, where the host graph is  $\bar{U}^\Delta(\mathcal{L}^M, \mu', S)$ , with its vertices colored by  $\chi_{sp}$ , and the pattern graph is  $T^\Delta$ . Finally, using an algorithm for COL-SUB-ISO described in Proposition 4, we test whether, or not  $(\bar{U}^\Delta(\mathcal{L}^M, \mu', S), T^\Delta, \chi_{sp})$  is a YES-instance of COL-SUB-ISO. If the algorithm returns  $T^*$ , we can conclude that a stable matching  $\mu$  extends  $\mu'$ . If the outermost for-loop terminates without finding a YES-instance of COL-SUB-ISO, then we return that “no valid out-branching exists” (and hence no stable extension exists). This concludes the description of the algorithm. We refer the reader to Algorithm 4.1 for further details. The next lemma argues the correctness of Algorithm 4.1.

► **Lemma 8.** *Let  $(\mathcal{L}^M, \mu')$  denote an input to SEOPM. Then  $(\mathcal{L}^M, \mu')$  is a YES-instance of SEOPM if and only if Algorithm 4.1 returns a triangular matching tree  $T^*$ .*

**Proof.** Let  $(\mathcal{L}^M, \mu')$  be a YES-instance, i.e., there exists a perfect matching  $\mu$ , such that  $\mu' \subseteq \mu$ , and there exists  $\mathcal{L}^W$  such that  $\mu = \text{GS}(\mathcal{L}^M, \mathcal{L}^W)$ . By Lemma 6,  $G(\mathcal{L}^M, \mu)$  is isomorphic to a subgraph in  $U(\mathcal{L}^M, \mu')$ .

By Proposition 1  $\bar{G}(\mathcal{L}^M, \mu)$  has an out-branching rooted at  $r$ , denoted by  $\tilde{T}$ . Since by Lemma 6  $G(\mathcal{L}^M, \mu)$  is isomorphic to a subgraph in  $U(\mathcal{L}^M, \mu')$ , there exists a valid matching tree  $T'$  that is isomorphic to  $\tilde{T}$  contained in  $\bar{U}(\mathcal{L}^M, \mu', S^*)$ , where  $S^*$  denotes the set of sources in  $G(\mathcal{L}^M, \mu)$ . If we delete the labels on the vertices in  $T'$  (or  $\tilde{T}$ ), we get an out-branching (in fact, a matching tree) on  $2n + 1$  vertices, denoted by  $T$ . Thus,  $T \in \mathcal{F}$ , and we

---

**Algorithm 4.1:** Solves SEOPM.
 

---

**Input:** A set of men and women vertices  $(M, W)$ , preferences of men  $\mathcal{L}^M$ , and a partial matching  $\mu'$   
 Let  $\mathcal{F} \leftarrow \{\text{non-isomorphic out-branchings on } 2n + 1 \text{ vertices}\}$   
**forall**  $S \subseteq W$  **do**  
     **forall** matching tree  $T \in \mathcal{F}$  **do**  
         Using Proposition 4 test whether  $(\bar{U}^\Delta(\mathcal{L}^M, \mu', S), T^\Delta, \chi_{sp})$  is a YES-instance of COL-SUB-ISO.  
         **if** the algorithm returns a subgraph  $T^*$  **then**  
             **return**  $T^*$   
     **return** “No valid out-branching exists”

---

conclude that Algorithm 4.1 will find  $T^*$ , a valid triangular matching tree of  $\bar{U}^\Delta(\mathcal{L}^M, \mu', S^*)$  that is isomorphic to  $T^\Delta$ . Hence, the algorithm will return  $T^*$ .

Suppose that Algorithm 4.1 outputs  $T^*$ . Then there exists a subset  $S \subseteq W$ , and an out-branching on  $2n + 1$  vertices  $T$ , such that  $\bar{U}^\Delta(\mathcal{L}^M, \mu', S)$  contains as subgraph  $T^*$  which is isomorphic to the triangular matching tree  $T^\Delta$ . Observe that  $\bar{U}^\Delta(\mathcal{L}^M, \mu', S)$  has a unique triangle  $r, r^1, r^2$  and thus due to the isomorphism,  $T^*$  contains the triangle  $r, r^1, r^2$ . This implies that every vertex in  $T^*$  is reachable from the root of  $\bar{U}(\mathcal{L}^M, \mu', S)$ . Since male vertices are only reachable from a female vertex, this means that every male vertex has an in-coming female neighbor. Since,  $T^* \setminus \{r^1, r^2\}$  is a valid matching tree of  $\bar{U}(\mathcal{L}^M, \mu', S)$ , there is exactly one copy of every male vertex and every female vertex has a unique out-neighbor. Thus, if  $(w_i, m_k^i)$  is a female to male arc in  $T^*$ , then  $T^*$  does not contain any other out-going arc from  $w_i$ . Thus, the female to male arcs in  $T^*$  denote a perfect matching  $\mu$ . Note that  $\mu' \subseteq \mu$  because  $U(\mathcal{L}^M, \mu')$  contains a unique out-going arc for every matched woman in  $\mu'$ , hence those arcs must also be part of  $T^*$ . Hence, we can conclude that  $T^* \setminus \{r^1, r^2\}$  is an out-branching in the graph  $\bar{G}(\mathcal{L}^M, \mu)$ . By Proposition 1, this means that  $(\mathcal{L}^M, \mu')$  is a YES-instance of SEOPM. This concludes the proof. ◀

The next lemma gives the running time of Algorithm 4.1.

► **Lemma 9.** *Let  $(\mathcal{L}^M, \mu')$  be an input to SEOPM, where  $|M| = n$ . Then, Algorithm 4.1 decides whether  $(\mathcal{L}^M, \mu')$  is a YES-instance to SEOPM in time  $2^{\mathcal{O}(n)}$ .*

**Proof.** The running time of the algorithm is upper bounded by the following formula

$$|\{S \subseteq W\}| \times |\mathcal{F}| \times \text{Time taken by COL-SUB-ISO algorithm}$$

By applying Proposition 5 we upper bound  $|\mathcal{F}|$  by  $2.956^{2n+1}n^{\mathcal{O}(1)}$ . It is a well-known fact that the treewidth of a tree is one, from that it is easy to show that the treewidth of a triangular matching tree is at most 3. (One can first find the tree-decomposition of the tree and then add the two vertices  $r^1, r^2$  to every bag and thus increasing the treewidth by at most two. See [5, Chapter 7] for more details regarding treewidth.) Thus, when we apply Proposition 4, we have a host graph that has at most  $n + n^2 + 3$  vertices, and a pattern graph that has size  $2n + 3$  and treewidth at most 3. Therefore, the running time for using the subroutine for COL-SUB-ISO is  $2^{2n+3}n^{\mathcal{O}(1)}$ . Multiplying all these values together, gives the overall running time to be  $2^n \times 2.956^{2n+1}n^{\mathcal{O}(1)} \times 2^{2n+3}n^{\mathcal{O}(1)} = 2^{\mathcal{O}(n)}$ . ◀

Combining Lemmas 8 and 9 we get the following theorem.

► **Theorem 10.** *There is an algorithm for SEOPM running in time  $2^{\mathcal{O}(n)}$ .*

**Proof.** Given an instance  $(\mathcal{L}^M, \mu')$  to SEOPM, we first apply Algorithm 4.1. If it returns that “No valid out-branching exists” then we return that  $(\mathcal{L}^M, \mu')$  is a NO-instance of SEOPM. Else, if the output is  $T^*$ , we first obtain  $T$  by deleting  $r^1, r^2$  and then using  $T$  we obtain a perfect matching  $\mu' \subseteq \mu$ , by pairing every woman to its unique out-neighbor. Now we invoke **Algorithm Q1** mentioned in [13, Theorem 2] with  $(\mathcal{L}^M, \mu)$  and obtain the desired  $\mathcal{L}^W$ . Correctness and running time follow from Lemmas 8 and 9. This completes the proof. ◀

#### 4.4 A Lower Bound under Exponential Time Hypothesis

In this section we show that Theorem 10 is asymptotically optimal. That is, barring an unlikely scenario occurring in complexity theory, there cannot be a better algorithm for SEOPM. To prove this we will invoke the Exponential Time Hypothesis (ETH), and use the well-known NP-hardness reduction from SAT to SEOPM.

**Exponential Time Hypothesis (ETH):** Let  $\tau$  denote the infimum of the set of constants  $c$  for which there exists an algorithm solving 3-SAT in time  $\mathcal{O}(2^{cn}n^{\mathcal{O}(1)})$ . Then it is conjectured that  $\tau > 0$ .

ETH and its counterpart SETH, introduced by Impagliazzo et al. [11], have been extensively used recently to obtain tight lower bounds for several problems. We use this here to get a lower bound on the running time possible for SEOPM. To this end we will use the following result stated in [5, Theorem 14.4].

► **Theorem 11** ([5]). *Unless ETH fails, there exists a constant  $c > 0$  such that no algorithm for 3-SAT can achieve running time  $\mathcal{O}(2^{c(n+m)}n^{\mathcal{O}(1)})$ . In particular, 3-SAT cannot be solved in time  $2^{o(n+m)}$ . Here,  $n$  and  $m$  denote the number of variables and clauses in the input formula to 3-SAT.*

Using Theorem 11 we show the next result.

► **Theorem 12.** *Unless ETH fails, there is no algorithm for SEOPM running in time  $2^{o(n)}$ .*

**Proof.** Let us assume that we can find an algorithm  $\mathcal{A}$  that solves SEOPM in time  $2^{o(n)}$  where  $n$  is the number of men/ women. In [13], Kobayashi and Matsui showed that SEOPM is NP-complete, by giving a reduction from SAT to SEOPM. In particular, given a SAT instance with  $n$  variables and  $m$  clauses, they reduce it to an instance of SEOPM with  $2m + 3n$  men (and women). An easy observation is that in the reduction given by Kobayashi and Matsui [13], we could have started with 3-SAT and reduced it to an instance of SEOPM with  $2m + 3n$  men (and women). Now we show how to design an algorithm for 3-SAT running in time  $2^{o(n+m)}$  using algorithm  $\mathcal{A}$ . Given an instance  $\phi$  of 3-SAT, we start by applying the polynomial time reduction given in [13] and obtain an instance of SEOPM with  $2m + 3n$  men and  $2m + 3n$  women. Now we solve this instance of SEOPM using algorithm  $\mathcal{A}$  in time  $2^{o(m+n)}$ . Using the solution to an instance of SEOPM we decide in polynomial time whether  $\phi$  is satisfiable or not. Thus, we have given an algorithm for 3-SAT running in time  $2^{o(n+m)}$ , contradicting Theorem 11. This concludes the proof. ◀

## 5 Concluding thoughts

In this paper we designed an exact algorithm for STABLE EXTENSION OF PARTIAL MATCHING running in time  $2^{\mathcal{O}(n)}$ . We complemented this result by showing that unless ETH fails the



running time bound is asymptotically optimal. There are several problems in the stable matching model that are NP-complete and have been studied from the perspective of approximation algorithms. However, there is almost no study about these problems either from the view point of moderately exponential time algorithms or parameterized complexity. The area needs a thorough study in these algorithmic paradigms and is waiting to explode.

---

### References

- 1 Omid Amini, Fedor V. Fomin, and Saket Saurabh. Counting subgraphs via homomorphisms. *SIAM J. Discrete Math.*, 26(2):695–717, 2012.
- 2 Terry Beyer and Sandra Mitchell Hedetniemi. Constant time generation of rooted trees. *SIAM Journal on Computing*, 9(4):706–712, 1980.
- 3 Robert Bredereck, Jiehua Chen, Piotr Faliszewski, Jiong Guo, Rolf Niedermeier, and Gerhard J. Woeginger. Parameterized algorithmics for computational social choice: Nine research challenges. *Tsinghua Science and Technology*, 19(4):358–373, 2014.
- 4 A. Cseh and D. F. Manlove. Stable marriage and roommates problems with restricted edges: complexity and approximability. In *Proceedings of SAGT’15*, volume 9347 of *LNCS*, pages 15–26, 2015.
- 5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 6 Fedor V Fomin and Dieter Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010.
- 7 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Representative sets of product families. In *ESAs*, volume 8737 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 2014.
- 8 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and B. V. Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *J. Comput. Syst. Sci.*, 78(3):698–706, 2012.
- 9 David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.
- 10 D. Gusfield and R. W. Irving. *The Stable Marriage Problem-Structure and Algorithm*. The MIT Press, 1989.
- 11 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
- 12 H. Kobayashi and T. Matsui. Successful manipulation in stable marriage model with complete preference. *IEICE Trans. on Inf. Syst.*, E92-D(2):116–119, 2009.
- 13 H. Kobayashi and T. Matsui. Cheating strategies for the gale-shapley algorithm with complete preference lists. *Algorithmica*, 58:151–169, 2010.
- 14 D. F. Manlove. *Algorithmics of matching under preferences*, volume 2 of *Theoretical Computer Science*. World Scientific, 2013.
- 15 Dániel Marx and Ildikó Schlotter. Parameterized complexity and local search approaches for the stable marriage problem with ties. *Algorithmica*, 58(1):170–187, 2010.
- 16 Dániel Marx and Ildikó Schlotter. Stable assignment with couples: Parameterized complexity and local search. *Discrete Optimization*, 8(1):25–40, 2011.
- 17 Richard Otter. The number of trees. *Annals of Mathematics*, pages 583–599, 1948.
- 18 A. E. Roth and M. Sotomayor. *Two-Sided Matching: A Study in Game Theoretic Modeling and Analysis*. Cambridge Univ. Press, 1990.



# The Adwords Problem with Strict Capacity Constraints

Umang Bhaskar<sup>1</sup>, Ajil Jalal<sup>2</sup>, and Rahul Vaze<sup>3</sup>

- 1 School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai, India  
vaze@tcs.tifr.res.in
- 2 Department of Electrical and Computer Engineering, The University of Texas, Austin, USA  
ajiljalal@utexas.edu
- 3 School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai, India  
vaze@tcs.tifr.res.in

---

## Abstract

We study an online assignment problem where the offline servers have capacities, and the objective is to obtain a maximum-weight assignment of requests that arrive online. The weight of edges incident to any server can be at most the server capacity. Our problem is related to the adwords problem, where the assignment to a server is allowed to exceed its capacity. In many applications, however, server capacities are strict and partially-served requests are of no use, motivating the problem we study.

While no deterministic algorithm can be competitive in general for this problem, we give an algorithm with competitive ratio that depends on the ratio of maximum weight of any edge to the capacity of the server it is incident to. If this ratio is  $1/2$ , our algorithm is tight. Further, we give a randomized algorithm that is 6-competitive in expectation for the general problem. Most previous work on the problem and its variants assumes that the edge weights are much smaller than server capacities. Our guarantee, in contrast, does not require any assumptions about job weights. We also give improved lower bounds for both deterministic and randomized algorithms. For the special case of parallel servers, we show that a load-balancing algorithm is tight and near-optimal.

**1998 ACM Subject Classification** F.1.2 Online Computation

**Keywords and phrases** Online Algorithms, Adwords, Budgeted Matching, Greedy Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.30

## 1 Introduction

Motivated by the problem of assigning advertising slots to advertisers, the adwords problem is a well known and intensely studied online assignment problem. A set of advertisers or bidders has a fixed budget for buying ad slots. A search engine user enters a search query, based on which an advertisement is to be displayed corresponding to an ad slot. Each advertiser places a bid for the slot, and based on these bids, the slot is assigned to a winning bidder and the bid amount is collected as revenue. The objective is to maximize the revenue for the search engine, given the bids of the advertisers and their budgets. Since the search queries – and hence the bids – are not known in advance, this is an online problem, and the winning advertiser at each step must be chosen without knowing future arrivals.



© Umang Bhaskar, Ajil Jalal, and Rahul Vaze;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 30; pp. 30:1–30:14



Leibniz International Proceedings in Informatics

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

More generally, we can replace the advertisers with budgets by servers with capacities, advertising slots by jobs that require processing, and bids by weighted edges. The objective is then to assign jobs to servers to maximize total server utilization, subject to server capacities. In the adwords application, and in the papers that study the adwords problem, the capacities are assumed to be ‘violable’ constraints. That is, a bidder’s winning bids can exceed its budget. In this case, the revenue from a bidder is the minimum of its budget, and the sum of the winning bids placed by it. This is clearly reasonable, since no bidder is charged more than its budget.

However in many online assignment problems, the capacity may in fact be a ‘strict’ constraint. As an example, consider the case when advertisers are servers with capacities that dictate how long the server can be operated, slots are jobs that need to be processed, and bids are processing times. A partially-complete job should not contribute towards work completed, and hence server capacities are strict constraints. As another example, server capacities correspond to download limits on a (web- or image-) hosting site, and requests for items hosted on these sites arrive online. A partially-downloaded item is typically of no use, and hence again should not contribute towards quota used from the site.

The violable capacity constraints are particularly useful when the edge weights are small in comparison to the server capacities, since in this case even if the last job assigned to a server exceeds the server capacity, it can be removed with small loss to the objective. In fact most work on the adwords problem focuses on the problem with this restriction (see, e.g., [3, 4, 13]). For deterministic algorithms with violable capacity constraints, if large edge weights are allowed, there is a simple example that shows that the competitive ratio is at least 2, and this is achieved by a greedy algorithm [12].

We study an online maximum weight assignment problem that generalizes these problems, with strict capacity constraints. Our goal is to maximize total utilization of capacitated servers that are available offline. In each time step, a set of jobs arrives, that have to be assigned instantaneously and irrevocably to the servers to maximize capacity utilization, subject to strict capacity constraints on the servers. Further, the set of edges at each time step must constitute a matching, hence each server at each time step can be assigned at most one job. This thus corresponds more closely to the problem of adwords with multiple slots [3]. We consider both small and large edge weights, and obtain both upper and lower bounds for the problem.

## 1.1 Contributions

We make the following contributions in this paper.

- We propose a simple greedy algorithm that is shown to be 3-competitive, whenever the weight of any job is at most half of the corresponding server capacity. In fact, we prove a more general result that if the weight of any job on a server is at most  $\alpha$  times the corresponding server capacity, the greedy algorithm is  $\left(1 + \frac{1}{1-\alpha}\right)$ -competitive. We show via an example that our analysis of the algorithm is tight. Further, we tighten a lower bound given for online  $b$ -matching [8] and show for large edge weights, our algorithm is nearly tight as well.
- When each server has identical capacity 1 and is *parallel*, that is, a job has the same weight on every server, we give a deterministic  $1 + \epsilon$ -competitive algorithm, where  $\epsilon$  is the maximum job weight. Thus if  $\epsilon \rightarrow 0$ , this algorithm is nearly *optimal*. We also show that no deterministic algorithm obtains a better competitive ratio, even for a single server.
- For the unrestricted edge weights case, we propose a randomized version of the greedy algorithm and show that it is 6-competitive. For our algorithm, we define a job as *heavy*

for a server if its weight is more than half of the server capacity, and *light* otherwise. Our randomization is rather novel, where a server accepts or rejects heavy jobs depending on a coin flip. Typically, the randomization is on the edge side, where an edge is accepted or not depending on the coin flips (e.g., [11]). We also show a lower bound of 2 for any randomized algorithm.

- Lastly, we consider the case when jobs have finite span in addition to their weight, and release the resources consumed at the end of their span. The server capacity is thereafter available for other requests. If all jobs have the same span, then we show that our algorithm is 12-competitive. If they have unequal spans, and the maximum and minimum spans are given to the algorithm, we obtain an  $O(\log \frac{s_{\max}}{s_{\min}})$ -competitive algorithm, where  $s_{\max}$  and  $s_{\min}$  are the maximum and minimum spans respectively.

## 1.2 Related Work

In the adwords problem, a set of bidders with individual budgets is given offline. At each time step, a new request arrives with weighted edges to the bidders. The objective is to assign each request to a bidder, to maximize the total weight of edges selected. If the weight of edges incident to a bidder exceeds its capacity, this capacity is included in the sum, rather than the incident edges. The adwords problem was introduced by Mehta et al. [13], for which the authors give a deterministic algorithm with competitive ratio  $e/(e-1)$  when the ratio of each bid to the bidder's budget is small (and budgets are assumed to be a violable constraint). The problem was further studied in a number of other papers (e.g., [3, 4]) that give different algorithms and analyses, but with the same competitive ratio of  $e/(e-1)$ . In the adwords problem with multiple slots, multiple requests arrive at each time step, and the assignment at each time step must be a matching. This extension is studied by Buchbinder, Jain and Naor [3], and they give an online algorithm based on primal-dual techniques where the competitive ratio is shown to be  $(1-1/c)(1-R_{\max})$ , for  $c = (1+R_{\max})^{1/R_{\max}}$  and  $R_{\max}$  is the ratio of the maximum bid to the minimum budget of any advertiser. As  $R_{\max} \rightarrow 0$ , the competitive ratio tends to  $\frac{e}{e-1}$ , and this is optimal. A special case of the multiple slot setting was earlier studied by Mehta et al. [13] as well.

A variant of the problem where edges have values as well as weights, and the capacities of servers restrict the weight of edges incident, is called the Generalized Assignment Problem. Note that in the adwords problem, the weights and values coincide. This problem is studied by Feldman et al. [5]. With the earlier assumption of small edge weights, and assuming *free disposal*, i.e., earlier assigned items can be discarded later, they give an  $e/(e-1)$ -competitive algorithm. When the job arrivals are stochastic, rather than adversarial, the  $e/(e-1)$  ratio can be improved upon [7]. Without the small bid-to-budget ratio assumption, greedy is known to be 2-competitive, and this is tight [12] for deterministic algorithms.

Our problem is also closely related to the problem of online matching, where each server has capacity 1 and each edge has weight either 1 or 0. Here a randomized  $e/(e-1)$ -competitive algorithm, called RANKING, was given by Karp, Vazirani, and Vazirani [10], and this was shown to be tight for randomized algorithms. The analysis of this algorithm was simplified and extended in later papers [1, 4]. Further, even if multiple online vertices arrive together, the lower bound of  $e/(e-1)$  essentially holds, unless a constant fraction of the vertices arrive together [9]. For the problem of online  $b$ -matching, a deterministic algorithm was given with a competitive ratio of  $\frac{(1+1/b)^b}{(1+1/b)^b-1}$ , which is 2 when  $b = 1/2$ , and tends to  $e/(e-1)$  as  $b$  increases.

The offline version of our problem is a special case of a separable assignment problem (SAP) [6]. An SAP is defined by a set of  $n$  bins and a set of  $m$  items to pack in the bins,

with value  $v_{ij}$  for assigning item  $j$  to bin  $i$ . In addition, there are separable constraints for each bin, describing which subset of items can fit in that bin. The objective is to maximize the total value of items packed in the bins, subject to the bin constraints. The online version of SAP has been studied in [2] with expected competitive ratio  $\frac{1}{1-\frac{1}{\sqrt{k}}}$ , where similar to prior work two restrictions are made; that the weights and sizes of each item are stochastic and each items' size is less than a fraction  $\frac{1}{k}$  of the bin capacity.

## 2 Problem Definition

We are given a set  $I$  of  $n$  servers, where server  $i$  has capacity  $C_i$ . We consider an *online* scenario, in which at each time step  $t \in \{1, \dots, T\}$ , a set of jobs  $J(t)$  and a set of edges  $E(t)$  from servers  $I$  to jobs  $J(t)$  is revealed<sup>1</sup>. Edges are weighted, and  $w(e)$  for  $e = (i, j)$  is the weight of job  $j$  on server  $i$ . In particular, if job  $j$  is assigned to server  $i$ , it consumes  $w(e)$  resources of server  $i$  out of the possible  $C_i$ . In general, a job may have different weights on different servers, thus for distinct servers  $i$  and  $i'$ ,  $w(i, j) \neq w(i', j)$ . The entire set of jobs is  $J = \cup_{t \leq T} J(t)$ , and  $E = \cup_{t \leq T} E(t)$ . For a set of edges  $F$ , define  $W(F) := \sum_{e \in F} w(e)$ . Define  $G(t)$  as the bipartite graph  $(I \cup J(t), E(t))$ . A set of edges  $F$  is *feasible* if (i)  $F(t)$  is a matching for all  $t \leq T$ , i.e., each server and each job is connected to at most one job and one server respectively at each  $t$ , and (ii) for each server  $i$ , the weight of edges in  $F$  incident to  $i$  is at most  $C_i$  (this is the strict capacity constraint). We will also call a feasible set of edges an *allocation*. Our objective is to maximize the weight of the allocation obtained.

An optimal allocation has maximum weight among all allocations. The *competitive ratio* for an algorithm is defined as the maximum over all instances of the ratio of the weight of the optimal allocation, to that obtained by the algorithm. For a *randomized* algorithm, the competitive ratio is obtained by taking the denominator of the previous ratio as the *expected* weight of the allocation obtained by the algorithm. Note that the competitive ratio is always at least 1.

In Section 5, we consider the case where jobs have finite span. Here each edge  $e = (i, j)$  has a tuple  $(w, s)$  associated with it, where  $w$  is the weight and  $s$  is the time steps for which job  $j$  is active, if assigned to machine  $i$ . If job  $j$  arrives at time  $t$  and is assigned to server  $i$ , it consumes  $w$  resources from server  $i$  in time steps  $\tau \in [t, t + s - 1]$ . We then say that  $j$  is active on server  $i$  in this period. Thus a set of edges  $F$  is feasible if, for each  $t$ , (i)  $F(t)$  is a matching, and (ii) the total weight of jobs active on each server  $i$  at time  $t$  is at most its capacity. Our objective is now to obtain an allocation to maximize the total weight of active jobs, summed over servers as well as time steps.

Due to space constraints, all missing proofs appear in the full version.

## 3 Deterministic Algorithms

We begin by giving a simple example that shows that no deterministic algorithm can be competitive for our problem.

► **Example 1.** In Fig. 1, there is a single server with capacity 1. At  $t = 1$ , a job of weight  $\epsilon \ll 1$  arrives. If the algorithm does not accept the job, the input ends; in this case, the optimal value is  $\epsilon$  while the the algorithm obtains value zero. If the algorithm accepts the

<sup>1</sup> Note that while our algorithms are designed for the setting where multiple jobs arrive at a single time-step, all our lower bounds hold for the case where a single job arrives at each time step.



■ **Figure 1** Illustration for Example 1.

---

**Algorithm 1:** GREEDY( $G, S$ )
 

---

**Input** : Weighted bipartite graph  $G$ , set of active servers  $S$   
**Output** : Matching  $M$   
**begin**  
 $M \leftarrow \emptyset$   
**for**  $e = (i, j) \in G$  *in descending order of weight* **do**  
  **if**  $(M \cup e$  *is a matching*) **AND**  $(i \in S)$  **then**  $M \leftarrow M \cup e$  ;  
**return**  $M$

---

job, the second job with weight 1 arrives. Since the capacity is 1, the algorithm cannot accept this job. In this case, the optimal value is 1 while the algorithm obtains  $\epsilon$ , and hence any deterministic algorithm has competitive ratio at least  $1/\epsilon$ .

If we restrict the maximum weight of a job to be  $\frac{1}{2}$ , then every server can accept at least two jobs, and a deterministic algorithm can give a non-trivial competitive ratio even on adversarial sequences. Under this restriction, we propose an ONLINEGREEDY algorithm that is shown to be 3-competitive next.

In the discussion of the following algorithms, we use  $M(t)$  to denote the set of edges selected by the algorithm in time step  $t$ ,  $A(t) := \cup_{\tau \leq t} M(\tau)$ , and  $M_i(t)$  and  $A_i(t)$  to denote the set of edges in  $M(t)$  and  $A(t)$  incident to server  $i$ .

### 3.1 Deterministic Algorithm for Restricted Edge Weights

We begin with the notion of active servers.

► **Definition 2.** Active server: The server  $i$  is active at time step  $t + 1$  if the sum of the weights of edges assigned to it so far is at most half its capacity, i.e.,  $W(A_i(t)) \leq \frac{1}{2}C_i$ . We will use  $S$  to denote the set of active servers.

The deterministic algorithm GREEDY takes as inputs a weighted bipartite graph  $G$ , as well as a set  $S$  of active servers. GREEDY greedily picks maximum weight edges from the bipartite graph  $G$  that are incident to active servers to form a matching  $M$ .

#### 3.1.1 OnlineGreedy

We now present a deterministic algorithm ONLINEGREEDY that is 3-competitive for the restricted weights case, where the weight of each edge incident to a server is at most half the server capacity, i.e.,  $w(i, j) \leq \frac{1}{2}C_i$  for each server  $i$  and job  $j$ .

ONLINEGREEDY maintains a set of active servers  $S$ , along with sets  $A_i(t)$  for each server  $i$ , where  $A_i(t)$  is the set of edges selected that are incident to server  $i$  until time  $t$ . At each time step  $t$ , ONLINEGREEDY calls GREEDY and passes to it as input the weighted bipartite graph  $G(t)$  along with the current set of active servers  $S$ . For each edge  $(i, j) \in M(t)$ , where  $M(t)$  is the matching returned by GREEDY, edge  $(i, j)$  is added to the allocation  $A_i(t)$ . ONLINEGREEDY then checks if  $W(A_i(t)) > \frac{1}{2}C_i$ , in which case server  $i$  is no longer active and is removed from the set of active servers  $S$  for next time slot. If a server  $i$  is active at

**Algorithm 2:** ONLINEGREEDY

---

**Input** : Server capacities  $C_1, C_2, \dots, C_n$   
 Weighted bipartite graphs  $G(t)$  for  $t \leq T$ , such that  $w(i, j) \leq \frac{1}{2}C_i \forall i, j$

**Output** : Feasible allocation  $A(T) = \cup_{t \leq T} M(t)$

**begin**  
 $S \leftarrow I$   
 $A_i(0) \leftarrow \emptyset \forall i \in I$   
**for**  $t \leftarrow 1$  **to**  $T$  **do**  
 $M(t) \leftarrow \text{GREEDY}(G(t), S)$ ,  $A(t) \leftarrow A(t-1) \cup M(t)$   
**for**  $(i, j) \in M(t)$  **do**  
**if**  $W(A_i(t)) > \frac{C_i}{2}$  **then**  $S \leftarrow S \setminus \{i\}$  ;

---

time  $t$ , i.e.,  $W(A_i(t-1)) \leq \frac{1}{2}C_i$ , and an edge  $e$  is added to  $A_i(t-1)$ , then  $W(A_i(t-1))$  increases by at most  $\frac{1}{2}C_i$ , and hence  $W(A_i(t)) \leq C_i$ . Hence, assigning a job to an active server always results in a feasible allocation. Also, since GREEDY performs a matching at each time step, the degree constraints (one job/server is assigned to at most one server/job, respectively) are always satisfied. The algorithm continues either until  $S = \emptyset$  or  $t = T$ .

► **Remark.** We note that the restriction on edge weights is only used in proving the feasibility of the allocation obtained, and not in the proof of 3-competitiveness below. In particular, if the edge weights are unrestricted, the allocation obtained may violate the capacity constraints, but will be 3-competitive.

► **Theorem 3.** *ONLINEGREEDY is 3-competitive.*

**Proof.** For each time step  $t$ , let  $M(t)$  denote the matching produced by ONLINEGREEDY, and let  $M^*(t)$  denote the corresponding matching given by the optimal offline algorithm. Let  $A^*(t) = \cup_{\tau \leq t} M^*(\tau)$ , and  $A_i^*(t)$  is the set of edges to server  $i$  in the optimal allocation until time  $t$ . Also,  $A_i^* = A_i^*(T)$ ,  $A_i = A_i(T)$ , and  $A = \cup_{i \in I} A_i$ ,  $A^* = \cup_{i \in I} A_i^*$ .

We say that an edge  $e = (i, j) \in M^*(t) \setminus M(t)$ , has been *blocked* by a heavier weight edge  $f \in M(t)$  if  $w(f) \geq w(e)$  and  $f$  shares a server vertex ( $i$ ) or job vertex ( $j$ ) with  $e$ . As  $f$  has more weight than  $e$ , GREEDY would select it first in  $M(t)$ , and hence  $e$  cannot be selected without violating matching constraints. For each edge  $(i, j) \in M^*(t) \setminus M(t)$ , there are three possible reasons why the edge was not selected by ONLINEGREEDY:

1. An edge  $f = (i, j') \in M(t)$ ,  $j' \neq j$  *blocks*  $(i, j)$ , i.e. server  $i$  was matched to some job  $j'$  by GREEDY, such that  $w(i, j') \geq w(i, j)$ .
2. An edge  $f = (i', j) \in M(t)$ ,  $i' \neq i$  *blocks*  $(i, j)$ , i.e. job  $j$  was matched to some server  $i'$  by GREEDY, such that  $w(i', j) \geq w(i, j)$ .
3. The server  $i$  was inactive at time step  $t$ , i.e.,  $i \notin S$ .

Let  $E_1(t)$ ,  $E_2(t)$  and  $E_3(t)$  denote the set of edges in  $M^*(t) \setminus M(t)$  that satisfy the first, second and third condition respectively. Clearly,  $E_1(t) \cup E_2(t) \cup E_3(t) = M^*(t) \setminus M(t)$ . *Note:* No edge can satisfy the first and third condition simultaneously, as a server which is inactive at time  $t$  cannot be matched to any job at time  $t$ . Therefore,  $E_1(t) \cap E_3(t) = \emptyset$ . However, in general,  $E_1(t) \cap E_2(t) \neq \emptyset$  and  $E_2(t) \cap E_3(t) \neq \emptyset$ , as edges can satisfy conditions 1 and 2 or 2 and 3.

Let  $S$  be the set of active servers at time  $T+1$ . For all servers  $i, i \notin S$ , since  $W(A_i^*) \leq C_i$  and  $W(A_i) > \frac{1}{2}C_i$ , the allocation  $A_i$  is a  $\frac{1}{2}$  approximation to  $A_i^*$ , i.e.,

$$\sum_{i: i \notin S} \sum_{e \in A_i^*} w(e) < 2 \sum_{i: i \notin S} \sum_{e \in A_i} w(e). \quad (1)$$

Let  $E_1 = \cup_{t=1}^T E_1(t)$ ,  $E_2 = \cup_{t=1}^T E_2(t)$ ,  $E_3 = \cup_{t=1}^T E_3(t)$ . Define  $E_1^S = \{e = (i, j) \in E_1 \mid i \in S\}$ ,  $E_2^S = \{e = (i, j) \in E_2 \mid i \in S\}$ . Clearly,  $E_1^S \cup E_2^S = \cup_{i:i \in S} (A_i^* \setminus A_i)$ , as no edge  $e = (i, j)$ ,  $i \in S$  can satisfy the third condition.

The edges  $e \in E_1^S \cup E_2^S$  were not selected in the greedy allocation as they were blocked by edges of heavier weight from  $A \setminus A^*$ . The edges in the set  $A \setminus A^*$  are of two types:

1.  $f = (i, j) \in A_i \setminus A_i^*$ ,  $i \in S$ . As all edges  $e = (i', j') \in E_1^S \cup E_2^S$  are such that  $i' \in S$ ,  $e$  was blocked either because  $e$  and  $f$  share a server vertex ( $i = i'$ ) or they share a job vertex ( $j = j'$ ). Thus, for every edge  $f = (i, j) \in A_i \setminus A_i^*$ ,  $i \in S$ , there may exist at most two edges  $e_1 = (i, j')$ ,  $e_2 = (i', j)$  that are blocked by  $f$ , so that  $e_1, e_2 \in E_1^S \cup E_2^S$  and  $w(f) \geq w(e_1)$ ,  $w(f) \geq w(e_2)$ .
2.  $g = (i, j) \in A_i \setminus A_i^*$ ,  $i \notin S$ . As all edges  $e = (i', j') \in E_1^S \cup E_2^S$  are such that  $i' \in S$ ,  $e$  was blocked only because  $g$  and  $e$  share the same job vertex ( $j = j'$ ) and  $g$  was greedily picked first. Thus, for every edge  $g = (i, j) \in A \setminus A^*$ ,  $i \notin S$ , there may exist at most one edge  $e_1 = (i', j) \in E_1^S \cup E_2^S$  that is blocked by  $g$  and is such that  $w(g) \geq w(e_1)$ .

As  $f = (i, j) \in A_i \setminus A_i^*$ ,  $i \in S$  can block at most two edges in  $E_1^S \cup E_2^S$  and  $g = (i, j) \in A_i \setminus A_i^*$ ,  $i \notin S$  can block at most one edge in  $E_1^S \cup E_2^S$ ,

$$\sum_{i:i \notin S} \sum_{e \in A_i^* \setminus A_i} w(e) = \sum_{e \in E_1^S \cup E_2^S} w(e) \leq 2 \sum_{i:i \in S} \sum_{f \in A_i \setminus A_i^*} w(f) + \sum_{i:i \notin S} \sum_{g \in A_i \setminus A_i^*} w(g). \quad (2)$$

Adding (1), (2),

$$\sum_{i:i \notin S} \sum_{e \in A_i^*} w(e) + \sum_{i:i \in S} \sum_{e \in A_i^* \setminus A_i} w(e) \leq 2 \sum_{i:i \notin S} \sum_{e \in A_i} w(e) + 2 \sum_{i:i \in S} \sum_{f \in A_i \setminus A_i^*} w(f) + \sum_{i:i \notin S} \sum_{g \in A_i \setminus A_i^*} w(g).$$

Adding  $\sum_{i:i \in S} \sum_{e \in A_i \cap A_i^*} w(e)$  to LHS and RHS,

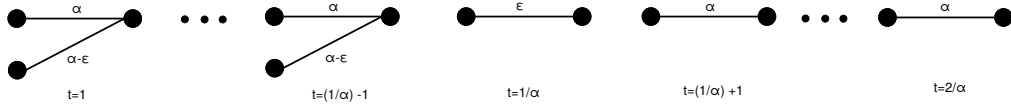
$$\sum_{i:i \notin S} \sum_{e \in A_i^*} w(e) + \sum_{i:i \in S} \sum_{e \in A_i^*} w(e) \leq \sum_{i:i \in S} \sum_{e \in A_i \cap A_i^*} w(e) + 2 \sum_{i:i \in S} \sum_{f \in A_i \setminus A_i^*} w(f) + 3 \sum_{i:i \notin S} \sum_{g \in A_i} w(g).$$

Simplifying, we get  $\sum_{i \in I} \sum_{e \in A_i^*} w(e) \leq 3 \sum_{i \in I} \sum_{e \in A_i} w(e)$ , as required.  $\blacktriangleleft$

► **Remark.** In the more general case, where edge weights are restricted to be at most  $\alpha$  ( $\leq 1$ ) times the corresponding server capacities, i.e., if  $w(i, j) \leq \alpha C_i \forall i, j$ , the following modification of ONLINEGREEDY makes it  $\left(1 + \frac{1}{1-\alpha}\right)$ -competitive. Instead of removing a server  $i$  from the set of active servers  $S$  when  $W(A_i(t)) > \frac{1}{2}C_i$ , if we remove it when  $W(A_i(t)) > (1-\alpha)C_i$ , then (1) can be changed to  $\sum_{i:i \notin S} \sum_{e \in A_i^*} w(e) < \left(\frac{1}{1-\alpha}\right) \sum_{i:i \notin S} \sum_{e \in A_i} w(e)$ . The rest of the proof follows directly to give a  $\left(1 + \frac{1}{1-\alpha}\right)$ -competitive algorithm. Clearly, as  $\alpha \rightarrow 1$ , the competitive ratio tends to 0, and ONLINEGREEDY will fail, as expected from Example 1. To handle the case of unrestricted job weights, in the next subsection, we present a randomized algorithm RANDOMONLINEGREEDY which is 6-competitive.

► **Example 4.** This example is used to show the tightness of analysis for **Theorem 3**. There are 2 servers with capacity 1. We assume for simplicity that  $1/\alpha$  is integral, but the example can be modified to remove this constraint. The sequence of jobs is illustrated in Fig. 2. There





■ Figure 2 Illustration for Example 4.

are  $(1/\alpha) - 1$  jobs that have weight  $\alpha$  to the first server, and weight  $\alpha - \epsilon$  to the second server. `ONLINEGREEDY` assigns all of these to the first server, as well as the next job, and thus the first server now has remaining capacity  $\alpha - \epsilon$ . The online algorithm then cannot assign any of the remaining jobs, and obtains weight  $1 - \alpha + \epsilon$ . The optimal offline assigns the first  $(1/\alpha) - 1$  jobs to the second machine, ignores the job of weight of  $\epsilon$ , and assigns the remaining  $1/\alpha$  jobs to the first machine, obtaining a total weight of  $1 + (\alpha - \epsilon) \left(\frac{1}{\alpha} - 1\right)$ . As  $\epsilon$  tends to zero, this gives a lower bound of  $1 + \frac{1}{1-\alpha}$ .

### 3.2 A lower bound for deterministic algorithms

The lower bound example by Kalyanasundaram and Pruhs [8] for online  $b$ -matching holds for our problem as well, and shows that if the maximum ratio of edge-weight to server capacity is  $\alpha$ , then no deterministic algorithm can obtain competitive ratio better than  $(1 + 1/b)^b / (1 + 1/b)^b - 1$ , where  $b := \lceil 1/\alpha \rceil$ . As  $\alpha$  goes to zero, this ratio tends to  $e/e - 1$ .

We can use the strict capacity constraint and strengthen this lower bound slightly, to obtain a lower bound of  $\frac{(1+1/b)^{b-1}}{(1+1/b)^{b-1}-1}$ . For  $\alpha = 1/2$  and  $1/3$ , this evaluates to 3 and 2.28 respectively, while our algorithm is 3- and 2.5-competitive respectively in these cases. Thus for  $\alpha = 1/2$ , the competitive ratio we obtain is tight. Since the construction and proof are similar to the earlier example in [8], we give a sketch of the proof here.

► **Theorem 5.** *No deterministic algorithm obtains competitive ratio better than  $\frac{(1+1/b)^{b-1}}{(1+1/b)^{b-1}-1}$  for the online max-weight assignment problem with strict capacity constraints.*

**Proof Sketch.** Our example closely follows the lower bound for  $b$ -matching [8], deviating only at the very beginning. Informally, the earlier example starts with  $(b + 1)^b$  servers of capacity 1, and jobs of weight  $1/b$  to a subset of machines. The example sends jobs in  $b + 1$  groups, and ensures that jobs in group  $i$  can only be assigned by the online algorithm to servers in  $S_i$ , where  $S_1$  is the set of all  $(b + 1)^b$  servers. Group  $R_i$ ,  $i \leq b$ , consists of  $b^i (1 + 1/b)^{b-i}$  jobs, and group  $R_{b+1}$  consists of  $b^{b+1}$  jobs. Further, it ensures that the last group  $R_{b+1}$  of jobs cannot be assigned to any server by the algorithm, while the offline optimal assigns all jobs. This gives the earlier lower bound of  $(1 + 1/b)^b / (1 + 1/b)^b - 1$ . We modify the example by ensuring that the last *two* groups cannot be assigned by any online algorithm, giving us the improved lower bound.

If  $b = 1$ , then the lower bound from Example 1 can be used for the theorem. Otherwise,  $b \geq 2$ . To ensure that jobs from the penultimate group  $b$  cannot be assigned by the online algorithm, we will start off with  $(b + 1)^{b+1}$  servers, and send  $(b + 1)^{b+1}$  jobs of weight  $\epsilon$ , each of which can only be assigned to a distinct machine. After these jobs are sent, if at most  $(b + 1)^b$  jobs are assigned to their unique machine, then we stop. In this case, the optimal offline algorithm assigns all the jobs, and the online algorithm has competitive ratio  $(b + 1)$ , which is at least the bound in the theorem for  $b \geq 2$ . If at least  $(b + 1)^b$  machines have a job assigned to them, we select  $(b + 1)^b$  of these machines, let these machines be  $S_1$ , and run the earlier lower bound example. In the example, the jobs in group  $R_b$  only have edges



**Algorithm 3:** PARALLELOADBALANCE

---

**Input** : Capacities  $C$  of servers  
Jobs  $J(t)$  at each time step  $t \in \{1, \dots, T\}$ , with weight  $w(j)$  for  $j \in J(t)$ .

**Output**: Feasible server allocations  $A_i, i \in \{1, 2, \dots, n\}$

**begin**  
 $A_i \leftarrow \emptyset \forall i \in \{1, \dots, n\}$  initially.  
**for**  $t \leftarrow 1$  **to**  $T$  **do**  
**for**  $j \in J(t)$ , in decreasing order of weight **do**  
Let  $i$  be the machine with highest remaining capacity  $C - W(A_i)$  that is  
not assigned a job in current time step.  
**if**  $W(A_i \cup \{j\}) \leq C$  **then**  $A_i \leftarrow A_i \cup \{j\}$  ;  
**else return**;

---

to servers in  $S_b$ , and have weight  $1/b$  to these servers. However, with our initial step, when the jobs in group  $R_b$  arrive, each server in  $S_b$  has remaining capacity at most  $(1/b) - \epsilon$ , and hence cannot serve any more jobs. Hence, jobs in both groups  $R_b$  and  $R_{b+1}$  cannot now be scheduled by any online algorithm, while the optimal offline algorithm ignores the initial jobs of weight  $\epsilon$ , and successfully assigns all the remaining jobs. Thus, the optimal offline algorithm obtains total weight  $(b+1)^b$ , while any online algorithm obtains total weight at most  $(b+1)^b - b^{b-1} - b^b$ . ◀

### 3.3 Parallel Servers

Servers are *parallel* if  $C_i = C_{i'}$  and  $w(i, j) = w(i', j)$  for all jobs  $j$  and all servers  $i, i'$ . That is, the servers are identical, and each job consumes the same quantity of resources on each server. Thus instead of edge weights we now refer to the weight of each job. If servers are parallel, each with capacity  $C$ , and each job has weight at most  $\epsilon$ , then we show a simple deterministic load-balancing algorithm that is  $\frac{1}{1 - \epsilon/C}$ -competitive.

► **Lemma 6.** *After any time step  $t$ , the remaining capacity of any pair of machines  $i, i'$  differs by at most  $\epsilon$  with the PARALLELOADBALANCE algorithm.*

**Proof.** The proof is by induction. Suppose the lemma is true at the end of time step  $t-1$ , and  $A_i(t-1), A_{i'}(t-1)$  are the set of jobs assigned by the algorithm to machines  $i, i'$  until time step  $t-1$ . Assume without loss of generality that  $W(A_i(t-1)) \leq W(A_{i'}(t-1))$ . Then by the inductive hypothesis,  $W(A_i(t-1)) \geq W(A_{i'}(t-1)) - \epsilon$ . Further if  $j, j'$  are the jobs assigned to  $i, i'$  respectively in time step  $t$ , then by the algorithm  $\epsilon \geq w(j) \geq w(j')$ . It follows that  $|W(A_i(t)) - W(A_{i'}(t))| \leq \epsilon$ . ◀

► **Theorem 7.** *Algorithm PARALLELOADBALANCE is  $\frac{1}{(1 - \epsilon/C)}$ -competitive. Further, no deterministic algorithm can perform better.*

**Proof.** If the **else** condition in PARALLELOADBALANCE is never encountered, then at every time step the  $n$  jobs of largest weight are assigned, and hence the assignment obtained is optimal. Suppose that for some time step  $t$ , job  $j$ , and server  $i$ , the **else** condition is encountered. Then  $W(A_i(t-1)) + w(j) > C$ , and in fact every server has remaining capacity at most  $\epsilon$ . This is obviously true of server  $i$ , since  $w(j) \leq \epsilon$ . To see this for the other servers, consider any server  $i'$  with  $W(A_{i'}(t-1)) < W(A_i(t-1))$ . Then in

**Algorithm 4:** RANDOMONLINEGREEDY

---

**Input** : Server capacities  $C_1, C_2, \dots, C_n$   
 Weighted bipartite graph  $G(t)$  for  $t \leq T$ , such that  $w(i, j) \leq C_i \forall i, j$

**Output** : Random feasible allocation  $A = \cup_{i \in I} A_i$

**begin**  
 $S \leftarrow I$   
 $S_1, S_2, A_i(0), B_i(0) \leftarrow \emptyset \forall i \in I$   
**for**  $k \leftarrow 1$  **to**  $n$  **do**  
 $v_k \sim \text{Bernoulli}(\frac{1}{2})$   
**if**  $v_k = 1$  **then**  $S_1 \leftarrow S_1 \cup \{k\}$  // accept only heavy jobs ;  
**else**  $S_2 \leftarrow S_2 \cup \{k\}$  // accept only light jobs ;  
**for**  $t \leftarrow 1$  **to**  $T$  **do**  
 $M(t) \leftarrow \text{GREEDY}(G(t), S)$   
**for**  $e = (i, j) \in M(t)$  **do**  
 $B_i(t) \leftarrow B_i(t-1) \cup \{e\}$   
**if**  $W(B_i(t)) > \frac{C_i}{2}$  **then**  $S \leftarrow S \setminus \{i\}$   
 ;  
**if**  $(i \in S_1 \text{ AND } w(i, j) > \frac{C_i}{2}) \text{ OR } (i \in S_2 \text{ AND } w(i, j) \leq \frac{C_i}{2})$  **then**  
 $A_i(t) \leftarrow A_i(t-1) \cup \{e\}$

---

time step  $t$ , server  $i'$  is assigned a job  $j'$  with weight at least  $w(j)$ . Further, by Lemma 6,  $W(A_{i'}(t-1)) \geq W(A_i(t-1)) - \epsilon$ , and hence

$$W(A_{i'}(t)) = W(A_{i'}(t-1)) + w(j') \geq W(A_{i'}(t-1)) + w(j) \geq W(A_i(t-1)) + w(j) - \epsilon > C - \epsilon.$$

Thus in this case, on any machine the remaining capacity is at most  $\epsilon$ . The proof of the upper bound immediately follows.

For the lower bound, consider a single server with capacity 1. The adversary behaves as follows. At any time, if the total weight of requests sent is at least 1, the adversary stops. The adversary first sends requests of size at most  $\epsilon$  to the online algorithm, until the server has exactly  $\epsilon$  remaining capacity. The adversary now sends requests of size  $\delta \ll \epsilon$  until the online algorithm assigns exactly one. The remaining capacity on the server is then  $\epsilon - \delta$ . It then sends a single request of size  $\epsilon$ . The optimal offline algorithm ignores the request of size  $\delta$  and obtains weight 1, while the online algorithm has weight at most  $1 - \epsilon + \delta$ , giving the lower bound for small  $\delta$ .  $\blacktriangleleft$

#### 4 A Randomized Algorithm for Unrestricted Edge Weights

Now we present a randomized algorithm, called RANDOMONLINEGREEDY, that is 6-competitive for the general case of unrestricted edge weights.

Note that while  $w(i, j)$  can be unbounded, any edge such that  $w(i, j) > C_i$  will be ignored as it can never be allocated to server  $i$ .

► **Definition 8.** An edge  $e = (i, j)$  that satisfies  $\frac{C_i}{2} < w(i, j) \leq C_i$  is called a *heavy edge* and the corresponding job is called a *heavy job* for that server. In other words, the weight of a heavy edge  $(i, j)$  connected to a server  $i$  is at least half the server's initial capacity. An edge that is not heavy is called *light*, and the corresponding job is called *light* for that server.

At the start of the algorithm RANDOMONLINEGREEDY, an unbiased coin is flipped for each server  $i$ . If heads, then server  $i$  is added to set  $S_1$ , else it is added to set  $S_2$ . If server

$i \in S_1$ , it can only accept jobs corresponding to heavy edges, while if  $i \in S_2$ , it can only accept jobs corresponding to light edges.

Similar to **ONLINEGREEDY**, **RANDOMONLINEGREEDY** maintains a set of active servers  $S$ , along with sets  $A(t)$  and  $B(t)$ . At each time step  $t$ , the weighted bipartite graph  $G_t$  and set of active servers  $S$  are passed as input to **GREEDY**, which returns a matching  $M(t)$ . The set  $B(t) := \cup_{\tau \leq t} M(\tau)$  and  $B_i(t)$  represents the set of edges in  $B(t)$  connected to server  $i$ . The set  $A_i(t)$  is conditioned on the coin toss for server  $i$ . If  $i \in S_1$ ,  $A_i(t)$  only contains the heavy edges in  $B_i(t)$ . Otherwise, if  $i \in S_2$ ,  $A_i(t)$  only contains the light edges in  $B_i(t)$ .

At time  $t$ , if **RANDOMONLINEGREEDY** adds an edge  $e = (i, j)$  to  $B$ , the algorithm checks the weight  $W(B_i(t))$  to see if it should be active for the next time step. If  $W(B_i(t)) > \frac{1}{2}C_i$ , then server  $i$  is removed from  $S$ . The reason for maintaining two sets  $B$  and  $A$  is that it is possible for  $B_i(T)$  to be infeasible for some server  $i$ . However,  $A_i(T)$  is a feasible allocation  $\forall i$ , and  $\mathbb{E}[W(A_i(T))] = \frac{1}{2}W(B_i(T))$ . The algorithm continues until either  $S = \emptyset$  or  $t = T$ .

► **Lemma 9.** *The allocation  $A_i(T)$  is feasible for each machine  $i \in I$ .*

► **Example 10.** This example illustrates how  $B_i(T)$  may be an infeasible allocation, while  $A_i(T)$  is feasible. Consider a single server with capacity  $C$ . At each time step, one job is presented, and  $T = 2$ . At  $t = 1$ , a job of weight  $\frac{C}{2} - \epsilon$  is presented, while at time  $t = 2$ , a job of weight  $C$  is presented. **RANDOMONLINEGREEDY** will put both jobs into  $B(2)$ . If the coin showed heads,  $A(2)$  will contain the second edge. If the coin showed tails,  $A(2)$  will contain the first edge at time  $t = 1$ , i.e.,  $A(2) = \{\frac{1}{2}C - \epsilon\}$  or  $A(2) = \{C\}$ , and both allocations occur with probability  $\frac{1}{2}$ . However,  $W(B(2)) = (\frac{3}{2}C - \epsilon)$ , which is an infeasible allocation.

► **Lemma 11.**  $\frac{W(A^*(T))}{W(B(T))} \leq 3$ .

**Proof.** As the arguments for (1), (2) hold for the sets  $B_i(t) \forall i$ , the proof for **Lemma 11** follows similar to the proof for **Theorem 3**. ◀

► **Lemma 12.**  $\frac{W(B(T))}{\mathbb{E}[W(A(T))]} = 2$ .

**Proof.** The set  $B_i(t)$  can be partitioned into two mutually exclusive subsets  $X_i(t)$  and  $Y_i(t)$ , such that  $X_i(t)$  only contains heavy edges, while  $Y_i(t)$  only contains light edges. Note that  $|X_i(t)| \leq 1$ . Let  $v_i = 1 (= 0)$  if server  $i$  accepts only heavy (light) jobs. As  $A_i(t)$  is a feasible allocation  $\forall t$  and  $A_i(t) = X_i(t), t \leq T$  if  $v_i = 1$ , and  $A_i(t) = Y_i(t), t \leq T$  if  $v_i = 0$ ,  $X_i(t), Y_i(t), t \leq T$  are both feasible allocations. Therefore  $B_i(t) = X_i(t) \cup Y_i(t)$ ,  $X_i(t) \cap Y_i(t) = \emptyset \forall t$ , and  $W(B_i(t)) = W(X_i(t)) + W(Y_i(t))$ . Hence

$$\begin{aligned} \mathbb{E}[W(A_i(T))] &= \mathbb{P}[v_i = 1] W(A_i(T) \mid v_i = 1) + \mathbb{P}[v_i = 0] W(A_i(T) \mid v_i = 0), \\ &= \frac{1}{2} (W(X_i(T)) + W(Y_i(T))). \end{aligned}$$

Therefore,  $\mathbb{E}[W(A_i(T))] = \frac{1}{2}W(B_i(T))$ . Summing over all servers  $i$ ,  $\sum_{i=1}^n \mathbb{E}[W(A_i(T))] = \frac{1}{2} \sum_{i=1}^n W(B_i(T))$ , and  $\frac{\mathbb{E}[W(A(T))]}{W(B(T))} = \frac{1}{2}$ . ◀

► **Theorem 13.** ***RANDOMONLINEGREEDY** is 6-competitive.*

**Proof.** Let  $W(A^*(T)) = W(\cup_{i=1}^n A_i^*(T))$  be the value of the allocation made by the optimal offline algorithm, and  $W(B(T)) = W(\cup_{i=1}^n B_i(T))$  be the value of the infeasible allocation  $B(T)$ . Moreover, let  $\mathbb{E}[W(A)] = \mathbb{E}[W(\cup_{i=1}^n A_i(T))]$  be the expected value of the feasible allocation  $A(T)$  made by **RANDOMONLINEGREEDY** (denoted as  $\mathcal{A}$ ), then from **Lemma 11** and

**Lemma 12**, the competitive ratio of  $\text{RANDOMONLINEGREEDY} = \max \left( \frac{W(A^*(T))}{\mathbb{E}[W(A(T))]} \right) \leq 6$ .  $\blacktriangleleft$

► **Example 14.** Example 1 can be extended to show that a lower bound of 2 on the competitive ratio of any randomized algorithm. Consider the randomized adversary that sends only the job in the first step (with weight  $\epsilon$ ) with probability  $1 - \epsilon$ , and both jobs (with weights  $\epsilon$  and 1) with probability  $\epsilon$ . Then any deterministic algorithm for this distribution gets value at most  $\epsilon$  while the optimal expected value is  $2\epsilon - \epsilon^2$ . The lower bound on randomized algorithms follows by an application of Yao's lemma [14].

## 5 Finite Span Jobs

We now generalise our model by assuming that the jobs do not consume server resources for infinite time, i.e, along with the job weight, the adversary also announces the span over which the job remains in the server. If a job is presented at time  $t'$  and has span  $s$ , then it consumes resources for  $t$  such that  $t' \leq t < t' + s$ . Once an allocated job expires, the capacity corresponding to the weight of that job is made available to the server for future job requests.

► **Example 15.** For each job let  $(w, s)$  be the tuple representing the weight and span, respectively. Let there be a single server with capacity  $C$ . Let the input sequences be  $S_1 = \{(\epsilon, T), \underbrace{(0, 1), \dots, (0, 1)}_{T-1}\}$ , and  $S_2 = \{(\epsilon, T), \underbrace{(\frac{C}{2}, T-1), (\frac{C}{2}/0, 1), \dots, (\frac{C}{2}/0, 1)}_{T-1}\}$ , where

$\frac{C}{2}/0$  means either the weight is  $\frac{C}{2}$  or 0 depending on earlier matchings. If at time  $t = 1$ , the job is not matched to the server, the competitive ratio on  $S_1$  is  $\infty$ . Otherwise, the adversary presents the sequence  $S_2$ , where if job at time  $t = 2$  is not matched then the weights of jobs for all further time instants are 0, and the competitive ratio is  $0.5C/\epsilon$ . If the server does accept the job at  $t = 2$ , then all future jobs have weight  $\frac{C}{2}$  and span 1. The server cannot accept any jobs for  $t \geq 3$  due to lack of capacity, and the competitive ratio is  $\frac{0.5C(T-2)}{0.5C+\epsilon} \approx T - 2$ . This shows that as  $T \rightarrow \infty$ , the competitive ratio can be made arbitrarily bad for all deterministic algorithms, even when edge weights are restricted to be at most half the server capacity.

### 5.1 Uniform Span

We first look at the case where all jobs have the same span  $s$ . For this case, we propose algorithms  $\text{UNIFORMGREEDY}$  and  $\text{RANDOMUNIFORMGREEDY}$ , which are similar to our previous algorithms  $\text{ONLINEGREEDY}$  (if each job weight is at most half the server capacity) and  $\text{RANDOMONLINEGREEDY}$  (for general weights), with the following modification. If the algorithm assigns job  $i$  to server  $j$  at time  $t$ , the resources used are released at time  $t + s$ . The analysis is similar to  $\text{ONLINEGREEDY}$  and  $\text{RANDOMONLINEGREEDY}$ . However, a more intricate argument is required since we can no longer argue about the jobs allocated at each time step. Instead, our analysis considers a window of size  $s$ , and obtains bounds on the weight of all jobs that are active within this window.

► **Theorem 16.** *UNIFORMGREEDY is 6-competitive where all job requests to a server are at most half the capacity of the corresponding server.*

► **Theorem 17.** *RANDOMUNIFORMGREEDY is 12-competitive.*

The proof follows similar to Theorem 13, with Theorem 16 replacing Theorem 3.

## 5.2 Non Uniform Span

RANDOMNONUNIFORMGREEDY is a  $\mathcal{O}\left(\log\left(\frac{s_{max}}{s_{min}}\right)\right)$ -competitive algorithm for the case when jobs may not have the same span. The algorithm works by dividing the jobs into  $\log\left(\frac{s_{max}}{s_{min}}\right)$  logarithmically spaced levels based on their span and accepting jobs that belong to only one level. This level is chosen uniformly at random before execution of the algorithm.

► **Theorem 18.** *RANDOMNONUNIFORMGREEDY is  $\mathcal{O}\left(\log\left(\frac{s_{max}}{s_{min}}\right)\right)$ -competitive.*

**Acknowledgments.** We are grateful to Anupam Gupta for helpful discussions on the problem, as well as to the anonymous referees of a previous version of this paper for their constructive comments.

---

### References

- 1 Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1253–1264, 2011.
- 2 Saeed Alaei, MohammadTaghi Hajiaghayi, and Vahid Liaghat. The online stochastic generalized assignment problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 11–25. Springer, 2013.
- 3 Niv Buchbinder, Kamal Jain, and Joseph Seffi Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Algorithms-ESA 2007*, pages 253–264. Springer, 2007.
- 4 Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 101–107, 2013.
- 5 Jon Feldman, Nitish Korula, Vahab S. Mirrokni, S. Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In *Internet and Network Economics, 5th International Workshop, WINE 2009, Rome, Italy, December 14-18, 2009. Proceedings*, pages 374–385, 2009.
- 6 Lisa Fleischer, Michel X Goemans, Vahab S Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 611–620. Society for Industrial and Applied Mathematics, 2006.
- 7 Bernhard Haeupler, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. In *Internet and Network Economics – 7th International Workshop, WINE 2011, Singapore, December 11-14, 2011. Proceedings*, pages 170–181, 2011.
- 8 Bala Kalyanasundaram and Kirk Pruhs. An optimal deterministic algorithm for online b-matching. *Theor. Comput. Sci.*, 233(1-2):319–325, 2000. doi:10.1016/S0304-3975(99)00140-1.
- 9 Ming-Yang Kao and Stephen R. Tate. Online matching with blocked input. *Inf. Process. Lett.*, 38(3):113–116, 1991.
- 10 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358, 1990.
- 11 Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *Automata, Languages and Programming*, pages 508–520. Springer, 2009.

## 30:14 The Adwords Problem with Strict Capacity Constraints

- 12 Benny Lehmann, Daniel J. Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006.
- 13 Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22, 2007.
- 14 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 222–227. IEEE, 1977.

# Most Likely Voronoi Diagrams in Higher Dimensions\*

Nirman Kumar<sup>1</sup>, Benjamin Raichel<sup>2</sup>, Subhash Suri<sup>3</sup>, and Kevin Verbeek<sup>4</sup>

- 1 University of Memphis, Memphis, USA  
nkumar8@memphis.edu
- 2 University of Texas, Dallas, USA  
bar150630@utdallas.edu
- 3 University of California, Santa Barbara, USA  
suri@cs.ucsb.edu
- 4 TU Eindhoven, The Netherlands  
k.a.b.verbeek@tue.nl

---

## Abstract

The Most Likely Voronoi Diagram is a generalization of the well known Voronoi Diagrams to a stochastic setting, where a stochastic point is a point associated with a given probability of existence, and the cell for such a point is the set of points which would classify the given point as its most likely nearest neighbor. We investigate the complexity of this subdivision of space in  $d$  dimensions. We show that in the general case, the complexity of such a subdivision is  $\Omega(n^{2d})$  where  $n$  is the number of points. This settles an open question raised in a recent (ISAAC 2014) paper of Suri and Verbeek [24], which first defined the Most Likely Voronoi Diagram. We also show that when the probabilities are assigned using a random permutation of a fixed set of values, in expectation the complexity is only  $\tilde{O}(n^{\lceil d/2 \rceil})$  where the  $\tilde{O}(\cdot)$  means that logarithmic factors are suppressed. In the worst case, this bound is tight up to polylog factors.

**1998 ACM Subject Classification** I.3.5 Computational Geometry and Object Modeling

**Keywords and phrases** Uncertainty, Lower bounds, Voronoi Diagrams, Stochastic

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.31

## 1 Introduction

Voronoi diagrams are a well known data structure in Computer Science. Given a finite set of points  $S$ , say in Euclidean  $d$ -space, the Voronoi diagram is a partition of space into cells, one for each point of  $S$ , such that the cell for each point  $s \in S$  contains all points closer to  $s$  than to any other point of  $S$ . We study this classical data structure in the presence of uncertainty. For example, consider a situation in which a set of facilities (say car repair shops) are modeled as points, and the probability that a particular facility can provide a desired service (repairing your car) is known. A natural question is then, which facility is the most likely to be the nearest facility that can provide the desired service.

While uncertainty has been modeled in several ways in different contexts, we investigate the problem in the presence of *existential uncertainty*. The input in this model is a set

---

\* Work by N. Kumar and S. Suri was partially supported by NSF grants CCF-1161495 and CCF-1525817. Work by B. Raichel was partially supported by NSF CRII award CCF-1566137. Work by K. Verbeek was partially supported by the Netherlands Organization for Scientific Research (NWO) under grant number 639.021.541.





$\mathcal{P} = \{p_1, \dots, p_n\}$  of  $n$  stochastic points, where each stochastic point  $p_i$  is a tuple  $(s_i, \pi_i)$  where  $s_i$  is a regular point in  $\mathbb{R}^d$  (we will call them sites, or simply points when it is clear they are not meant as probabilistic points) and  $\pi_i$  is its probability of existence. Consider the product distribution induced by these individual distributions. Under this distribution, we can compute for any query point  $x$ , its most likely nearest neighbor (MLNN). The partition of  $\mathbb{R}^d$  where for each  $p_i$  we have the associated region of points which would classify it as its most likely nearest neighbor, is called the Most Likely Voronoi Diagram (MLVD). This data structure was introduced in [24], where the authors investigated its properties in the simplest setting of  $d = 1$  dimensions. Even in this simple setting, it was not obvious how to bound the worst case complexity of this data structure, but it was shown to be  $\Theta(n^2)$ . However, the lower bound construction relied on very carefully chosen probability values and location of the sites involved. Under certain conditions, such as the probabilities assigned to the sites coming from a random permutation of a fixed set of values, the authors showed an upper bound of  $O(n \log n)$  on the complexity. We investigate the complexity of this data structure in higher dimensions.

**Results.** Our contributions can be summarized as follows.

- We show that in the worst case, the complexity of the MLVD is  $\Omega(n^{2d})$ . This settles an open question raised in [24].
- When the probabilities assigned to the stochastic points come from a random permutation on a fixed set of  $n$  values, we show that the expected complexity of the MLVD is  $\tilde{O}(n^{\lceil d/2 \rceil})$  where the  $\tilde{O}(\cdot)$  means that terms poly-logarithmic in  $n$  are suppressed. Note that this includes the case when all values are independently sampled from a single distribution (as one can first sample and then randomly permute the values). This generalizes a result of [24]. In the worst case, this bound is tight up to polylog factors.

**Related work.** The work most closely related to ours is of course the paper [24] which defined the MLVD and investigated upper and lower bounds for it in 1-d. Under the aegis of proximity searching, there has also been work under different uncertainty models [5, 3]. These papers investigate different definitions of closeness in the presence of uncertainty, and thereby the resulting Voronoi diagrams are different. For example the Expected Nearest Neighbor (ENN) Voronoi diagram is defined in [5], and the nonzero Nearest Neighbor Voronoi diagram is studied in [3]. These works focus on the 2-dimensional case, and even there tight bounds on the complexity of the Voronoi diagrams defined are not known. More generally, there has been a lot of work on uncertainty in several communities including databases, machine learning, optimization, and computational geometry [7, 8, 15]. Several fundamental problems have been studied in uncertain settings; see [5, 3] for work on nearest neighbors, [4] for work on range searching, [2] for skylines, and [1] for work on coresets. In the existential uncertainty model there has been a flurry of recent work on convex hulls [6], separability [13, 25] (see also [11] for work on separability in a different model of uncertainty), containment and evasion problems [20], arrangements [21], skylines [9] and optimization problems such as minimum spanning trees, and closest pair problems [16, 17].

For the upper bound, we use the candidate set technique, developed by Har-Peled and Raichel [14], and in particular the notation and background we present to use this technique closely follows that of [14]. Their aim was to bound the expected complexity of the multiplicative Voronoi diagram. While such diagrams differ from the MLVD, the candidate set technique is general enough, that with some modification and generalization, it can also be used to bound the complexity of the MLVD. We remark that while our upper bound



proof is a significant contribution of this paper, in particular as it generalizes the bounds of [24] to higher dimensions, the main new technical contribution is our lower bound proof.

## 2 Preliminaries

**Notation.** Let  $S$  be a set of  $n$  points in general position in  $\mathbb{R}^d$ , which we call **sites**. Throughout the paper we assume  $d$  is a constant. Let  $\Pi$  be a set of  $n$  values in the interval  $(0, 1)$ . The values in  $\Pi$  are indexed in decreasing order,  $\pi_1 \geq \dots \geq \pi_n$ .

We use  $T = \langle s_1, \dots, s_n \rangle$  to denote a random permutation of the sites in  $S$ . Let  $T_i = \langle s_1, \dots, s_i \rangle$  denote the **prefix** of this permutation of length  $i$ .

When we care only about what elements appear in a permutation,  $T$ , but not their internal ordering, we use the notation  $S = \text{set}(T)$  to denote the associated set. As such,  $S_i = \text{set}(T_i)$  is the **unordered prefix** of length  $i$  of  $T$ .

We let  $[n]$  denote the set  $\{0, 1, \dots, n-1\}$  for any natural number  $n$ .

**Arrangements.** As it will be used throughout the paper, we now define the standard terminology of arrangements (see [23]). A set  $H$  of  $n$  hyperplanes in  $\mathbb{R}^d$ , induces a partition of  $\mathbb{R}^d$  into connected cells, called the arrangement of  $H$ , and denoted  $\mathcal{A}(H)$ . Specifically, each cell is a maximal connected region of the intersection of a (possibly empty) subset of  $H$ , which does not intersect any hyperplane not in this subset. In particular, the  $d$ -dimensional cells are the maximal connected subsets of  $\mathbb{R}^d$  which do not intersect any hyperplanes in  $H$ . The combinatorial complexity of  $\mathcal{A}(S)$  is the total number of cells of all dimensions.

### 2.1 Voronoi diagrams

Let  $S = \{s_1, \dots, s_n\}$  be a set of  $n$  point sites in the  $\mathbb{R}^d$ . For a closed set  $Y \subseteq \mathbb{R}^d$ , and any point  $x \in \mathbb{R}^d$ , let  $d(x, Y) = \min_{y \in Y} \|x - y\|$  denote the **distance** of  $x$  to the set  $Y$ . For any two sites  $s, r \in S$ , we define their **bisector**  $\beta(s, r)$  as the set of points  $x \in \mathbb{R}^d$  such that  $d(x, s) = d(x, r)$ . Clearly,  $\beta(s, r)$  is a hyperplane, passing through the midpoint of the segment  $[s, r]$  and orthogonal to it.

Each  $s \in S$ , induces the function  $f_s(x) = d(x, s)$ , where  $x$  is any point in  $\mathbb{R}^d$ . For any subset  $H \subseteq S$  and any site  $s \in H$ , the **Voronoi cell** of  $s$  with respect to  $H$ , denoted  $\mathcal{V}_{\text{cell}}(s, H)$ , is the subset of  $\mathbb{R}^d$  whose closest site in  $H$  is  $s$ , i.e.  $\mathcal{V}_{\text{cell}}(s, H) = \{x \in \mathbb{R}^d \mid \forall r \in H \ f_s(x) \leq f_r(x)\}$ . Finally, for any subset  $H \subseteq S$ , the Voronoi diagram of  $H$ , denoted  $\mathcal{V}(H)$ , is the partition of space into Voronoi cells induced by the minimization diagram of the set of functions  $\{f_s \mid s \in H\}$ .

### 2.2 Most Likely Voronoi diagrams

We now consider a set  $\mathcal{P}$  of  $n$  stochastic points where the  $i$ th stochastic point  $p_i = (s_i, \pi_i)$ , and  $\pi_i > 0$ .

For a given query point  $x$ , let  $B_i(x)$  denote the set of sites in the open ball with center  $x$  and radius  $\|x - s_i\|$ . The probability that a site  $s_i$  is the nearest neighbor to a query point  $x$  is given by the expression

$$\Pi_{nn}(s_i, x) = \pi_i \prod_{s_j \in B_i(x)} (1 - \pi_j) \quad (1)$$

The **most likely nearest neighbor** of the query point  $x$  is then  $MLNN(x) = \arg \max_{s_i \in \mathcal{P}} \Pi_{nn}(p_i, x)$ .

For any subset  $H \subseteq S$  and any point  $s \in H$ , the **most likely Voronoi cell** of  $s$  with respect to  $H$ , denoted  $\mathcal{M}_{\text{cell}}(s, H)$ , is the subset of  $\mathbb{R}^d$  whose most likely nearest neighbor in  $H$  is  $s$ , i.e.  $\mathcal{M}_{\text{cell}}(s, H) = \left\{x \in \mathbb{R}^d \mid \forall r \in H \ \Pi_{nn}(s, x) \geq \Pi_{nn}(r, x)\right\}$ . Finally, for any subset  $H \subseteq S$ , the **most likely Voronoi diagram** (MLVD) of  $H$ , denoted  $\mathcal{M}(H)$ , is the partition of space into Voronoi cells induced by the maximization diagram of the set of functions  $\{\Pi_{nn}(s, \cdot) \mid s \in H\}$ . The MLVD is a polyhedral partition of space such that the cell for each site is the union of a set of polyhedral sets, where some of these sets may possibly be open. The cell for a given site is not necessarily a connected set. The **complexity** of the MLVD is the total number of faces, over all dimensions, of this polyhedral partition.

Let  $g(n)$  denote the worst case complexity of the most likely Voronoi diagram. Consider the arrangement of all the bisectors of the points involved. This has complexity  $O(n^{2d})$ . Within each cell of the arrangement, the ordering of distances is fixed and as such the probability of each site to be a MLNN is also fixed, where we will resolve ties using a fixed (but arbitrary ordering of the points). Therefore, within a cell the MLNN is the same for all points. Thus,  $O(n^{2d})$  is an upper bound on the total complexity of the MLVD. However, the total complexity of the MLVD can be smaller as adjacent cells can merge and the MLVD can be a true coarsening of the arrangement.

### 3 Bounding the expected complexity of the MLVD

For most of this section, we follow closely the presentation in [14] to develop the relevant machinery. Let  $S$  be a set of sites and  $T = \langle s_1, \dots, s_n \rangle$  be a random permutation of the sites in  $S$ . Let  $\pi_1 \geq \pi_2 \geq \dots \geq \pi_n$  be a fixed set of  $n$  probability values in  $(0, 1)$ . Consider the (random) set of stochastic points  $\mathcal{P}$  where the  $i$ th stochastic point  $p_i = (s_i, \pi_i)$ . In this section we show that the expected value of the complexity of the MLVD of  $\mathcal{P}$  where the expectation is over the random permutation  $T$ , is given by  $\tilde{O}(n^{\lceil d/2 \rceil})$  where the  $\tilde{O}(\cdot)$  suppresses factors logarithmic in  $n$ .

#### 3.1 Candidate sets

► **Definition 1.** Let  $T = \langle s_1, \dots, s_n \rangle$  be an ordered set of  $n$  sites in  $\mathbb{R}^d$ . For any point  $x$  in the  $\mathbb{R}^d$ , the **candidate set** of  $x$ , denoted by  $L(x, T)$ , is the set of all sites  $s_i \in T$ , such that  $\|x - s_i\| = \mathbf{d}(x, T_i)$ , for  $i = 1, \dots, n$ . In words,  $s_i$  is in  $L(x, T)$  if it is the closest site to  $x$  in its prefix  $T_i$ .

Suppose we assign probabilities to the sites  $s_i$  such that  $\pi_i$  is assigned to  $s_i$ , where recall that  $\pi_1 \geq \pi_2 \geq \dots \geq \pi_n$ . A prerequisite for a site  $s_j$  of  $S$  to be the most likely nearest neighbor to  $x$ , is that  $s_j$  is in the candidate set  $L(x, T)$ .

► **Lemma 2.** For a point  $x$  in  $\mathbb{R}^d$ , if  $MLNN(x) = s_j$ , then  $s_j$  is in  $L(x, T)$ , where  $T$  is the ordering of  $S$  by decreasing probabilities.

**Proof.** Let  $s_j$  be the most likely nearest neighbor of  $x$ , and suppose that  $\|x - s_j\| \neq \mathbf{d}(x, T_j)$ . This implies there exists some  $i < j$  such that  $\|x - s_i\| < \|x - s_j\|$ . However, by the definition of  $T$ , for  $i < j$ , we have  $\pi_i \geq \pi_j$ , and so

$$\begin{aligned} \Pi_{nn}(s_j, x) &= \pi_j \prod_{s_k \in B_j(x)} (1 - \pi_k) \leq \pi_i \prod_{s_k \in B_j(x)} (1 - \pi_k) \\ &< \pi_i \prod_{s_k \in B_i(x)} (1 - \pi_k) = \Pi_{nn}(s_i, x), \end{aligned}$$

which is a contradiction. Therefore,  $s_j$  must be the closest point to  $x$  in its prefix  $T_j$ . ◀

Consider a random permutation  $T$  of  $S$ , and the candidate set as a random variable. Next, we investigate the size of this set for all points of space. We prove that with high probability, the candidate set is logarithmic in size for all points in space. To this end, we need the following well known fact. The expectation bound is well known from [19, 10], and the high probability result was probably well known before as well, but see [14] for a recent written proof.

► **Lemma 3.** *Let  $\Pi = \langle \pi_1, \dots, \pi_n \rangle$  be a random permutation of  $\{1, \dots, n\}$ , and let  $X_i$  be an indicator variable which is 1 if  $\pi_i$  is the smallest number among  $\pi_1, \dots, \pi_i$ , for  $i = 1, \dots, n$ . Let  $Z = \sum_{i=1}^n X_i$ , then  $Z = O(\log n)$ , with high probability (i.e., for any constant  $c > 0$ , one can choose the constant in the  $O(\cdot)$  above such that the probability is at least  $\geq 1 - 1/n^c$ ).*

► **Corollary 4.** *Let  $\pi_1 \geq \dots \geq \pi_n$  be a set of  $n$  probabilities in  $(0, 1)$ . Let  $S$  be a set of  $n$  points in  $\mathbb{R}^d$ , and let  $T = \langle s_1, \dots, s_n \rangle$  be a random permutation of  $S$ . Assign the probability  $\pi_i$  to  $s_i$ , for all  $i$ . Then simultaneously for all points in  $\mathbb{R}^d$ , their candidate set for  $T$  is of size  $O(\log n)$ , with high probability.*

**Proof.** Fix a point  $x \in \mathbb{R}^d$ . Since  $T = \langle s_1, \dots, s_n \rangle$  is a random permutation of  $S$ , the sequence  $\|x - s_1\|, \dots, \|x - s_n\|$  is a random permutation of the distance values from  $x$  to the sites in  $S$ . Therefore, by the definition of the candidate set and Lemma 3, we have  $|L(x, T)| = O(\log n)$  with high probability.

Consider the arrangement of all the bisectors of all the pairs of sites in  $S$  of complexity  $O(n^{2d})$ . Within each face of this arrangement, the candidate set cannot change, for any permutation, since all points in this face have the same ordering of their distances to the sites in  $S$ . So pick a representative point for each of the  $O(n^{2d})$  faces. For any such representative, with probability  $\leq 1/n^c$ , the candidate set has  $> \alpha \log(n)$  sites, for any constant  $c$  of our choosing (where  $\alpha$  is a constant that depends only on  $c$ ). Therefore, by choosing  $c$  to be sufficiently large, taking the union bound on the bad events (where a bad event is that the size of the candidate set for some face exceeds  $\alpha \log(n)$ ), and then taking the complement, the claim follows. ◀

### 3.2 Getting a compatible partition

The goal now is to find a low complexity subdivision of space, such that within each cell of the subdivision the candidate set is fixed. As we know, the arrangement of the bisectors already provides such a subdivision. However, the complexity of this subdivision is high. The main insight is that by using the standard Voronoi diagram one can get such a subdivision, which

- (A) is sensitive to an ordering of the sites (thus, it can intuitively save on certain permutations over the worst case), and,
- (B) its complexity in expectation can be bounded by  $\tilde{O}(n^{\lceil d/2 \rceil})$ .

Let  $K_i$  denote the Voronoi cell of  $s_i$  in the usual Voronoi diagram of the  $i$ th prefix  $S_i = \{s_1, \dots, s_i\}$ . Let  $\mathcal{A}$  denote the arrangement formed by the overlay of the regions  $K_1, \dots, K_n$ . The complexity of  $\mathcal{A}$ , denoted by  $|\mathcal{A}|$ , is the total number of these faces of all dimensions in the arrangement.

► **Lemma 5.** *For any face  $F$  of  $\mathcal{A} = \mathcal{A}(K_1, \dots, K_n)$ , the candidate set is the same, for all points in  $F$ .*

**Proof.** Consider adding points in the order  $s_1, \dots, s_n$ . Initially, before any points are added, all points in  $\mathbb{R}^d$  have the same candidate set, namely the empty set. When the site  $s_i$  is

## 31:6 Most Likely Voronoi Diagrams in Higher Dimensions

added, the only points in  $\mathbb{R}^d$  whose candidate set changes are those for which  $s_i$  is their nearest neighbor in  $S_i$ . However, these are precisely the points in the Voronoi cell of  $s_i$  in the usual Voronoi diagram of  $S_i$ . That is, the candidate set changes only for the points covered by  $K_i$ .

The claim now easily follows, as  $\mathcal{A}$  is the overlay arrangement of these regions.  $\blacktriangleleft$

► **Theorem 6.** *Let  $\pi_1 \geq \dots \geq \pi_n$  be a set of  $n$  probabilities in  $(0, 1)$ . Let  $S$  be a set of  $n$  points in  $\mathbb{R}^d$ , and let  $T = \langle s_1, \dots, s_n \rangle$  be a random permutation of  $S$ , where probability  $\pi_i$  is assigned to  $s_i$ , for all  $i$ . Let  $K_i = \mathcal{V}_{\text{cell}}(s_i, T_i)$ , for  $i = 1, \dots, n$ . Finally, let  $\mathcal{A} = \mathcal{A}(K_1, \dots, K_n)$  be the arrangement formed by the overlay of all these cells.*

*Then, the expected complexity of the most likely Voronoi diagram is  $O(\mathbf{E}[|\mathcal{A}|] g(\log n))$ , where  $|\mathcal{A}|$  is the total complexity of  $\mathcal{A}$ , and  $g(m)$  denotes the worst case complexity of the most likely Voronoi diagram of  $m$  sites.*

**Proof.** Let  $Z$  be the set of all permutations of  $S$ . For any  $z \in Z$ , let  $C(z)$  denote the size of the largest candidate set of any point in  $\mathbb{R}^d$  determined by  $z$ . We first argue that for  $z$  sampled uniformly at random from  $Z$ ,  $\mathbf{E}[|\mathcal{A}| g(C(z))] = O(\mathbf{E}[|\mathcal{A}|] g(\log n))$ .

Partition  $Z$  into two sets, **good** and **bad**, such that for any  $z \in Z$ ,  $z \in \text{good}$  if  $C(z) \leq \alpha \log n$ , and  $z \in \text{bad}$  otherwise, for some constant  $\alpha$ . Using Corollary 4, we choose  $\alpha$  large enough such that for  $z$  sampled uniformly at random from  $Z$ ,  $\mathbf{P}[z \in \text{bad}] \leq 1/n^\beta$ , where  $\beta = \beta(\alpha)$  is some sufficiently large constant to be determined shortly. We then have:

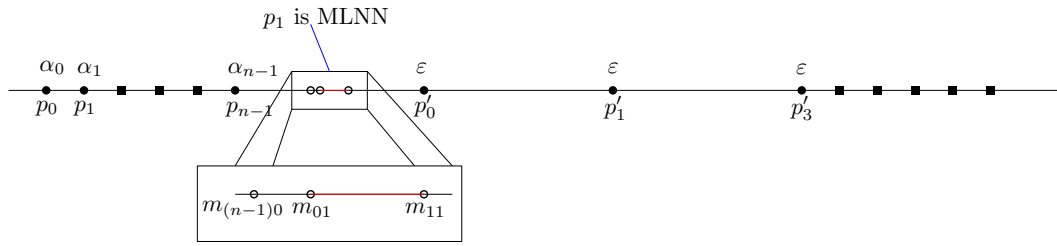
$$\begin{aligned} & \mathbf{E}[|\mathcal{A}| g(C(z))] \\ &= \mathbf{E}[|\mathcal{A}| g(C(z)) \mid z \in \text{good}] \mathbf{P}[z \in \text{good}] + \mathbf{E}[|\mathcal{A}| g(C(z)) \mid z \in \text{bad}] \mathbf{P}[z \in \text{bad}] \\ &\leq \mathbf{E}[|\mathcal{A}| g(\alpha \log n) \mid z \in \text{good}] \mathbf{P}[z \in \text{good}] + \mathbf{E}[|\mathcal{A}| g(C(z)) \mid z \in \text{bad}] / n^\beta \\ &= g(\alpha \log n) \mathbf{E}[|\mathcal{A}| \mid z \in \text{good}] \mathbf{P}[z \in \text{good}] + \mathbf{E}[|\mathcal{A}| g(C(z)) \mid z \in \text{bad}] / n^\beta. \end{aligned}$$

Now the first term in the above sum is bounded by  $g(\alpha \log n) \mathbf{E}[|\mathcal{A}|]$  because of the following equality:  $\mathbf{E}[|\mathcal{A}| \mid z \in \text{good}] \mathbf{P}[z \in \text{good}] = \mathbf{E}[|\mathcal{A}|] - \mathbf{E}[|\mathcal{A}| \mid z \in \text{bad}] \mathbf{P}[z \in \text{bad}] \leq \mathbf{E}[|\mathcal{A}|]$ . To bound the second term in the sum, observe that both  $|\mathcal{A}|$  and  $g(C(z))$  are in the worst case bounded by  $O(n^{2d})$ , as both the MLVD and  $|\mathcal{A}|$  are coarsenings of the arrangement of all bisectors of the sites, which itself has complexity  $O(n^{2d})$ . Hence by choosing  $\beta > 4d$  we can bound the second term of the above expectation as:  $\mathbf{E}[|\mathcal{A}| g(C(z)) \mid z \in \text{bad}] / n^\beta = O(n^{4d}/n^\beta) = O(1)$ . Combining the bounds on each term of the sum we get:

$$\mathbf{E}[|\mathcal{A}| g(C(z))] \leq g(\alpha \log n) \mathbf{E}[|\mathcal{A}|] + O(1) = O(\mathbf{E}[|\mathcal{A}|] g(\log n)).$$

We now argue that for any fixed  $z \in Z$ , the corresponding complexity of  $\mathcal{M}(S)$  is  $O(|\mathcal{A}| g(C(z)))$ , which combined with the above bound on the expectation will complete the proof. First decompose all faces (of all dimensions) of  $\mathcal{A}$  into constant complexity simplices. (Note that the simplices are constant complexity since  $d$  is constant). This can be done by computing a bottom vertex triangulation (see for example [22]). Again since  $d$  is assumed to be constant, this triangulation has the same asymptotic complexity as  $|\mathcal{A}|$ .

Now we have a partition of space into  $O(|\mathcal{A}|)$  constant complexity simplices, and by Lemma 5 within each such simplex the candidate set is fixed. So consider such a simplex  $\Delta$ , and let  $L$  be its candidate set. Lemma 2 implies that the only sites whose most likely Voronoi cells can have non-zero area in  $\Delta$  are the sites in  $L$ . That is, the most likely Voronoi diagram restricted to  $\Delta$  is the intersection of  $\Delta$  with the most likely Voronoi diagram of some subset of  $L$ . Now the most likely Voronoi diagram of  $\leq |L|$  points has worst case complexity  $g(|L|)$ . Since  $\Delta$  is a constant complexity region this implies that the complexity of the most



■ **Figure 1** The lower bound example in 1-d with  $\Omega(n^2)$  complexity.

likely Voronoi diagram in  $\Delta$  is  $O(g(|L|))$ . By definition, no candidate set has size more than  $C(z)$ , and hence for any simplex  $\Delta$ ,  $O(g(|L|)) = O(g(C(z)))$ . Hence the total complexity over all simplices is  $O(|\mathcal{A}|g(C(z)))$ . ◀

A naive bound on the worst case complexity is  $g(m) = O(m^{2d})$ . Kaplan *et al.* [18] showed that for a random permutation of  $n$  points (as is the case here) the expected total complexity of  $\mathcal{A}$  is  $O(n^{\lceil d/2 \rceil} \log n)$  when  $d$  is even, and  $O(n^{\lceil d/2 \rceil})$  when  $d$  is odd. We therefore readily have the following result.

► **Theorem 7.** *Let  $\pi_1 \geq \dots \geq \pi_n$  be a set of  $n$  probabilities in  $(0, 1)$ . Let  $S$  be a set of  $n$  points in  $\mathbb{R}^d$ , and let  $T = \langle s_1, \dots, s_n \rangle$  be a random permutation of  $S$ , where probability  $\pi_i$  is assigned to  $s_i$ , for all  $i$ .*

*Then the expected complexity of the most likely Voronoi diagram is  $O(n^{\lceil d/2 \rceil} \log^{2d+1} n)$  when  $d$  is even, and  $O(n^{\lceil d/2 \rceil} \log^{2d} n)$  when  $d$  is odd.*

► **Corollary 8.** *Let  $S$  be a set of  $n$  points in  $\mathbb{R}^d$ , where independently for each site  $s$  we sample a probability value from a single fixed distribution over  $(0, 1)$ . Then the expected complexity of the most likely Voronoi diagram is  $O(n^{\lceil d/2 \rceil} \log^{2d+1} n)$  when  $d$  is even, and  $O(n^{\lceil d/2 \rceil} \log^{2d} n)$  when  $d$  is odd.*

**Proof.** The distribution induced by choosing the  $\pi_i$  from a fixed distribution over  $(0, 1)$  is the same as would be induced by first choosing  $\pi_1$  then, choosing  $\pi_2 \leq \pi_1$  and so on, and then permuting them randomly. Under every random permutation for any fixed choice of the  $\pi_i$  the complexity of the most likely Voronoi diagram is bounded by Theorem 7, and this expression is independent of the choice of the  $\pi_i$ . As such in expectation over the  $\pi_i$  we have the same bound. ◀

Notice that the regular Voronoi diagram of any set of sites is a special case of the MLVD since if all the probabilities are equal, the MLNN is always the nearest neighbor. Our bound above holds for *any* set of sites and a random assignment of probability values from an arbitrary set, while it is known that the worst case complexity of the regular Voronoi diagram is  $\Omega(n^{\lceil d/2 \rceil})$  [12], hence it follows that the bound we establish is tight up to polylog factors in the worst case.

#### 4 Worst-case lower bound

In this section we show that  $g(n) = \Omega(n^{2d})$ . Since the construction is a generalization and uses part of the construction for the lower bound in  $d = 1$  dimensions from [24], we briefly recall it here.

The construction of [24] uses two groups of stochastic points each with  $n$  points. In the first group  $S$  the stochastic points are  $p_i = ((i + 1)/n, \alpha_i)$  for  $i = 0, 1, \dots, n - 1$ ,

where in [24] they choose  $\alpha_i = 1/(i+1)$ , but as we show different sets of values also work. In the second group  $T$  the stochastic points are  $p'_i = (2+j, \varepsilon)$  for  $\varepsilon$  sufficiently small, for  $j = 0, 1, \dots, n-1$ . They also follow the convention that if two points have the same minimum probability of being the MLNN at a point the one with the smaller index is designated as the MLNN. With this convention they basically show that the MLNN changes across each of the bisector points of  $S$  and  $T$ . The  $\Omega(n^2)$  bisection points  $m_{ij}$  are  $m_{ij} = i/(2n) + j/2 + (1 + 1/(2n))$  and these are ordered as per the sequence  $m_{00}, m_{10}, \dots, m_{(n-1)0}, m_{01}, m_{11}, \dots, m_{(n-1)1}, \dots, m_{0(n-1)}, m_{1(n-1)}, \dots, m_{(n-1)(n-1)}$ , i.e. in lexicographical order first by  $j$  and then by  $i$ , see Figure 1 for an illustration. They choose  $\varepsilon$  so small that a probabilistic point in  $T$  can never be the MLNN. For their choice of the  $\alpha_i = 1/(i+1)$ , it turns out it is sufficient to have  $\varepsilon$  be so small as to satisfy  $(1-\varepsilon)^n/n > \varepsilon$  which is achieved for  $\varepsilon \leq 1/n^2$ . Next, observe that, for the choice  $\alpha_i = 1/(i+1)$ , at a point infinitesimally to the left of  $m_{ij}$  the probability of  $p_k$  being the MLNN is precisely  $(1-\varepsilon)^j/n$  for  $k \geq i$  and it is  $(1-\varepsilon)^{j+1}/n$  for  $0 \leq k < i$ . Thus, to the left of midpoint  $m_{ij}$  the MLNN is  $p_i$  and therefore it assumes  $n$  values before  $j$  increments by 1 at which point it cycles through all of  $p_i$  again for  $i = 0, \dots, n-1$ .

We need a small modification of the above construction for our proof. Notice that the probabilities for being the MLNN are always of the form  $\frac{(1-\varepsilon)^j}{n}$ . However, as we show for any  $p$  sufficiently small (depending on a function of  $n$ ) we can choose probabilities for the points in  $S$  and  $T$  so that the probabilities of the MLNN are always of the form  $(1-\varepsilon)^j p$ . This can be seen from the following lemma, and by choosing  $\varepsilon$  small enough, so that a probabilistic point in  $T$  is never the MLNN. Intuitively, we can use the  $\alpha_i$  for the probabilities of the sites in  $S$  in the construction outlined above.

► **Lemma 9.** *For any  $n \geq 1$  and  $0 < \alpha_0 \leq 1/n$ , there exist numbers  $\alpha_1, \dots, \alpha_{n-1} \in (0, 1]$  such that*

$$\alpha_0 = \alpha_1(1 - \alpha_0) = \alpha_2(1 - \alpha_1)(1 - \alpha_0) = \dots = \alpha_{n-1}(1 - \alpha_{n-2}) \dots (1 - \alpha_0).$$

**Proof.** Given  $\alpha_0 \in (0, 1/n]$  we can define  $\alpha_1 = \alpha_0/(1 - \alpha_0)$  and then continue inductively by  $\alpha_i = \alpha_{i-1}/(1 - \alpha_{i-1})$ . This ensures that,

$$\alpha_0 = \alpha_1(1 - \alpha_0) = \alpha_2(1 - \alpha_1)(1 - \alpha_0) = \dots = \alpha_{n-1}(1 - \alpha_{n-2}) \dots (1 - \alpha_0).$$

It can be verified that,  $\alpha_i = \alpha_0/(1 - i\alpha_0)$  and clearly  $\alpha_0 \leq \alpha_1 \leq \dots \leq \alpha_{n-1}$ . The condition  $\alpha_{n-1} \leq 1$  implies  $\alpha_0 \leq 1/n$ , and moreover, any such  $\alpha_0$  leads to a valid sequence. ◀

We need another lemma for the proof below.

► **Lemma 10.** *For any  $\delta$  small enough, there exist numbers  $\pi_0, \dots, \pi_{n-1}$  and  $\alpha_0, \dots, \alpha_{n-1}$  all in  $(0, 1)$ , such that for any  $\varepsilon$  small enough the following are satisfied:*

- (A)  $\alpha_0 = \alpha_1(1 - \alpha_0) = \alpha_2(1 - \alpha_1)(1 - \alpha_0) = \dots = \alpha_{n-1}(1 - \alpha_{n-2}) \dots (1 - \alpha_0)$ .
- (B)  $\pi_0 = \pi_1(1 - \pi_0)(1 - \delta) = \pi_2(1 - \pi_1)(1 - \pi_0)(1 - \delta)^2 = \dots = \pi_{n-1}(1 - \pi_{n-2}) \dots (1 - \pi_0)(1 - \delta)^{n-1}$ .
- (C)  $\varepsilon < (1-\varepsilon)^n \alpha_0$ ,  $\delta < \pi_0(1-\delta)$ ,  $\pi_0 > \alpha_0$ , and  $\pi_0(1-\delta) < (1-\varepsilon)^n \alpha_0$ , i.e.,  $[(1-\varepsilon)^n \alpha_0, \alpha_0] \subseteq ((1-\delta)\pi_0, \pi_0)$ .

**Proof.** Start with a symbolic  $\delta$  and  $\pi_0$  and compute  $\pi_1, \dots, \pi_{n-1}$  iteratively by  $\pi_i = \frac{\pi_{i-1}}{(1-\pi_{i-1})(1-\delta)}$  for  $i = 1, \dots, n-1$ . This recursive definition guarantees that  $\pi_0 = \pi_1(1 - \pi_0)(1 - \delta) = \pi_2(1 - \pi_1)(1 - \pi_0)(1 - \delta)^2 = \dots = \pi_{n-1}(1 - \pi_{n-2}) \dots (1 - \pi_0)(1 - \delta)^{n-1}$ . It can

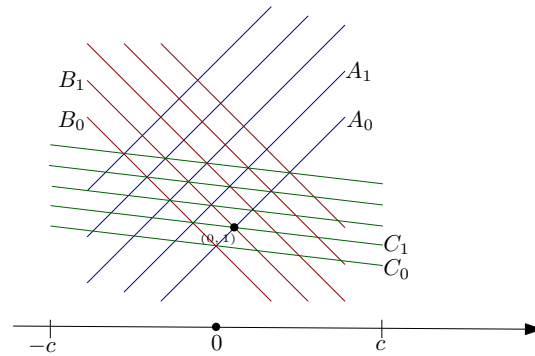


Figure 2 The grid of lines  $A_i, B_j, C_k$ .

be verified that,

$$\pi_i = \frac{\pi_0}{(1 - \pi_0)(1 - \delta)^i - \left(\pi_0 \sum_{j=1}^{i-1} (1 - \delta)^j\right)}.$$

It can be seen that  $\pi_0 < \pi_1 < \dots < \pi_{n-1}$ . The constraint  $\pi_{n-1} \leq 1$  gives us,

$$\pi_0 \leq \frac{(1 - \delta)^{n-1}}{1 + (1 - \delta) + \dots + (1 - \delta)^{n-1}}.$$

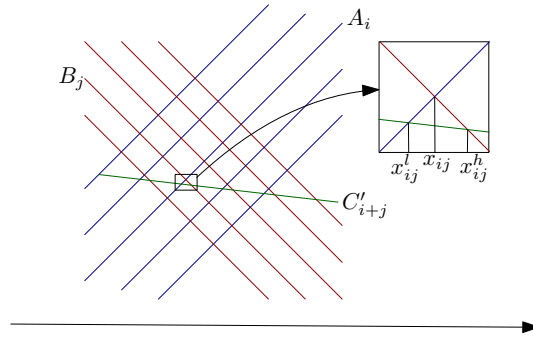
When  $\delta \rightarrow 0$  the upper bound for  $\pi_0$  goes to  $1/n$  and when  $\delta$  increases the upper bound decreases. By choosing a very small  $\delta$  and  $\pi_0$  to be the upper bound we get for it, we can ensure  $1/n > \pi_0(1 - \delta) > \delta$ . The first condition can be satisfied for any small enough  $\alpha_0 \leq 1/n$  by Lemma 9. In particular we choose  $\alpha_0$  such that  $\alpha_0$  lies in  $((1 - \delta)\pi_0, \pi_0)$  and then choose  $\varepsilon$  small enough so that  $\pi_0(1 - \delta) < \alpha_0(1 - \varepsilon)^n$  as well as  $\varepsilon < (1 - \varepsilon)^n \alpha_0$ . Thus, all the desired conditions can be satisfied, and moreover they are satisfied for every small enough  $\varepsilon$ . Also, any starting choice of  $\delta$  which is small enough will work. ◀

► **Theorem 11.** *There is a set of  $(3d - 1)n$  probabilistic points in  $\mathbb{R}^d$  whose MLVD has complexity at least  $\Omega(n^{2d})$ .*

**Proof.** The proof is an inductive argument. The base case, the result for  $d = 1$  was already shown in [24], and is sketched above. We show how the induction step works for  $d = 2$  dimensions; the general case is similar and we sketch the details later.

We consider four sets of sites  $P, Q, Q_l$ , and  $Q_r$ , where  $|P| = 2n$  and  $|Q| = |Q_l| = |Q_r| = n$ . We first explain how we place the sites in  $Q_l \cup Q_r \cup Q$ . To understand how the sites are placed we need to understand how the distance to them varies from a point  $(x, 0)$  on the  $x$ -axis. Since it is equivalent to consider squared distance functions we will work with them instead. Consider the function  $f_{(a,b)}(x)$  which is the square of the distance function of point  $(a, b)$  to point  $(x, 0)$  on the  $x$ -axis. We have  $f_{(a,b)}(x) = (a - x)^2 + b^2$ . The graph of each such function is a parabola but when  $x$  is small the graph is approximately a straight line. In order to define the placement of sites in  $Q_l \cup Q_r \cup Q$  we will choose sites such that the graphs of the corresponding distance functions are approximately the lines of the grid we define below. We let  $Q_l = \{(a_0, b_0), \dots, (a_{n-1}, b_{n-1})\}$ ,  $Q_r = \{(c_0, d_0), \dots, (c_{n-1}, d_{n-1})\}$ ,  $Q = \{(e_0, f_0), \dots, (e_{n-1}, f_{n-1})\}$  and let the corresponding distance functions be  $F_i, G_j, H_k$  i.e.,  $F_i(x) = f_{(a_i, b_i)}(x), G_j(x) = f_{(c_j, d_j)}(x), H_k(x) = f_{(e_k, f_k)}(x)$  for  $i, j, k \in [n]$ . It will be clear towards the end of the proof what the values of the coordinates are.





■ **Figure 3** The grid with  $C_k$  pushed down to  $C'_k$ . In  $[x_{ij}^l, x_{ij}^h]$  the line  $C'_k$  where  $k = i + j$  lies above  $k$  of the  $A_i, B_j$  otherwise, not near an intersection point it lies strictly above  $k + 1$  of them.

To define the grid, consider the lines  $A_i, B_j, C_k$  for  $i, j, k \in \{0, \dots, n-1\}$ , where  $A_i$  has the equation:  $y = x + ci/n + ci/n^2 + M$  where  $c > 0$  is a number depending on  $n$  we define later, and  $M > 0$  is some constant we fix later. The line  $B_j$  has the equation  $y = -x + cj/n + M$ . Notice that the lines  $A_0, \dots, A_{n-1}$  are parallel and equally spaced; so are the lines  $B_j$ , though they have a different spacing. The intersection point of  $A_i, B_j$  is the point  $(x_{ij}, y_{ij})$  where,

$$x_{ij} = \frac{c(j-i)}{2n} - \frac{ci}{2n^2}, \quad y_{ij} = \frac{c(j+i)}{2n} + \frac{ci}{2n^2} + M.$$

All of the  $x_{ij}$  lie within  $[-c, c]$ . Moreover, all the  $x_{ij}$  are distinct. We can number the point  $(x_{ij}, y_{ij})$  by  $(i, j)$  (see Figure 2). The points  $(x_{ij}, y_{ij})$  for a fixed value of  $j - i$  lie on a line with large negative slope.

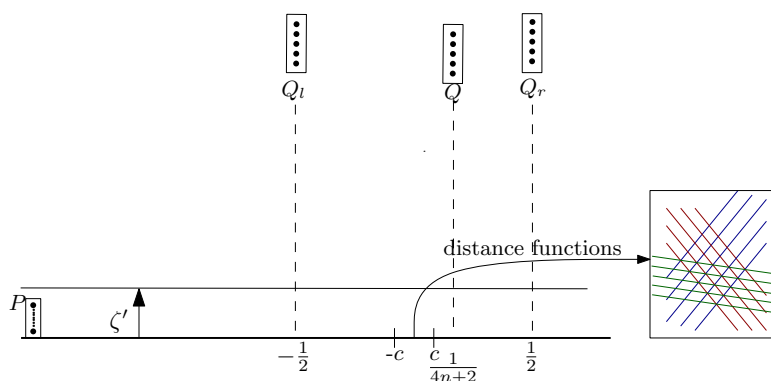
It turns out that the points for the same value of  $(i + j)$  for  $0 \leq i + j \leq n - 1$  also lie on lines with small negative slope. These are defined by lines  $C_k$  for  $k = 0, 1, \dots, n - 1$  where  $C_k$  is defined by the equation:  $y = \frac{-1}{2n+1}x + kc\frac{n+1}{n(2n+1)} + M$ . It can be verified easily that  $C_k$  passes through all the intersection points  $(x_{ij}, y_{ij})$  with  $i + j = k$ . See Figure 2.

We now slightly push down the lines  $C_k$  to  $C'_k$  where  $C'_k$  is defined by the equation  $y = \frac{-1}{2n+1}x + kc\frac{n+1}{n(2n+1)} + M - c'$  where  $c' = O(c/n^3)$ . Notice that for each intersection point  $(i, j)$  there is an interval on the line  $C'_k$  cut off by the lines  $A_i, B_j$  - this interval is also of length  $O(c/n^3)$ . Consider the projection of this interval onto the  $x$ -axis and denote it by  $[x_{ij}^l, x_{ij}^h]$ . This interval contains  $x_{ij}$ , see Figure 3. It can be verified that the  $x_{ij}$  are all separated by at least  $c/2n^2$ , as such the intervals corresponding to all the  $x_{ij}$  are disjoint if  $c'$  is chosen  $O(c/n^3)$ . This property is crucial to us. This grid structure defined by the lines  $A_i, B_j, C'_k$  is what we need for the remainder of the proof, and the crucial properties are the following:

- (i) the intervals for each  $x_{ij}$  where  $0 \leq i + j \leq n - 1$ , i.e.,  $[x_{ij}^l, x_{ij}^h]$  are all disjoint, and,
- (ii) for  $x \in [x_{ij}^l, x_{ij}^h]$  there are  $k$  lines strictly below  $C'_k$  where  $k = i + j$ , while if  $x$  is not in such an interval then there are at least  $k + 1$  lines among the  $A_i, B_j$  strictly below  $C'_k$ .

We want the following correspondence between the distance functions  $F_i, G_j, H_k$  and the set of lines  $A_i, B_j, C'_k$ :  $F_i \leftrightarrow A_i, G_j \leftrightarrow B_j, H_k \leftrightarrow C'_k$ . Intuitively we assume that the distance functions look like the lines as per the correspondence. The rest of the proof uses precisely the above two properties of the lines, and so we continue the proof assuming these two properties of the distance functions. Unfortunately, distance functions are parabolas and we cannot assume that they will behave like the lines. Fortunately, we can show that if  $c$  is small enough, within the interval  $[-c, c]$  the distance functions can be made to behave





■ **Figure 4** The main construction of the 2d worst case example.

precisely like the lines as desired. Since this issue is somewhat of a technicality we defer the formal demonstration of it to the end of the proof. Thus, in what follows, we assume that the distance functions  $F_i, G_j, H_k$  for the points of  $Q_l \cup Q_r \cup Q$  behave like the grid of lines  $A_i, B_j, C'_k$  and have the properties as desired. We continue calling the intersection points of  $F_i, G_j$  as  $x_{ij}$  and the intervals around them as  $[x_{ij}^l, x_{ij}^h]$ , where  $0 \leq i + j \leq n - 1$ .

Recall that as we move from  $-c$  to  $c$  on the  $x$ -axis the graphs of  $F_i, G_j, H_k$  indicate the square of distances from sites of  $Q_l \cup Q_r \cup Q$  to  $(x, 0)$ . We will now describe how to place the sites of  $P$  – this depends on having the sites of  $Q_l \cup Q_r \cup Q$  placed as described. Consider the  $x$ -coordinates of all intersections among  $F_i, G_j, H_k$  for  $x \in [-c, c]$  – this includes all the  $x_{ij}, x_{ij}^l, x_{ij}^h$ , for  $0 \leq i + j \leq n - 1$ . Call this set  $\mathcal{I}$ . By the properties above of the distance functions in  $[-c, c]$  all of these points are distinct. We now want that from a point  $(x, y)$  where  $x \in [-c, c]$  the ordering of distances to sites among  $Q_l \cup Q_r \cup Q$  is the same as that for  $(x, 0)$ . Intuitively, this should be true if  $y$  is small enough, but at intersection points of the distance functions it may not be true. So, consider a number  $\zeta$  much smaller than (say it is  $1/10$  of) the minimum distance between any two of these points in  $\mathcal{I}$ . Consider the points of the  $x$ -axis between  $[-c, c]$  but at least  $\zeta$  away from all these intersection points <sup>1</sup>. We call this set of points  $X$ . Notice that each of  $[x_{ij}^l, x_{ij}^h] \cap X$  is still non-empty. For any such point  $(x, 0) \in X$  we have the following important property: if  $(x, 0) \notin X \cap [x_{ij}^l, x_{ij}^h]$  where  $i + j = k$  then at least  $k + 1$  among the  $Q_l \cup Q_r$  lie strictly closer to it than  $(e_k, f_k)$ , otherwise, if  $(x, 0) \in X \cap [x_{ij}^l, x_{ij}^h]$  then only  $k$  of them are strictly closer. Moreover, there is a number  $\zeta'$  depending on  $\zeta$  such that if we move to a point  $(x, y)$  where  $x \in X$ , and,  $-\zeta' \leq y \leq \zeta'$  this property is still true. This number  $\zeta' > 0$  will be used below for placement of the sites in  $P$ .

The sites of  $P$  are arranged according to the lower bound example for  $d = 1$  on a vertical line parallel to the  $y$ -axis to the left of the origin, such that the sites in  $Q_l \cup Q_r \cup Q$  are closer for  $x \in [-c, c]$ , see Figure 4 for what the placement of sites looks like. (The assigned  $x$ -coordinates in Figure 4 are derived later on, when we discuss how the grid lines are approximated by distance functions.) However, they have been “squished” to all lie very close to the  $x$ -axis; the 1-d lower bound construction works for any squishing. In particular, the squishing ensures all points lie within the strip of width  $\zeta'$  around the  $x$ -axis. All bisectors

<sup>1</sup> The number  $\zeta$  is not really significant for this proof, rather a technicality. If we are  $\zeta$  away from the intersection points, the distance functions have some gap amongst themselves. Then, as we move away from the  $x$ -axis to  $(x, y)$  for small  $y$ , the relative ordering of distances to sites in  $Q_l \cup Q_r \cup Q$  from  $(x, y)$  remains same as that for  $(x, 0)$ .

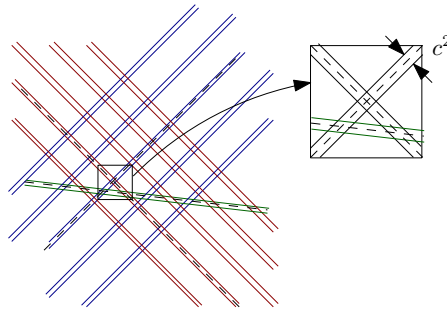
of the points among  $P$ , which are now lines parallel to the  $x$ -axis, lie within the strip as well (these bisector lines are important to the 1-d construction, being the places where the MLNN among the sites of  $P$  changes).

We now indicate the probabilities assigned to the sites. Consider  $\varepsilon, \delta, \pi_0, \dots, \pi_{n-1}$ , and,  $\alpha_0, \dots, \alpha_{n-1}$  from Lemma 10. We let all sites in  $Q_l \cup Q_r$  have associated probability  $\delta$ . With the site  $(e_k, f_k) \in Q$  we associate  $\pi_k$ , for  $k \in [n]$ , and to the sites in  $P$  we assign the probabilities  $\alpha_i$  and  $\varepsilon$ , for  $i \in [n]$ , see the 1-d construction above. We fix our attention on  $x \in X$ . We have the property that if  $x$  is within  $X \cap [x_{ij}^l, x_{ij}^h]$ , there are  $k$  sites among  $Q_l \cup Q_r$  closer to  $(x, 0)$  than  $(e_k, f_k)$  for  $k = i + j$ , otherwise there are at least  $k + 1$  of them closer than  $(e_k, f_k)$ . By the properties of the numbers from Lemma 10, and the properties of distances to sites in  $Q_l \cup Q_r \cup Q$  we have the following: If  $x \in X \cap [x_{ij}^l, x_{ij}^h]$ , then the probability of  $(e_k, f_k)$  being a MLNN where  $k = i + j$  is precisely  $\pi_0 > \alpha_0$ . Otherwise, the probability of  $(e_k, f_k)$  being a MLNN is at most  $\pi_0(1 - \delta)$ . Moreover, for a site in  $Q_l \cup Q_r$  the probability is  $\delta < \pi_0(1 - \delta)$ . Recall from the 1-d construction that the probabilities of the MLNN are of the form  $\alpha_0(1 - \varepsilon)^j$  for  $0 \leq j < n$ . Therefore, when  $x \in X \cap [x_{ij}^l, x_{ij}^h]$  then  $(e_k, f_k)$  is the MLNN where  $k = i + j$  since  $\pi_0 > \alpha_0$ . If  $x \in X$  but not within any of the intervals  $[x_{ij}^l, x_{ij}^h]$  then the MLNN will be a site in  $P$ , since  $\pi_0(1 - \delta) < \alpha_0(1 - \varepsilon)^n$  and  $\alpha_0(1 - \varepsilon)^n$  is a lower bound on the minimum possible probability for a site in  $P$  to be the MLNN.

We now analyze the complexity of the resulting MLVD. Note that  $P$  has  $n^2$  bisectors at which the MLNN in  $P$  changes. Furthermore, a site  $(e_k, f_k) \in Q$  becomes the MLNN briefly at  $k + 1$  different ‘‘intervals’’  $X \cap [x_{ij}^l, x_{ij}^h]$  (we will call this a pseudo-interval, since this set is not actually an interval but lies within  $[x_{ij}^l, x_{ij}^h]$  and all such pseudo-intervals are disjoint from each other by construction), where  $i + j = k$  as argued above. Therefore the most likely nearest neighbor is in  $Q$  for  $n(n + 1)/2$  different pseudo-intervals, corresponding to the intersection points from the lower half of the grid, see Figure 2. Moreover, between any two of the pseudo-intervals the MLNN lies in  $P$ . Since there are  $n^2$  bisectors of  $P$ , and each of these bisectors causes a vertex of the MLVD for each of the  $n(n + 1)/2$  pseudo-intervals, the total complexity of the MLVD is  $\Omega(n^4)$ .

For  $d > 2$ , consider the same construction, but now with  $P$  replaced by the lower bound construction for  $d - 1$ . The sites in  $Q_l \cup Q_r \cup Q$  can be placed in the  $x_1x_2$  plane and we reason with the  $[-c, c]$  interval on the  $x_1$ -axis now. In this case, the strips corresponding to the MLVD for the sites in  $P$ , will be replaced by tubes (or higher dimensional versions of it), which may lie all ‘‘around’’ the  $x_1$ -axis. However, moving to a point close to  $x_1$ -axis in any direction orthogonal to it, will still preserve the ordering of distances as before to sites in  $Q$  (away from intersection points as before), i.e., if we are within a  $\zeta'$  tube of the  $x_1$  axis within the set  $X$  then we are in a similar situation as in the 2-d case. We will need a more general version of Lemma 10, but the crucial point to observe is that the possible probability values for a site in  $P$  to be MLNN will lie in a very small interval depending on  $\delta$ , and then one can choose a  $\pi_0$  and  $\delta$  such that  $(\pi_0(1 - \delta), \pi_0)$  entirely contains this interval. The rest of the construction details are tedious but work similar to the 2-d case. This leads to a total complexity of  $\Omega(n^{2d})$  for the MLVD, as by assumption the complexity of the MLVD of  $P$  was  $\Omega(n^{2d-2})$ . The number of points involved in the construction increase by  $3n$  for each dimension and start with  $2n$ , thus only  $(3d - 1)n$  points are involved.

In order to finish the proof we still need to show how to approximate the lines by actual distance functions. Consider a line  $y = Px + Q$  where  $Q \geq P^2/4$ . Consider the point  $(\beta, \gamma) = (-P/2, \sqrt{Q - P^2/4})$ . It can be verified that the distance function  $f_{(\beta, \gamma)}$  is  $x^2 + Px + Q$ . As such if  $x \in [-c, c]$  we have that  $0 \leq f_{(\beta, \gamma)}(x) - (Px + Q) \leq c^2$  and if  $c$  is



■ **Figure 5** Grid of strips each of width  $c^2$ . The dotted lines show the actual distance functions trapped in the strip within  $[-c, c]$ . The lower strip line in each case is the earlier  $A_i, B_j$  or  $C'_k$ .

“small” the parabolas are approximately lines. In particular, consider the grid of Figure 2. As we noticed the important intersection points are  $(x_{ij}, y_{ij})$  for  $0 \leq i + j \leq n - 1$ . Moreover, the  $x$ -coordinates of any two intersection points are separated by  $\Omega(c/n^2)$ . The intervals  $[x_{ij}^l, x_{ij}^h]$  are of length  $O(c/n^3)$ . Consider replacing the lines of the grid by thin strips of width  $c^2$ . If  $c = O(1/n^4)$ , then  $c^2 \ll c/n^2, c/n^3$  and the picture looks like the modified picture of the grid, see Figure 5. As the distance functions lie inside the corresponding strips, the intersection points lie inside the “diamonds” that occur at the intersection of the strips. Interestingly, the  $x$  coordinates of the intersection points do not change, as can be verified. Moreover because the width of these strips is  $O(c^2)$  the intersection points will be still distinct and the corresponding intervals will be disjoint as well. The distance functions otherwise behave like lines (i.e., they are continuous and intersect at most once), moreover all the  $F_i$  are “parallel” and equally spaced, as are the  $G_j$  and the  $H_k$ , as such the grid induced by them looks like and has the essential properties of the grid of lines  $A_i, B_j, C'_k$  of Figure 2 and will suffice for the proof. To fix the coordinates, notice that we can choose the constant  $M$  large enough so that  $Q > P^2/4$  for each of the lines involved; in particular  $M = 1$  works. The corresponding points can be computed by the formulas for  $\beta, \gamma$  above and it can be easily seen that the  $x$  coordinates of all points in  $Q_l$  is  $-1/2$ , in  $Q_r$  it is  $1/2$  and it is  $1/(4n + 2)$  for each point of  $Q$ , as shown in Figure 4. This completes the proof. ◀

## References

- 1 A. Abdullah, S. Daruki, and J. M. Phillips. Range counting coresets for uncertain data. In *Proc. 29th Annu. Sympos. Comput. Geom.*, pages 223–232. ACM, 2013.
- 2 P. Afshani, P. K. Agarwal, L. Arge, K. G. Larsen, and J. M. Phillips. (Approximate) uncertain skylines. *Theory of Comput. Syst.*, 52:342–366, 2013.
- 3 P. K. Agarwal, B. Aronov, S. Har-Peled, J. M. Phillips, K. Yi, and W. Zhang. Nearest neighbor searching under uncertainty II. In *Proc. 32nd ACM PODS*, pages 115–126, 2013.
- 4 P. K. Agarwal, S.-W. Cheng, and K. Yi. Range searching on uncertain data. *ACM Trans. on Algorithms*, 8(4):43:1–43:17, 2012.
- 5 P. K. Agarwal, A. Efrat, S. Sankararaman, and W. Zhang. Nearest-neighbor searching under uncertainty. In *Proc. 31st ACM PODS*, pages 225–236. ACM, 2012.
- 6 P. K. Agarwal, S. Har-Peled, S. Suri, H. Yıldız, and W. Zhang. Convex hulls under uncertainty. In *Proc. 22nd ESA*, pages 37–48, 2014.
- 7 C. C. Aggarwal. *Managing and Mining Uncertain Data*. Springer, 2009.
- 8 C. C. Aggarwal and P. S. Yu. A survey of uncertain data algorithms and applications. *IEEE TKDE.*, 21(5):609–623, 2009.

- 9 A. Agrawal, S. Rahul, Y. Li, J. Xue, and R. Janardan. Skyline problems for stochastic points: Algorithms and hardness results. Unpublished manuscript (personal communication), 2015.
- 10 Z. Bai, L. Devroye, H. Hwang, and T. Tsai. Maxima in hypercubes. *Random Structures & Algorithms*, 27(3):290–309, 2005.
- 11 M. de Berg, A. D. Mehrabi, and F. Sheikhi. Separability of imprecise points. In *Proc. 14th SWAT*, pages 146–157. Springer, 2014.
- 12 H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag New York, Inc., 1987.
- 13 M. Fink, J. Hershberger, N. Kumar, and S. Suri. Hyperplane separability and convexity of probabilistic point sets. In *Proc. 32nd Annu. Sympos. Comput. Geom.*, pages 38:1–38:16, 2016.
- 14 S. Har-Peled and B. Raichel. On the complexity of randomly weighted Voronoi diagrams. In *Proc. 30th Annu. Sympos. Comput. Geom.*, pages 232–241, 2014.
- 15 A. Jørgensen, M. Löffler, and J. M. Phillips. Geometric computations on indecisive and uncertain points. *CoRR*, abs/1205.0273, 2012.
- 16 P. Kamousi, T. M. Chan, and S. Suri. Stochastic minimum spanning trees in Euclidean spaces. In *Proc. 27th Annu. Sympos. Comput. Geom.*, pages 65–74, 2011.
- 17 P. Kamousi, T. M. Chan, and S. Suri. Closest pair and the post office problem for stochastic points. *Comput. Geom. Theory Appl.*, 47(2):214–223, 2014.
- 18 H. Kaplan, E. Ramos, and M. Sharir. The overlay of minimization diagrams in a randomized incremental construction. *Discrete Comput. Geom.*, 45(3):371–382, 2011.
- 19 A. I. Kuksa and N. Z. Šor. The method of estimating the number of conditionally optimal trajectories of discrete separable dynamic programming (in russian). *Kibernetika (Kiev)*, 6:37–44, 1972.
- 20 N. Kumar and S. Suri. Containment and evasion in stochastic point data. In *Proc. 12th Latin Am. Th. Infor. Sympos.*, pages 576–589, 2016.
- 21 Y. Li, J. Xue, A. Agrawal, and R. Janardan. On the arrangement of stochastic lines in  $\mathbb{R}^2$ . Unpublished manuscript (personal communication), 2015.
- 22 J. Matoušek. *Lectures on Discrete Geometry*, volume 212 of *Grad. Text in Math*. Springer, 2002. URL: <http://kam.mff.cuni.cz/~matousek/dg.html>.
- 23 K. Agarwal P. and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. North-Holland, 2000. URL: <http://www.cs.duke.edu/~pankaj/papers/arrangement-survey.ps.gz>.
- 24 S. Suri and K. Verbeek. On the most likely voronoi diagram and nearest neighbor searching. In *Proceedings of the 25th Int. Sympos. Alg. and Comp. (ISAAC 2014)*, pages 338–350, 2014.
- 25 J. Xue, Y. Li, and R. Janardan. On the separability of stochastic geometric objects, with applications. In *Proc. 32nd Annu. Sympos. Comput. Geom.*, pages 62:1–62:16, 2016.

# Fréchet Distance Between a Line and Avatar Point Set\*

Aritra Banik<sup>1</sup>, Fahad Panolan<sup>2</sup>, Venkatesh Raman<sup>3</sup>, and Vibha Sahlot<sup>4</sup>

- 1 Indian Institute of Technology, Jodhpur, India  
aritrabanik@gmail.com
- 2 Department of Informatics, University of Bergen, Norway  
fahad.panolan@ii.uib.no
- 3 The Institute of Mathematical Sciences, HBNI, Chennai, India  
vraman@imsc.res.in
- 4 Indian Institute of Technology, Jodhpur, India  
sahlotvibha@gmail.com

---

## Abstract

Fréchet distance is an important geometric measure that captures the distance between two curves or more generally point sets. In this paper, we consider a natural variant of Fréchet distance problem with multiple choice, provide an approximation algorithm and address its parameterized and kernelization complexity. A multiple choice problem consists of a set of color classes  $\mathcal{Q} = \{Q_1, Q_2, \dots, Q_n\}$ , where each class  $Q_i$  consists of a pair of points  $Q_i = \{q_i, \bar{q}_i\}$ . We call a subset  $A \subset \{q_i, \bar{q}_i : 1 \leq i \leq n\}$  conflict free if  $A$  contains at most one point from each color class. The standard objective in multiple choice problem is to select a conflict free subset that optimizes a given function.

Given a line segment  $\ell$  and set  $\mathcal{Q}$  of a pair of points in  $\mathbb{R}^2$ , our objective is to find a conflict free subset that minimizes the Fréchet distance between  $\ell$  and the point set, where the minimum is taken over all possible conflict free subsets. We first show that this problem is NP-hard, and provide a 3-approximation algorithm. Then we develop a simple randomized FPT algorithm which is later derandomized using universal family of sets. We believe that this technique can be of independent interest, and can be used to solve other parameterized multiple choice problems. The randomized algorithm runs in  $O(2^k n \log^2 n)$  time, and the derandomized deterministic algorithm runs in  $O(2^k k^{O(\log k)} n \log^2 n)$  time, where  $k$ , the parameter, is the number of elements in the conflict free subset solution. Finally we present a simple branching algorithm for the problem running in  $O(2^k n^2 \log n)$  time. We also show that the problem is unlikely to have a polynomial sized kernel under standard complexity theoretic assumption.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Fréchet Distance, Avatar Problems, Multiple Choice, FPT

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.32

## 1 Introduction

The Fréchet distance measures similarity between two curves by considering an ordering of the points along the two curves. An intuitive definition of the Fréchet distance is to imagine

---

\* The research of the second author leading to these results received funding from the European Research Council (ERC) via grants Rigorous Theory of Preprocessing, reference 267959.



that a dog and its handler are walking on their respective curves. Both can control their speed but can only go forward. The Fréchet distance of these two curves is the minimum length of any leash necessary for the handler and the dog to move from the starting points of the two curves to their respective endpoints [6].

Eiter and Mannila [14] introduced discrete Fréchet distance. Intuitively, the discrete Fréchet distance replaces the dog and its owner by a pair of frogs that can only reside on any of the  $n$  and  $m$  specific pebbles on the curves  $A$  and  $B$  respectively. These frogs hop from a pebble to the next without backtracking. Formally let  $A = \{a_1, a_2, \dots, a_n\}$  and  $B = \{b_1, b_2, \dots, b_m\}$  be a sequence of points. For any  $r \in \mathbb{R}$  we define the graph  $G_r$  with vertices  $A \times B$  and there exists an edge between  $(a_i, b_j)$  and  $(a_{i+1}, b_j)$  if  $d(a_{i+1}, b_j) < r$  and there exists an edge between  $(a_i, b_j)$  and  $(a_i, b_{j+1})$  if  $d(a_i, b_{j+1}) < r$ . Discrete Fréchet distance between  $A$  and  $B$  is the infimum value of  $r$  such that in  $G_r$  there is a path between  $(a_1, b_1)$  and  $(a_n, b_m)$ .

In this paper we introduce a semi-discrete Fréchet distance which is, given a continuous curve  $S$  and a set of points  $P$ , the minimum length of a leash that simultaneously allows the owner to walk on  $S$  continuously and the frog to have discrete jumps from one point to another in  $P$  without backtracking. Hence the leash is allowed to switch discretely when frog jumps from one point to another. We assume that  $S$  is a line segment. Our main point of consideration is the multiple choice problem in this setting. Here instead of a set of points  $P$ , we are given a set of pair of points  $\mathcal{Q}$  in  $\mathbb{R}^2$  such that at most one point is selected from each pair so that the length of leash needed is minimized.

These problems are motivated by 2D curve fitting and object construction from noisy data which can further be used in computer vision for data comparison and biomolecules structure comparison. Here the “resemblance” corresponds to minimizing the semi-discrete Fréchet distance. For example, given a noisy data with/without multiple choice constraints, we may construct a curve/object resembling the standard curve/object and may find the resemblance parameter (specified by semi-discrete Fréchet distance).

**Related Work.** Fréchet distance problem has been extensively studied in the literature. Alt et al. [3] presented an algorithm to compute the Fréchet distance between two polygonal curves of  $n$  and  $m$  vertices in time  $O(nm \log^2(nm))$ . The discrete Fréchet distance can be computed in  $O(mn)$  time by a straightforward dynamic programming algorithm. Agarwal et al. [1] presented a sub-quadratic algorithm for computing the discrete Fréchet distance between two sequences of points in the plane.

The following problem has been recently addressed by Shahbaz [19]. Given a point set  $S$  and a polygonal curve  $P$  in  $\mathbb{R}^d (d > 2)$ , find a polygonal curve  $Q$ , with its vertices chosen from  $S$ , such that the Fréchet distance between  $P$  and  $Q$  is minimum with the relaxation that not all points in  $S$  need to be chosen, and a point in  $S$  can appear more than once as a vertex in  $Q$ . They show that a curve minimizing the Fréchet distance can be computed in  $O(nk^2 \log(nk))$  time where  $n$  and  $k$  represent the sizes of  $P$  and  $S$  respectively. In a recent paper [9] Consuegra and Narasimhan introduce the concept of Avatar problems that deal with situations where each entity has multiple copies or “avatars” and the solutions are constrained to use exactly one of the avatars. Further study of the problems of same flavor can be found in [5, 4]. An Avatar problem consists of a set of color classes  $\mathcal{Q} = \{Q_1, Q_2, \dots, Q_n\}$ , where each color class  $Q_i$  consists of a pair of points  $Q_i = \{q_i, \bar{q}_i\}$  (in general  $k \geq 2$  points can be in each class). We call a subset  $A \subset \{q_i, \bar{q}_i : 1 \leq i \leq n\}$  conflict free if  $A$  contains at most one point from each color class.

**Problems we address.** We formally define the problems considered in this paper starting with the Semi-discrete Fréchet Distance problem.

## SEMI-DISCRETE FRÉCHET DISTANCE

**Input:** A set of points  $P = \{p_1, p_2, \dots, p_n\}$  and a line segment  $\ell$  in  $\mathbb{R}^2$ .

**Question:** Find a sequence of points  $\lambda^* = \{q_1, q_2 \dots q_k\}$  where  $q_i \in P$ , which minimizes Fréchet distance with  $\ell$  where the minimum is taken over all the sequence of points in  $P$ . We denote the minimum distance by  $d^F(P, \ell)$ .

Next we consider the following problems involving choices.

## CONFLICT-FREE FRÉCHET DISTANCE

**Input:** A set  $\mathcal{Q}$  of pairs of points, and a line segment  $\ell$  in  $\mathbb{R}^2$ .

**Question:** Find a conflict free subset of points  $P^* \subset \bigcup_{i=1}^n Q_i$  which minimizes  $d^F(P^*, \ell)$

The natural decision version of this problem is as follows.

## CONFLICT-FREE FRÉCHET DISTANCE (DECISION VERSION)

**Input:** A set  $\mathcal{Q}$  of pairs of points, a line segment  $\ell$  in  $\mathbb{R}^2$ , and  $d \in \mathbb{R}$ .

**Question:** Is there a conflict free set of points  $P^* \subset \bigcup_{i=1}^n Q_i$  such that  $d^F(P^*, \ell) \leq d$ .

The natural parameterized version of the problem is

## PARAMETERIZED CONFLICT-FREE FRÉCHET DISTANCE

**Parameter:**  $k$

**Input:** A set  $\mathcal{Q}$  of pairs of points, a line segment  $\ell$  in  $\mathbb{R}^2$ ,  $d \in \mathbb{R}$ , and  $k \in \mathbb{N} \cup \{0\}$ .

**Question:** Is there a conflict free subset of points  $P^*$  of cardinality at most  $k$  such that  $d^F(P^*, \ell) \leq d$ .

We also consider parameterized version of “minimum maxGap” introduced in [9]. Here given a set of points  $x_1, \dots, x_n$  on a line,  $maxGap$  is the largest gap between consecutive points in the sorted order. The problem is as follows.

## PARAMETERIZED MINIMUM MAXGAP

**Parameter:**  $k$

**Input:** A set  $\mathcal{Q}$  of pairs of points on a line  $L$ , two points  $p_s$  and  $p_e$  on  $L$ ,  $d \in \mathbb{R}$ , and  $k \in \mathbb{N} \cup \{0\}$ .

**Question:** Is there a conflict free subset of points  $P^*$  of cardinality at most  $k$  between  $p_s$  and  $p_e$  such that the minimum  $maxGap$  of  $P^* \cup \{p_s, p_e\}$  is at most  $d$ .

**Our Results and the organization of the paper.** In Section 2 we prove that CONFLICT-FREE FRÉCHET DISTANCE (DECISION VERSION) is NP-Complete. In Section 3 we show that SEMI-DISCRETE FRÉCHET DISTANCE is solvable in  $O(n \log n)$  time. In Section 4 we provide a constant factor approximation algorithm for CONFLICT-FREE FRÉCHET DISTANCE. In Section 5 we consider the parameterized complexity of the problem, i.e, PARAMETERIZED CONFLICT-FREE FRÉCHET DISTANCE. In parameterized complexity, algorithm runtimes are measured in terms of input length and a parameter, which is expected to be small. More precisely, a parameterized problem is *fixed-parameter tractable (FPT)* if an instance  $(I, k)$  can be solved in time  $f(k) \cdot |I|^{O(1)}$  for some function  $f$ . Another major research field in parameterized complexity is kernelization. A parameterized problem is said to admit a *polynomial kernel* if any instance  $(I, k)$  can be reduced to an equivalent instance  $(I', k')$ , in polynomial time, with  $|I'|$  and  $k'$  bounded by a polynomial in  $k$ . There is also a lower bound framework for kernelization which allows us to rule out the existence of polynomial kernels

for some problems under standard complexity-theoretic assumptions [8, 11, 15]. For more details about parameterized complexity we refer to monographs [12, 10].

We begin with a simple randomized FPT algorithm and provide a method to derandomize the algorithm using universal sets. In Section 5.2 we give another FPT algorithm using branching. Finally in Section 5.3 we show that the problem is unlikely to have a polynomial sized kernel using OR-composition.

## 2 Hardness of Conflict-free Fréchet distance Problem

In this section we show that CONFLICT-FREE FRÉCHET DISTANCE (DECISION VERSION) is NP-complete by giving a reduction from Rainbow covering problem mentioned in [4]. Suppose we are given a set  $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$  where each  $P_i$  contains a pair of intervals  $\{I_i, \bar{I}_i\}$  such that each interval is a finite continuous subset of the  $x$ -axis. A set of intervals  $Q \subseteq \bigcup_{i=1}^n P_i$  is a rainbow, if it contains at most one interval from each interval pair. An interval is said to cover a point if the point lies inside the interval. The formal definition of Rainbow covering problem is as follows.

RAINBOW COVERING

**Input:** A set of pairs of intervals  $\mathcal{P}$  and a set of points  $S = \{s_1, s_2, \dots, s_n\}$  on  $x$ -axis.

**Question:** Does there exist a rainbow  $Q$  such that each point in  $S$  is covered by at least one interval in  $Q$ .

RAINBOW COVERING is known to be NP-complete [4]. We introduce an intermediate problem called RAINBOW LINE COVER and show it NP-complete using a reduction from RAINBOW COVERING. Then we give a reduction from RAINBOW LINE COVER to CONFLICT-FREE FRÉCHET DISTANCE (DECISION VERSION).

RAINBOW LINE COVER

**Input:** Set  $\mathcal{P}' = \{P'_1, P'_2, \dots, P'_m\}$  where each  $P'_i$  contains a pair of left open intervals  $\{I_i, \bar{I}_i\}$  and a line segment on  $x$ -axis,  $\ell^{in} = [x_1, x_2]$ .

**Question:** Is there a rainbow  $Q^{in}$  such that it covers line segment  $\ell^{in}$ .

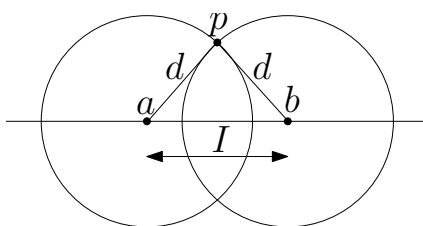
► **Lemma 1.** RAINBOW LINE COVER is NP-hard.

**Proof.** The proof is by a polynomial time reduction from RAINBOW COVERING. Let  $(\mathcal{P}, S)$  be an instance of RAINBOW COVERING. Without loss of generality, let  $s_1, s_2, \dots, s_n$  be the arrangement of points from  $S$  in increasing order on  $x$ -axis according to their  $x$ -coordinates and each interval from  $\mathcal{P}$  covers at least one point in  $S$ . We will create an instance  $(\mathcal{P}', \ell^{in})$  of RAINBOW LINE COVER as follows. Each interval in  $\mathcal{P}$  will be extended and the pairing in the new set  $\mathcal{P}'$  is same as the old one. Now for each interval  $I_j = [a_j, b_j]$  covering  $s_1$ , i.e, the first point in  $S$ , consider the interval formed via extending it by a small distance  $\delta \in \mathbb{R}$  on left such that it is open at the extended point. Denote it as  $I_j^{in} = (a_j - \delta, b_j]$ . For each remaining intervals  $I_i = [a_i, b_i]$ , consider the point  $s$  in  $S$  such that  $s$  is strictly to the left of  $a_i$  and is closest to it. Extend  $I_i$  to the left such that it is open ended at that point to make  $I_i^{in}$ . For example, if  $s = (c, 0)$  then  $I_i^{in} = (c, b_i]$ . Now suppose  $s_1 = (a_1, 0)$  and  $s_n = (a_n, 0)$ , then  $\ell^{in}$  is the line segment on  $x$ -axis is  $[a_1, a_n]$ .

► **Claim 2.** There exists a rainbow from  $\mathcal{P}$  of size  $d$ , covering  $S$ , if and only if there exists a rainbow from  $\mathcal{P}'$  of size  $d$  covering  $\ell^{in}$ .

**Proof.** Let  $Q$  be a rainbow from  $\mathcal{P}$  covering  $S$ . Let  $Q^{in}$  be the set of intervals constructed from  $Q$  in the reduction. We claim that  $Q^{in}$  is a rainbow covering  $\ell^{in}$ . Since  $Q$  is a rainbow,





■ **Figure 1** Reduction from Rainbow line cover problem.

$Q^{in}$  is also also a rainbow. Since all all the points in  $S$  is covered by  $Q$  and each interval of  $Q$  is extended left by non-zero distance,  $S$  is covered by  $Q^{in}$ . Let  $q \notin S$  be a point in  $\ell^{in}$ . Let  $s$  be the point in  $S$  to the right of  $q$  and closest to  $q$ . By construction any interval covering  $s$ , also covers  $q$ . Hence  $Q^{in}$  covers  $\ell^{in}$ .

Similarly if there is rainbow  $Q^{in}$  covering  $\ell^{in}$  then there exists a rainbow  $Q$  such that it covers  $S$ . Here  $Q$  will be the set of intervals that were used to construct intervals in  $Q^{in}$ . Since  $Q^{in}$  is a rainbow,  $Q$  is also a rainbow. Since  $Q^{in}$  covers  $S \subseteq \ell^{in}$  and the intervals in  $Q^{in}$  are obtained by extending openly to nearest left point from  $S$ ,  $Q$  covers  $S$ . ◀

This completes the proof. ◀

► **Theorem 3.** CONFLICT-FREE FRÉCHET DISTANCE (DECISION VERSION) is NP-complete.

**Proof.** Given a sequence of at most  $n$  points as witness, we can check in polynomial time whether the the points in the sequence is conflict free and Fréchet distance is at most  $d$ , thus the problem is in NP.

To prove NP-hardness we give a polynomial time reduction from RAINBOW LINE COVER. Let  $(\mathcal{P}', \ell)$  be an instance of RAINBOW LINE COVER, where  $|\mathcal{P}'| = n$ . From  $\mathcal{P}'$  we create a set of pairs of points  $\mathcal{Q}$ . For each pair  $P_i \in \mathcal{P}'$  we create a pair of points  $Q_i \in \mathcal{Q}$ . To do this, for each interval  $I = (a, b] \in P_i$  (similarly  $\bar{I}$ ) we create a point  $p$  (similarly  $\bar{p}$ ) as follows. If  $a < x_1$ , then prune the interval such that  $a = x_1$ . Similarly if  $b > x_2$  then make  $b = x_2$ . Let the length of an interval  $I_i = (a_i, b_i]$  be  $b_i - a_i$  and  $len$  be the largest length among the length of all the intervals in  $\mathcal{P}'$ . Define  $d = len + 1$ . Now for each interval  $I = (a, b] \in \bigcup_{P \in \mathcal{P}'} P$ , we create a point  $p$ . Consider two disks  $D(a)$  and  $D(b)$  of radius  $d$  centred at  $(a, 0)$  and  $(b, 0)$  respectively. Let  $p$  be the intersection point above  $x$ -axis between  $D(a)$  and  $D(b)$ .

The set  $\mathcal{Q}$  is the set of pairs of points created as above, one for each  $P \in \mathcal{P}'$ . The pair  $(\mathcal{Q}, \ell)$  is the output of the reduction. Clearly, the reduction takes polynomial time.

► **Claim 4.** There is a rainbow covering for  $(\mathcal{P}', \ell)$  if and only if the conflict free Fréchet distance between  $\mathcal{Q}$  and  $\ell$  is at most  $d$ .

**Proof.** Consider a rainbow covering  $R$  for  $(\mathcal{P}', \ell)$ . Now consider the set  $S$  constructed from intervals in rainbow  $R$ . Since  $R$  is a rainbow,  $S$  is conflict free. Also as the intervals were covering  $\ell$ , each point on  $\ell$  has a point in  $S$  which is at maximum distance of  $d$ . Hence the Fréchet distance between  $\mathcal{Q}$  and  $\ell$  is at most  $d$ .

For the reverse direction, assume the Fréchet distance between  $\mathcal{Q}$  and  $\ell$  is  $d$ . That is, there exists a sequence  $T = (p_1, p_2, \dots, p_k)$  of conflict free points from  $\mathcal{Q}$ , which should be traversed in order to attain Fréchet distance  $d$ . Here  $p_i$  is point in  $\mathcal{Q}$  for all  $1 \leq i \leq k$ . Assume  $\ell = [x_1, x_2]$  and  $y_i = [a_i, a_{i+1}]$  be the interval on  $\ell$  nearest to point  $p_i$  for all points  $p_i \in T$ . We have  $a_1 = x_1$  and  $a_{k+1} = x_2$ . Also for all points  $z \in y_i, d(z, p_i) \leq d$  where  $1 \leq i \leq k$ . Thus by construction, corresponding to each  $y_i$ , we have an interval  $I_i \in \mathcal{P}'$  such

that  $y_i \subseteq I_i$ . Let the set of such intervals be  $R$ . Since  $T$  is conflict free,  $R$  is a rainbow. Also as Fréchet distance is  $d$ ,  $\ell = \bigcup_{i=1}^k y_i$ . Hence  $R$  also covers  $\ell$ . Therefore  $R$  is a rainbow covering of  $\ell$ . ◀

This completes the proof of the theorem. ◀

### 3 Polynomial Algorithm for Semi-discrete Fréchet distance problem

In this section we first prove that SEMI-DISCRETE FRÉCHET DISTANCE problem can be solved in  $O(n \log n)$  time. Without loss of generality, assume that the line segment  $\ell$  coincides with the  $X$ -axis and has end points  $(x_1, 0)$  and  $(x_2, 0)$ . Take any point  $p_i \in P$  where  $p_i = (a_i, b_i)$  and let  $x$  be a variable depicting the position of a point on line segment  $\ell$  with  $x_1 \leq x \leq x_2$ . Then the function  $f_i(x)$  representing the distance between the point  $p_i$  and  $x$  is  $f_{p_i}(x) = \sqrt{(x - a_i)^2 + b_i^2}$ .

For each point  $p_i \in P$  we can find out the function  $f_i(x)$ , where each such function represents one sided hyperbola lying above the  $X$ -axis and in interval between  $x_1$  and  $x_2$ . Let the lower envelop of such functions defined in the domain  $[x_1, x_2]$  be  $\Gamma(P)$ . Let  $d^*$  be the maximum perpendicular distance between  $\Gamma(P)$  and  $\ell$ . Then we can see that

► **Observation 5.**  $d^*$  is the minimum Fréchet distance between  $\ell$  and  $P$ .

Note that two hyperbolas will intersect at at most one point. To see this, note that solving the two equations  $f_{p_i}(x) = \sqrt{(x - a_i)^2 + b_i^2}$  and  $f_{p_j}(x) = \sqrt{(x - a_j)^2 + b_j^2}$  gives only one solution. Thus each hyperbola can appear in the lower envelop at most once.

Before proceeding further let us have a look at Davenport–Schinzel sequence. Davenport–Schinzel sequences were introduced by H. Davenport and A. Schinzel in the 1960s.

► **Definition 6.** For two positive integers  $n$  and  $s$ , a finite sequence  $U = \langle u_1, u_2, u_3, \dots, u_m \rangle$  is said to be a Davenport–Schinzel sequence of order  $s$  (denoted as DS( $n, s$ )-sequence) if it satisfies the following properties:

1.  $1 \leq u_i \leq n$  for each  $i \leq m$ .
2.  $u_i \neq u_{i+1}$  for each  $i < m$ .
3. If  $x$  and  $y$  are two distinct values in the sequence  $U$ , then  $U$  does not contain a subsequence  $\dots x \dots y \dots x \dots y \dots$  consisting of  $s + 2$  values alternating between  $x$  and  $y$ .

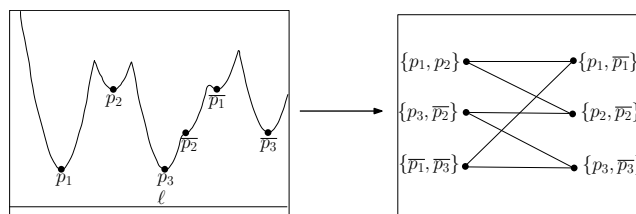
► **Theorem 7** ([17, 7, 2]). *The lower envelope of a set  $\mathcal{F}$  of  $n$  continuous, totally defined, univariate functions, each pair of whose graphs intersects in at most  $s$  points, can be constructed in an appropriate model of computation, in  $O(\lambda_s(n) \log n)$  time where  $\lambda_s(n)$  is the Davenport–Schinzel sequence of order  $s$  including  $n$  distinct values.*

Since  $\lambda_1(n) = n$ , by substituting  $s = 1$  in Theorem 7, we get,

► **Theorem 8.** SEMI-DISCRETE FRÉCHET DISTANCE problem can be solved in  $O(n \log n)$  time.

### 4 Approximation algorithm for Conflict-free Fréchet distance problem

In this section we present an approximation algorithm for CONFLICT-FREE FRÉCHET DISTANCE. Let us first define some terminology. As before, assume that the line segment  $\ell$  coincides with the  $X$ -axis and has end points  $(x_1, 0)$  and  $(x_2, 0)$ . For any point set  $A$ , denote Semi-discrete Fréchet distance between  $A$  and line-segment  $\ell$  by  $d^F(A, \ell)$ . Also let  $\Gamma(A)$  be



■ **Figure 2** Creating the bipartite graph from the lower envelope.

the lower envelope of the functions  $f_{p_i}(x) = \sqrt{(x-a)^2 + b^2}$  for all  $p_i = (a, b) \in A$  where  $x_1 \leq x \leq x_2$ . Now let us start our discussion with the following observation about the Semi-discrete Fréchet distance.

► **Observation 9.** For any set of points  $A$  and  $B$  where  $A \subseteq B$ ,  $d^F(A, \ell) \geq d^F(B, \ell)$ .

**Proof.** Let  $C \subseteq A$  be the set of points that achieves  $d^F(A, \ell) = d$ . Since  $A \subseteq B$ , we have that  $C \subseteq B$  and hence  $d^F(B, \ell) \leq d$ . ◀

Let  $(Q = \{Q_1, Q_2 \dots Q_n\}, \ell)$  be the input instance of CONFLICT-FREE FRÉCHET DISTANCE, where  $Q_i = \{q_i, \bar{q}_i\}$ . Let  $Q = \bigcup_{i=1}^n Q_i$ . By Theorem 7 we can find  $d^F(Q, \ell)$  in  $O(n \log n)$  time. Among all conflict free subsets of  $Q$ , assume  $P^{opt}$  is a subset that minimizes the Semi-discrete Fréchet distance and let  $d^{opt} = d^F(P^{opt}, \ell)$ . If  $\Gamma(Q)$  contains at most one of  $f_{q_i}$  or  $f_{\bar{q}_i}$  for each  $Q_i = \{q_i, \bar{q}_i\}$ , then  $d^{opt} = d^F(Q, \ell)$ . As  $P^{opt} \subseteq Q$ , from Observation 9 we have following lemma.

► **Lemma 10.**  $d^{opt} \geq d^F(Q, \ell)$ .

Suppose the set of points for which the corresponding  $f_{q_i}(x)$  are in  $\Gamma(Q)$  be  $P'$ . Observe that if  $P'$  does not contain points from the same pair, then  $d^F(P', \ell)$  is the conflict free Semi-discrete Fréchet distance and we have  $d^{opt} = d^F(Q, \ell) = d^F(P', \ell)$ . If not, then our objective is to choose a conflict free subset  $P''$  of  $P'$  such that  $d^F(P'', \ell) \leq 3d^F(P', \ell)$ . First, for all the points  $q_i \in P'$  such that  $\bar{q}_i \notin P'$ , we include  $q_i$  in  $P''$ . For the rest of the points, let  $P_{pair} = \{p_1, p_2, \dots p_{2k}\}$  be the sorted order of points along  $x$ -axis where each  $p_i = q_j$  or  $\bar{q}_j$  for some  $j$ . Now from  $P_{pair}$ , we create bags  $B_1, B_2, \dots, B_k$  where  $B_i = \{p_{2i-1}, p_{2i}\}$ . We construct a bipartite graph  $G = (U, V, E)$  where  $U = \{B_1, B_2, \dots, B_k\}$  and  $V$  is set of all  $k$  pairs  $Q_i = \{q_i, \bar{q}_i\}$  such that both  $q_i$  and  $\bar{q}_i$  are in  $P_{pair}$ . We add an edge  $e_{ij} = (B_i, Q_j)$ , if  $B_i \cap Q_j \neq \emptyset$ . For an example, see Figure 2.

Now we have the following lemma.

► **Lemma 11.**  $G = (U, V, E)$  contains a perfect matching  $M$ .

**Proof.** Each vertex in  $U$  and  $V$  has degree at least 1 and at most 2. Also if vertex  $B_i$  in  $U$  has degree one, then the vertex  $Q_j$  to which it is connected in  $V$  also has degree one (as it implies that both  $B_i = Q_j = \{q_i, \bar{q}_i\}$ ). Thus every subset  $W$  of  $U$  has a set of neighbours  $N_G(W)$  such that  $|W| \leq |N_G(W)|$  (here the neighbours of  $W$  is the set of vertices in  $V$  to which vertices in  $W$  are connected). Hence by Hall's marriage theorem [16],  $G$  has a perfect matching  $M$ . ◀

Let  $M$  be a perfect matching in  $G$ . Now for each edge  $(B_i, Q_j)$  selected in matching  $M$ , if  $|B_i \cap Q_j| = 1$  then include  $|B_i \cap Q_j|$  in  $P''$ , else if  $|B_i \cap Q_j| = 2$  then we include one arbitrary point of  $B_i \cap Q_j$  in  $P''$ . Observe that from each pair of points in  $P_{pair}$ , only one point is selected. Thus  $P''$  is conflict free. Now we have following lemma.

► **Lemma 12.**  $d^F(P'', \ell) \leq 3d^F(Q, \ell)$ .

**Proof.** Since  $d^F(Q, \ell) = d^F(P', \ell)$ , it is enough to show that  $d^F(P'', \ell) \leq 3d^F(P', \ell)$ . Let  $\pi$  be the sorted order of points in  $P'$  along  $x$ -axis. We first prove the following claim.

► **Claim 13.** *For any point  $s \in P'$ , at least one among  $s$ , its predecessor in  $\pi$  and its successor in  $\pi$ , is in  $P''$ .*

**Proof.** We claim that (i) no three consecutive points from  $\pi$  can be in  $P' \setminus P''$ . For any three consecutive points  $q_1, q_2, q_3$ , either one of them does not belong to  $P_{pair}$  and thus belongs to  $P''$  or one among  $\{q_1, q_2\}$  and  $\{q_2, q_3\}$  belongs to  $P_{pair}$ . From the construction of  $P''$ , we include one among  $q_1, q_2, q_3$ , in  $P''$ . Now we claim that (ii) at least one among the first two points in  $\pi$  is in  $P''$ . Let  $s_1$  and  $s_2$  be the first two points in  $\pi$ . If  $\{s_1, s_2\} \not\subseteq P_{pair}$ , then  $P'' \cap \{s_1, s_2\} \neq \emptyset$ . Otherwise  $B_1 = \{s_1, s_2\}$  and by the construction of  $P''$ , we have that  $P'' \cap \{s_1, s_2\} \neq \emptyset$ . Similarly we can prove that (iii) at least one among the last two points in  $\pi$  is in  $P''$ .

The claim follows from the statements (i),(ii) and (iii). ◀

Let  $d = d^F(P', \ell)$ . Now we prove that  $d^F(P'', \ell) \leq 3d$ . Towards that it is enough to prove that for any point on  $\ell$ , there is a point in  $P''$ , which is at a distance at most  $3d$ . For any two points  $x, y$ , we use  $d(x, y)$  to denote the distance between  $x$  and  $y$ . Let  $z$  be a point in  $\ell$ . Since  $d = d^F(P', \ell)$ , there is a point  $s$  in  $P'$  such that  $d(z, s) \leq d$ . Now we show that there is a point  $s' \in P''$  such that  $d(z, s') \leq 3d$ . If  $s \in P''$ , then we set  $s' = s$ . Otherwise, by Claim 13, either its successor or its predecessor in  $\pi$  belongs to  $P''$ . Let  $s'$  be a point in  $P''$  which is either successor of  $s$  or predecessor of  $s$ . Since the  $d = d^F(P', \ell)$ , there is a point  $t$  on  $\ell$  such that  $d(t, s) \leq d$  and  $d(t, s') \leq d$ . Now we have that  $d(z, s) \leq d$ ,  $d(s, t) \leq d$  and  $d(t, s') \leq d$ . Hence by triangular inequality, we get  $d(z, s') \leq 3d$ . This completes the proof of the lemma. ◀

► **Theorem 14.** *There is a 3-approximation algorithm for CONFLICT-FREE FRÉCHET DISTANCE.*

## 5 Fixed Parameter Tractable Algorithms

Here we give two FPT algorithms for Parameterized Conflict-free Fréchet distance Problem. The first algorithm is based on randomization and the second is based on branching.

### 5.1 Randomized algorithm

We give a randomized FPT algorithm which succeeds with a constant success probability. It uses the following problem for which there is a simple greedy algorithm running in time  $O(n \log n)$ ; the algorithm is very similar to that of the INTERVAL POINT COVER [13].

INTERVAL LINE COVER

**Input:** A line segment  $\ell$  and a set  $Q$  of  $n$  intervals on  $\ell$ .

**Question:** Find a minimum cardinality subset  $Q' \subseteq Q$  such that the intervals in  $Q'$  cover all the points in the line segment  $\ell$ .

► **Theorem 15.** *There is a randomized algorithm for PARAMETERIZED CONFLICT-FREE FRÉCHET DISTANCE running in time  $O(2^k n \log n)$  which outputs NO for all NO-instances and outputs YES for all YES-instances with constant probability.*

**Proof.** Let  $|\mathcal{Q}| = n$ . The algorithm work as follows. It creates a set  $S$  of  $n$  points through the following random process. For each  $\{q_i, \bar{q}_i\} \in \mathcal{Q}$ , it uniformly at random picks one point from  $\{q_i, \bar{q}_i\}$  and adds to the set  $S$ . Then for each point  $p \in S$ , the algorithm then computes an interval on  $\ell$  as follows. Draw a circle  $C_p$  of radius  $d$  with  $p$  as the centre. The interval  $[a_p, b_p]$  on  $\ell$  is the interval on  $\ell$  covered by the circle  $C_p$ . Now run the  $O(n \log n)$  algorithm for INTERVAL LINE COVER on instance  $(\ell, \{[a_p, b_p] \mid p \in S\})$ . for the problem If this algorithm returns a solution of size at most  $k$ , then our algorithm outputs YES.

Now we show that if the input instance is an YES instance, then our algorithm outputs YES with probability  $\frac{1}{2^k}$ . Let  $P^*$  be a conflict free subset of points of cardinality  $k$  such that  $d^F(P^*, \ell) \leq d$ . Notice that for each  $p_i \in P^*$ , there is point  $\bar{p}_i \notin P^*$  such that  $\{p_i, \bar{p}_i\} \in \mathcal{Q}$  and with probability  $1/2$  we have added  $p_i$  to  $S$ . This implies that  $\Pr(S = P^*) = \frac{1}{2^k}$ . Since each point on  $\ell$  is at a distance at most  $d$  to some point  $P^*$ , when  $S = P^*$ , the algorithm of INTERVAL LINE COVER outputs YES Since  $\Pr(S = P^*) = \frac{1}{2^k}$  our algorithm output YES with probability at least  $\frac{1}{2^k}$ . Suppose input is a NO-instance. Then for each conflict free point set  $P^*$  of size at most  $k$ ,  $d^F(P^*, \ell) > d$ . Also note that the set  $S$  we constructed is a conflict free set. Since  $d^F(P^*, \ell) > d$ , we need more than  $k$  intervals from  $\{[a_p, b_p] \mid p \in S\}$  to cover  $\ell$ . This implies that the algorithm of INTERVAL LINE COVER will return a set of size more than  $k$ , and so our algorithm will output NO.

We can boost the success probability to a constant by running our algorithm  $2^k$  times. For an YES instance the algorithm will fail in all  $2^k$  run is at most  $(1 - \frac{1}{2^k})^{2^k} \leq \frac{1}{e}$ . Since we are running the algorithm of INTERVAL LINE COVER  $2^k$  time, the running time mentioned in the theorem follows.  $\blacktriangleleft$

## Derandomization

Here, we define matching universal sets. Then we give a derandomization of algorithm for the problem. First we define some notations. For  $n \in \mathbb{N}$ , let  $[n] = \{1, \dots, n\}$ . For a set  $U$ ,  $\binom{U}{k}$  denotes the family of subsets of  $U$ , where each subset is of size exactly  $k$ .

**Matching universal sets for a family of disjoint pairs.** Here we define a restricted version of universal sets (defined below) which we call matching universal sets and it is defined for a family of disjoint pairs. We give an efficient construction of these objects by reducing to universal sets. We use it to derandomize our algorithm given in the section. We believe that these objects will add to the list of tools used to derandomize algorithms and will be of independent interest.

► **Definition 16** ( $(n, k)$ -universal sets [18]). Let  $U$  be a set of size  $n$ . A family of subsets  $\mathcal{F}$  of  $A$  is called  $(n, k)$ -universal sets for  $U$ , if for any  $A, B \subseteq U$  such that  $A \cap B = \emptyset, |A \cup B| = k$ , there is a set  $F \in \mathcal{F}$  such that  $A \subseteq F$  and  $F \cap B = \emptyset$

► **Lemma 17** ([18]). *There is a deterministic algorithm which constructs an  $(n, k)$ -universal family of sets of cardinality  $2^k k^{O(\log k)} \log n$  in time  $2^k k^{O(\log k)} n \log n$ .*

► **Definition 18.** Let  $U = \{a_i, b_i \mid i \in [n]\}$  be a  $2n$  sized set and  $\mathcal{S} = \{\{a_i, b_i\} \mid i \in [n]\}$  be a family of pairwise disjoint subsets of  $U$ . A family of subsets  $\mathcal{F}$  of  $U$  is called an  $(n, k)$ -matching universal family for  $\mathcal{S}$ , if for each  $I \in \binom{[n]}{k}$ , and  $S \in \binom{U}{k}$  such that  $|S \cap \{a_j, b_j\}| = 1$  for all  $j \in I$ , we have a set  $F \in \mathcal{F}$  such that  $S \subseteq F$  and  $F \cap (\{a_j, b_j \mid j \in I\} \setminus S) = \emptyset$ .

Now we use Lemma 17, to get an efficient construction of  $(n, k)$ -matching universal sets.

► **Theorem 19.** *Given a  $2n$  sized set  $U = \{a_i, b_i \mid i \in [n]\}$  and a family  $\mathcal{S} = \{\{a_i, b_i\} \mid i \in [n]\}$  of pairwise disjoint subsets of  $U$ , there is a deterministic algorithm which constructs an  $(n, k)$ -matching universal family of cardinality  $2^k k^{O(\log k)} n \log n$  in time  $2^k k^{O(\log k)} n \log n$ .*

**Proof.** Let  $U' = \{e_1, \dots, e_n\}$  be a set of size  $n$ , where each  $e_i$  represents the set  $\{a_i, b_i\}$ . Now our algorithm first constructs an  $(n, k)$ -universal family  $\mathcal{F}'$  for the set  $U'$  using Lemma 17. Now the algorithm constructs an  $(n, k)$ -matching universal sets  $\mathcal{F}$  for  $\mathcal{S}$  from the family  $\mathcal{F}'$  as follows. For each set  $F' \in \mathcal{F}'$ , it creates a set  $F \subseteq U$  of size  $n$  and adds to  $\mathcal{F}$ : for each  $e_i \in U'$ , if  $e_i \in F'$ , then it adds  $a_i$  to  $F$ , otherwise it adds  $b_i$  to  $F$ .

Notice that  $|\mathcal{F}| = |\mathcal{F}'|$ , and hence the cardinality of  $(n, k)$ -matching universal family mentioned in the theorem follows. Since the algorithm mentioned in Lemma 17 takes time  $2^k k^{O(\log k)} n \log n$  and construction of  $F$  from  $F'$  takes time  $O(n)$ , the running time of our algorithm is  $2^k k^{O(\log k)} n \log n$ .

Now we show that  $\mathcal{F}$  is indeed an  $(n, k)$ -matching universal family for  $\mathcal{S}$ . Consider a set  $I \in \binom{[n]}{k}$  and  $S \in \binom{U}{k}$  such that  $|S \cap \{a_j, b_j\}| = 1$  for all  $j \in I$ . Let  $A' = S \cap \{a_j \mid j \in I\}$ ,  $B' = \{a_j \mid j \in I\} \setminus A'$  and  $C = \{b_j \mid a_j \in B'\}$ . Notice that  $S = A' \cup C$ ,  $A' \cap B' = \emptyset$  and since  $|I| = k$ , we have that  $|A' \cup B'| = k$ . Let  $A = \{e_j \mid a_j \in A'\}$  and  $B = \{e_j \mid a_j \in B'\}$ . Since  $A' \cap B' = \emptyset$  and  $|A' \cup B'| = k$  we have that  $A \cap B = \emptyset$  and  $|A \cup B| = k$ . By the definition of  $(n, k)$ -universal family, we know that there is a set  $F' \in \mathcal{F}'$  such that  $A \subseteq F'$  and  $F' \cap B = \emptyset$ . Now consider the set  $F$  created corresponding to  $F'$ . Since for each  $e_j \in A$ ,  $e_j \in F'$ , we have that  $a_j \in F$ . Since for each  $e_{j'} \in B$ ,  $e_{j'} \notin F'$ , we have that  $b_{j'} \in F$ . This implies that  $A' \subseteq F$  and  $C \subseteq F$ , and hence  $A \cup C = S \subseteq F$ . Since  $|F \cap \{a_i, b_j\}| = 1$  for all  $i \in [n]$  and  $S \subseteq F$ , we have that  $F \cap (\{a_j, b_j \mid j \in I\} \setminus S) = \emptyset$ . This completes the proof of the lemma. ◀

Instead of creating the set  $S$  by the random process, we can use  $(n, k)$ -matching universal family  $\mathcal{F}$  for  $\mathcal{Q}$  to get a deterministic algorithm. That is for each  $S \in \mathcal{F}$ , run the algorithm for INTERVAL LINE COVER on the input created using  $\ell$  and  $S$  as above, and output YES, if at least once the algorithm for INTERVAL LINE COVER returns a solution of size at most  $k$ . The correctness of the algorithm follows from the definition of  $(n, k)$ -matching universal family. By Theorem 19, the running time to construct  $\mathcal{F}$  is  $2^k k^{O(\log k)} n \log n$  and  $|\mathcal{F}| = 2^k k^{O(\log k)} n \log n$ . Hence our deterministic algorithm will run in time  $2^k k^{O(\log k)} n \log^2 n$ . This gives us the following theorem.

► **Theorem 20.** *There is a deterministic algorithm for PARAMETERIZED CONFLICT-FREE FRÉCHET DISTANCE running in time  $O(2^k k^{O(\log k)} n \log^2 n)$ .*

**Note:** This technique is especially interesting because the same technique can be used to provide FPT algorithms for similar class of problems. Consider a generalized multiple choice problem  $\mathcal{P}(\mathcal{Q}, c)$  where we are given a set  $\mathcal{Q}$  with  $n$  color classes where each color class contains  $c$  objects. The objective is to select minimum number of objects taken at most one from each color class to satisfy certain conditions. If there exists a polynomial time algorithm for  $\mathcal{P}(\mathcal{Q}, 1)$  then the same technique gives a randomized  $c^k$  algorithm.

## 5.2 Branching algorithm

For this algorithm, we will consider the more general problem which is the parameterized version of RAINBOW COVERING.

PARAMETERIZED RAINBOW COVERING

Parameter:  $k$

**Input:** A set of  $n$  pairs of intervals  $\mathcal{P}$  and a set of points  $S$  on  $X$ -axis, and  $k \in \mathbb{N} \cup \{0\}$ .

**Question:** Is there a rainbow  $Q$  of cardinality at most  $k$  such that each point in  $S$  is covered by at least one interval in  $Q$ .

We now give an algorithm based on branching for this problem. The algorithm can be modified to solve PARAMETERIZED CONFLICT-FREE FRÉCHET DISTANCE.

Let  $S = \{s_1, s_2, \dots, s_n\}$ . Without loss of generality, assume that  $s_1, s_2, \dots, s_n$  are sorted in ascending order of their  $x$ -coordinates. Now for each interval  $I_i \in P_i$  where  $P_i \in \mathcal{P}$ , assume that the interval is starting not before  $s_1$  and ending not beyond  $s_n$ . If not, trim such intervals such that they satisfy above criteria. Also initialize an integer variable  $k' = k$ .

In the first step, consider the intervals covering  $s_1$ . Let the sorted order of these intervals according to their length in descending order be  $I_{c_1} = (I_1, I_2, \dots, I_q)$  (here the length of interval  $I = [a, b]$  is calculated as  $b - a$  where we have  $b > a$ ). Let  $s_i \in S$  be the first point right to  $I_1$ . If  $q = 1$ , then choose  $I_1$  in solution, delete  $I_1, \bar{I}_1, s_1, I_2, \dots, I_q$  and all points covered by  $I_1$ . Else if  $q > 1$  then we have the following lemma.

► **Lemma 21.** *There exists an optimal solution that contains  $I_1$  or  $\bar{I}_1$ .*

**Proof.** Suppose the lemma is false. Then we have some other  $I_j$  covering  $s_1$ . But  $I_j \subseteq I_1$  and also  $\bar{I}_1$  is not in solution. So we can choose  $I_1$  and delete  $I_j$  in our new optimal solution. ◀

Thus we can either choose  $I_1$  in optimal solution or may choose  $\bar{I}_1$  in it. If  $I_1$  is chosen then delete  $I_1, \bar{I}_1, s_1, I_2, \dots, I_q$  and all points covered by  $I_1$ . If  $\bar{I}_1$  is not chosen then put  $\bar{I}_1$  in solution, and delete  $I_1, \bar{I}_1$ , all intervals  $I_i$  such that  $I_i \subseteq \bar{I}_1$  and all points covered by  $\bar{I}_1$ . At the end of the first step, put  $k' = k' - 1$ . For the second step, start with  $s_i$  if  $I_1$  is chosen in the previous step. Else consider  $s_1$  again with branching on  $I_2$ . Repeat the same procedure till either all points are covered or  $k' = 0$ . Now if atleast one branch of these  $\mathcal{O}(2^k)$  choices covers all the points then accept else reject. The time complexity of this algorithm will be  $\mathcal{O}(2^k n^2 \log n)$ . Hence we have following theorem.

► **Theorem 22.** *There is branching algorithm for PARAMETERIZED RAINBOW COVERING running in time  $\mathcal{O}(2^k n^2 \log n)$ . Similarly, there is a branching algorithm for PARAMETERIZED CONFLICT-FREE FRÉCHET DISTANCE with runtime  $\mathcal{O}(2^k n^2 \log n)$ .*

We observe that the branching algorithm can be used to obtain FPT algorithm for the PARAMETERIZED MINIMUM MAXGAP. Outline of algorithm is as follows. Start from the first point  $p_s$ . Take the farthest point from  $p_s$  having distance less than  $d$ . Let the point chosen be  $p_i$ . Then we claim that there exists an optimal solution which contains either  $p_i$  or  $\bar{p}_i$ . So branch on  $p_i$ .

### 5.3 Kernel Lower bound

In this subsection we show that PARAMETERIZED RAINBOW COVERING does not admit a polynomial kernel unless  $\text{co-NP} \subseteq \text{NP/poly}$ . Towards that we first explain one of the tools to prove such a lower bound— called composition.

► **Definition 23** (Composition [8]). A composition algorithm (also called OR-composition algorithm) for a parameterized problem  $\Pi \subseteq \Sigma^* \times \mathbb{N}$  is an algorithm that receives as input a sequence  $((x_1, k), \dots, (x_t, k))$ , with  $(x_i, k) \in \Sigma^* \times \mathbb{N}$  for each  $1 \leq i \leq t$ , uses time polynomial in  $\sum_{i=1}^t |x_i| + k$ , and outputs  $(y, k') \in \Sigma^* \times \mathbb{N}$  with (a)  $(y, k') \in \Pi \iff (x_i, k) \in \Pi$  for some  $1 \leq i \leq t$  and (b)  $k'$  is polynomial in  $k$ . A parameterized problem is compositional (or OR-compositional) if there is a composition algorithm for it.



It is *unlikely* that an NP-complete problem has both a composition algorithm and a polynomial kernel as suggested by the following theorem.

► **Theorem 24** ([8, 15]). *Let  $\Pi$  be a compositional parameterized problem whose unparameterized version  $\tilde{\Pi}$  is NP-complete. Then, if  $\Pi$  has a polynomial kernel then  $\text{co-NP} \subseteq \text{NP/poly}$ .*

Towards getting a composition for PARAMETRIZED RAINBOW COVERING, we first show how we can compose two instances and then we use this to get a composition algorithm. Next we have the following lemma.

► **Lemma 25.** *There is a polynomial time algorithm which takes two instances  $((\mathcal{P}_1, S_1), k)$  and  $((\mathcal{P}_2, S_2), k)$  of PARAMETRIZED RAINBOW COVERING as input and outputs an instance  $((\mathcal{P}, S), k + 1)$  such that  $((\mathcal{P}, S), k + 1)$  is a YES-instance of PARAMETRIZED RAINBOW COVERING if and only if at least one among  $((\mathcal{P}_1, S_1), k)$  and  $((\mathcal{P}_2, S_2), k)$  is a YES-instance of PARAMETRIZED RAINBOW COVERING.*

**Proof.** Let  $S_1 = \{s_1, \dots, s_n\}$ , and  $S_2 = \{s'_1, \dots, s'_n\}$ . Without loss of generality assume that  $s_1 < s_2 < \dots < s_n$  and  $s'_1 < s'_2 < \dots < s'_n$ . Without loss of generality we can assume that for any interval  $J$  which is part of any pair in  $\mathcal{P}_1$  and for any interval  $J'$  which is part of any pair in  $\mathcal{P}_2$ ,  $J$  is contained in  $[s_1, s_n]$  and  $J'$  is contained in  $[s'_1, s'_n]$ . Now we create a set of points  $S' = \{s_n + 1 + s'_i \mid i \in [n]\}$ , and a pair of intervals  $(I, \bar{I}) = ([s_1, s_n], [s_n + 1 + s'_1, s_n + 1 + s'_n])$ . Now we shift each interval of the instance  $((\mathcal{P}_2, S_2), k)$  by  $s_n + 1$ . For any interval  $J = [a, b]$  and  $c \in \mathbb{R}$  we use  $c + J$  to denote the interval  $[c + a, c + b]$ . Let  $S = S_1 \cup S'$  and  $\mathcal{P} = \mathcal{P}_1 \cup \{(s_n + 1 + J, s_n + 1 + \bar{J}) \mid (J, \bar{J}) \in \mathcal{P}_2\} \cup \{(I, \bar{I})\}$ . Our algorithm will output  $((\mathcal{P}, S), k + 1)$ .

Now we need to show the correctness of the algorithm. Suppose  $((\mathcal{P}, S), k + 1)$  is a YES-instance of PARAMETRIZED RAINBOW COVERING and let  $\mathcal{I}$  be a solution of size  $k + 1$ . We know that at most one of  $I$  and  $\bar{I}$  belong to  $\mathcal{I}$ . Hence, if  $I \notin \mathcal{I}$ , then  $\mathcal{I} \setminus \{I\}$  covers all the points in  $S_1$ . From the construction of  $\mathcal{P}$ , we have that all the intervals which intersects  $[s_1, s_n]$  are from  $\{J, \bar{J} \mid (J, \bar{J}) \in \mathcal{P}_1\}$ . This implies that  $\mathcal{I} \cap \{J, \bar{J} \mid (J, \bar{J}) \in \mathcal{P}_1\}$  covers all the points in  $S_1$  and  $\mathcal{I} \cap \{J, \bar{J} \mid (J, \bar{J}) \in \mathcal{P}_1\}$  is a set of conflict free intervals from  $\mathcal{P}_1$ . This implies that  $((\mathcal{P}_1, S_1), k)$  is a YES-instance of PARAMETRIZED RAINBOW COVERING. When  $\bar{I} \notin \mathcal{I}$ , by similar arguments we can show that  $((\mathcal{P}_2, S_2), k)$  is a YES-instance of PARAMETRIZED RAINBOW COVERING.

Suppose one among  $((\mathcal{P}_1, S_1), k)$  and  $((\mathcal{P}_2, S_2), k)$  is a YES-instance of PARAMETRIZED RAINBOW COVERING. Assume  $((\mathcal{P}_1, S_1), k)$  is a YES-instance and let  $\mathcal{I}$  be a solution of size  $k$  for it. Then  $\mathcal{I} \cup \{\bar{I}\}$  is a set of conflict free intervals and these intervals cover all the points in  $S$ . The case when  $((\mathcal{P}_2, S_2), k)$  is a YES-instance can be proved by similar arguments. ◀

► **Lemma 26.** PARAMETRIZED RAINBOW COVERING *is compositional.*

**Proof.** Let  $((\mathcal{P}_1, S_1), k), \dots, ((\mathcal{P}_t, S_t), k)$  be the input of the composition algorithm. If  $t > 2^k$ , then the composition algorithm solves each instance separately using Theorem 22 and outputs a trivial YES instance if at least one of the given instances is a YES instance and outputs a trivial NO instance otherwise. In this case the running time of the algorithm is bounded by  $t^2 n^{O(1)}$  and hence it is a polynomial time algorithm.

So now we can assume that  $t \leq 2^k$ . Without loss of generality assume that  $t = 2^\ell$ , where  $\ell \leq k$ . If  $t$  is not a power of 2, we can add dummy NO instances to make the total number of instances a power of 2. Now we design a recursive algorithm to get a desired output. The pseudocode is mentioned in Algorithm 1.

By induction on  $\ell$  we show that the parameter in the output instance is  $k + \ell$ . The base case is when  $\ell = 1$ , and the statement is true by Lemma 25. Now consider the induction



---

**Algorithm 1:** Composition algorithm with inputs  $((\mathcal{P}_1, S_1), k), \dots, ((\mathcal{P}_{2^\ell}, S_{2^\ell}), k)$

---

```

1 if  $\ell = 1$  then
2    $\lfloor$  Run the algorithm mentioned in Lemma 25 and return the result
3    $((\mathcal{P}'_1, S'_1), k') := \text{Algorithm 1}((\mathcal{P}_1, S_1), k), \dots, ((\mathcal{P}_{2^{\ell-1}}, S_{2^{\ell-1}}), k)$ 
4    $((\mathcal{P}'_2, S'_2), k') := \text{Algorithm 1}((\mathcal{P}_{2^{\ell-1}}, S_{2^{\ell-1}}), k), \dots, ((\mathcal{P}_{2^\ell}, S_{2^\ell}), k)$ 
5   Run algorithm mentioned in Lemma 25 on  $((\mathcal{P}'_1, S'_1), k')$  and  $((\mathcal{P}'_2, S'_2), k')$ , and return
   the result

```

---

step. For the two instances created by recursively calling Algorithm 1 on  $2^{\ell-1}$  instances, the parameters are  $k + \ell - 1$  each, by induction hypothesis. Hence, in Step 5, by Lemma 25, the parameter in the output instance is  $k + \ell$ . This implies that the parameter in the output instance is  $k + \ell \leq 2k$ .

Again by induction on  $\ell$ , we can show that the output instance of Algorithm 1 is a YES instance if and only if at least one of the input instances is a YES instance. For the base case when  $\ell = 1$ , the statement is true by Lemma 25. Now consider the induction step. Suppose that there is a YES instance in the input. Then by induction hypothesis, at least one of the instances created in Step 3 or Step 4 is a YES instance. Then, by Lemma 25, in Step 5, Algorithm 1 will output a YES instance. Now suppose Algorithm 1 output a YES instance. Then, by Lemma 25, one of the instances created in Step 3 or Step 4 is a YES instance. Hence, by induction hypothesis, at least one of the input instances is a YES instance.  $\blacktriangleleft$

By Theorem 24 and Lemma 26, we get the following theorem.

**► Theorem 27.** PARAMETRIZED RAINBOW COVERING *does not admit a polynomial kernel unless  $\text{co-NP} \subseteq \text{NP/poly}$ .*

**Acknowledgements.** The first author thanks Esther M. Arkin, Paz Carmi, Gui Citovsky, Matthew J. Katz and Joseph S. B. Mitchell for helpful discussion regarding approximation algorithms for multiple choice problems. The authors are grateful to Saket Saurabh for many valuable discussions.

---

## References

---

- 1 Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete fréchet distance in subquadratic time. *SIAM J. Comput.*, 43(2):429–449, 2014. doi:10.1137/130920526.
- 2 Pankaj K. Agarwal and Micha Sharir. Davenport-schinzel sequences and their geometric applications, 1998.
- 3 Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995. doi:10.1142/S0218195995000064.
- 4 Esther M. Arkin, Aritra Banik, Paz Carmi, Gui Citovsky, Matthew J. Katz, Joseph S. B. Mitchell, and Marina Simakov. Choice is hard. In *Algorithms and Computation – 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, pages 318–328, 2015. doi:10.1007/978-3-662-48971-0\_28.
- 5 Esther M. Arkin, Aritra Banik, Paz Carmi, Gui Citovsky, Matthew J. Katz, Joseph S. B. Mitchell, and Marina Simakov. Conflict-free covering. In *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015, Kingston, Ontario, Canada, August 10-12, 2015*, 2015. URL: <http://research.cs.queensu.ca/cccg2015/CCCG15-papers/28.pdf>.

- 6 Boris Aronov, Sarel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet distance for curves, revisited. In *Algorithms – ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, pages 52–63, 2006. doi:10.1007/11841036\_8.
- 7 Mikhail J. Atallah. Some dynamic computational geometry problems. *Computers & Mathematics with Applications*, 11(12):1171–1181, 1985. doi:10.1016/0898-1221(85)90105-1.
- 8 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.
- 9 Mario E. Consuegra and Giri Narasimhan. Geometric Avatar Problems. In Anil Seth and Nisheeth K. Vishnoi, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013)*, volume 24 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 389–400, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2013.389.
- 10 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, 2015.
- 11 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *STOC*, pages 251–260, 2010. doi:10.1145/1806689.1806725.
- 12 R. Downey and M. Fellows. *Fundamentals of parameterized complexity*. Springer, 2013.
- 13 Jeff Edmonds. *How to Think About Algorithms*. Cambridge University Press, New York, NY, USA, 2008.
- 14 Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. Technical report, Citeseer, 1994.
- 15 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *STOC*, pages 133–142, 2008. doi:10.1145/1374376.1374398.
- 16 P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, s1-10(1):26–30, 1935. URL: <http://journals.oxfordjournals.org/content/s1-10/1/26.short>, arXiv:<http://journals.oxfordjournals.org/content/s1-10/1/26.full.pdf+html>, doi:10.1112/jlms/s1-10.37.26.
- 17 John Hershberger. Finding the upper envelope of  $n$  line segments in  $o(n \log n)$  time. *Information Processing Letters*, 33(4):169–174, 1989. doi:10.1016/0020-0190(89)90136-1.
- 18 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 182–191, 1995. doi:10.1109/SFCS.1995.492475.
- 19 Kaveh Shahbaz. Applied similarity problems using fréchet distance. *CoRR*, abs/1307.6628, 2013. URL: <http://arxiv.org/abs/1307.6628>.

# Greed is Good for Deterministic Scale-Free Networks

Ankit Chauhan<sup>1</sup>, Tobias Friedrich<sup>2</sup>, and Ralf Rothenberger<sup>3</sup>

1 Hasso Plattner Institute, Potsdam, Germany

Ankit.Chauhan@hpi.de

2 Hasso Plattner Institute, Potsdam, Germany

Tobias.Friedrich@hpi.de

3 Hasso Plattner Institute, Potsdam, Germany

Ralf.Rothenberger@hpi.de

---

## Abstract

Large real-world networks typically follow a power-law degree distribution. To study such networks, numerous random graph models have been proposed. However, real-world networks are not drawn at random. In fact, the behavior of real-world networks and random graph models can be the complete opposite of one another, depending on the considered property. Brach, Cygan, Lacki, and Sankowski [SODA 2016] introduced two natural deterministic conditions: (1) a power-law upper bound on the degree distribution (PLB-U) and (2) power-law neighborhoods, that is, the degree distribution of neighbors of each vertex is also upper bounded by a power law (PLB-N). They showed that many real-world networks satisfy both deterministic properties and exploit them to design faster algorithms for a number of classical graph problems like transitive closure, maximum matching, determinant, PageRank, matrix inverse, counting triangles and maximum clique.

We complement the work of Brach et al. by showing that some well-studied random graph models exhibit both the mentioned PLB properties and additionally also a power-law lower bound on the degree distribution (PLB-L). All three properties hold with high probability for Chung-Lu Random Graphs and Geometric Inhomogeneous Random Graphs and almost surely for Hyperbolic Random Graphs. As a consequence, all results of Brach et al. also hold with high probability for Chung-Lu Random Graphs and Geometric Inhomogeneous Random Graphs and almost surely for Hyperbolic Random Graphs.

In the second part of this work we study three classical NP-hard combinatorial optimization problems on PLB networks. It is known that on general graphs, a greedy algorithm, which chooses nodes in the order of their degree, only achieves an approximation factor of asymptotically at least logarithmic in the maximum degree for Minimum Vertex Cover and Minimum Dominating Set, and an approximation factor of asymptotically at least the maximum degree for Maximum Independent Set. We prove that the PLB-U property suffices such that the greedy approach achieves a constant-factor approximation for all three problems. We also show that all three combinatorial optimization problems are APX-complete, even if all PLB-properties hold. Hence, a PTAS cannot be expected, unless P=NP.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

**Keywords and phrases** random graphs, power-law degree distribution, scale-free networks, PLB networks, approximation algorithms, vertex cover, dominating set, independent set

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.33



© Ankit Chauhan, Tobias Friedrich, and Ralf Rothenberger;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 33; pp. 33:1–33:15



Leibniz International Proceedings in Informatics

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

## 1 Introduction

A wide range of real-world networks, like Internet topologies [18], the Web [27, 8], social networks [1], power grids [32], and many other networks [29, 5, 30], exhibit a power-law degree distribution. Power-law degree distribution means that the number of nodes of degree  $k$  is proportional to  $k^{-\beta}$ , where  $\beta > 1$  is the power-law exponent, a constant intrinsic to the network. Networks with a power-law degree distribution are also called scale-free networks and have been widely studied.

To capture the degree distribution and other properties of scale-free networks, a multitude of random graph models have been proposed. These models include Preferential Attachment [8], the Configuration Model [2], Chung-Lu Random Graphs [15] and Hyperbolic Random Graphs [26]. Despite the multitude of random models, none of the models truly has the same set of properties as real-world networks.

This shortcoming of random graph models motivates studying deterministic properties of scale-free models, which can be verified on real-world networks. To describe the properties of scale-free networks without the use of random graphs, Aiello et al. [4] define  $(\alpha, \beta)$ -Power Law Graphs. The problem of this model is that it essentially demands a perfect power-law degree distribution, whereas the degree distributions of real networks normally exhibit slight deviations from power laws. Therefore,  $(\alpha, \beta)$ -Power Law Graphs are too constrained and do not capture most real networks.

To allow for those deviations in the degree distribution Brach et al. [10] define buckets containing nodes of degrees  $[2^i, 2^{i+1})$ . If the number of nodes in each bucket is at most as high as for a power-law degree sequence, a network is said to be *power-law bounded*, which we denote as a network with property **PLB-U**. They also define the property of *PLB neighborhoods*: A network has PLB neighborhoods if every node of degree  $k$  has at most as many neighbors of degree at least  $k$ , as if those neighbors were picked independently at random with probability proportional to their degree. This property we abbreviate as **PLB-N**. A formal definition of both properties can be found in Section 3. Brach et al. [10] show that various classical graph problems can be solved more efficiently in networks with properties PLB-(U,N). The graph problems addressed are transitive closure, maximum matching, determinant, PageRank, matrix inverse, counting triangles and maximum clique. Brach et al. [10] also showed experimentally that PLB-(U,N) properties hold for many real-world networks, which implies that the mentioned graph problems can be solved faster on these real-world networks than worst-case lower bounds for general graphs suggest.

### 1.1 Motivation and Results

#### PLB properties in power-law random graph models

The PLB-(U,N) properties are designed to describe power-law graphs in a way that allows analyzing algorithms deterministically. As already mentioned, there is a multitude of random graph models [2, 15, 8, 26], which can be used to generate power-law graphs. Brach et al. [10] proved that the Erased Configuration Model [2] follows PLB-U and w. h. p. also PLB-N. Since the Erased Configuration Model has a fixed degree sequence, it is relatively easy to prove the PLB-U property, but it is quite technical to prove the PLB-N property. There are other power-law random graph models, which are based on the expected degree sequence, e.g. Chung-Lu Random Graphs [15]. Brach et al. argued that for showing the PLB-U property on these models, a typical concentration statement does not work, as it accumulates the additive error for each bucket. They leave it as a *challenging open question*, whether other

random graph models also produce graphs with PLB-(U,N) properties with high probability<sup>1</sup>.

In section 4 we address this question and extend the list of random graph models with the PLB-U and PLB-N property. We prove that Chung-Lu Random Graphs and Geometric Inhomogeneous Random Graphs are PLB-(U,N) graphs with high probability and that Hyperbolic Random Graphs are PLB-(U,N) graphs almost surely.

### Algorithmic Results

The above results imply that all results of Brach et al. [10] also hold w. h. p. for Chung-Lu Random Graphs and Geometric Inhomogeneous Random Graphs and almost surely for Hyperbolic Random Graphs. Therefore the problems transitive closure, maximum matching, determinant, PageRank, matrix inverse, counting triangles and maximum clique have faster algorithms on Chung-Lu and Geometric Inhomogeneous Random Graphs w. h. p. and on Hyperbolic Random Graphs almost surely.

In this work we additionally consider the three classical NP-complete problems MINIMUM DOMINATING SET(MDS), MAXIMUM INDEPENDENT SET(MIS) and MINIMUM VERTEX COVER(MVC) on PLB-U networks. For the first two problems, positive results are already known for  $(\alpha, \beta)$ -Power Law Graphs, which are a special case of graphs with the PLB-(U,L) properties. Note that this deterministic graph class is much more restrictive and does *not* cover typical real-world graphs. On the contrary, our positive results only assume the PLB-U property. Our algorithmic results can therefore be applied to real-world networks after measuring the respective constants of the PLB-model. In section 5 we prove our main lemma, Lemma 5.2 (the potential volume lemma). Using the potential volume lemma, we prove lower bounds for MDS, MIS and MVC in the order of  $\Theta(n)$  on PLB-U networks with exponent  $\beta > 2$ . This essentially means, even taking all nodes as a solution gives a constant factor approximation. Furthermore, in Theorem 5.5 we prove that the greedy algorithm actually achieves a better constant approximation ratio. These positive results also hold for  $(\alpha, \beta)$ -Power Law Graphs.

In section 6, we consider the mentioned NP-Complete problems and prove that these problems are APX-hard even for PLB-(U,L,N) networks with  $\beta > 2$ . As a side product we also get a lower-bound on the approximability of the respective problems under some complexity theoretical assumptions. Since the negative results for  $(\alpha, \beta)$ -Power Law Graphs imply the same non-approximability on graphs with PLB-(U,L), we only consider graphs with PLB-(U,L,N) in Section 6.

### Technical Ideas

The intuition behind our positive results is simple: In a power law graph with exponent  $\beta > 2$ , any set of  $o(n)$  vertices has a volume of at most  $o(n)$ . The potential volume lemma gives upper bounds on  $\sum_{x \in S} h(\deg(x))$  in terms of  $|S|$ , where  $S$  is any set with a certain minimum volume. This is done by upper-bounding the density  $\sum_{x \in S} h(\deg(x))/|S|$  by the highest possible density of a set of size  $|S|$  in a PLB-U graph. The lemma does not only enable us to prove the stated intuition formally, but also allows us to give upper bounds on the approximation ratios of some greedy algorithms.

Our negative results rely on the graph embedding technique introduced by Shen et al. [33] for  $(\alpha, \beta)$ -Power Law Graphs.

<sup>1</sup> We say that an event  $E$  holds *w. h. p.*, if there exists an  $\delta > 0$  such that  $\Pr[E] \geq 1 - \mathcal{O}(n^{-\delta})$ , and *almost surely* if it holds with probability  $\Pr[E] \geq 1 - o(1)$ .

■ **Table 1** Comparison of the approximation ratios achieved by greedy algorithms on networks with an upper bound on the power-law degree distribution (PLB-U) and  $\beta > 2$ , and on general graphs. While on general graphs, greedy achieves only a logarithmic or polynomial approximation, greedy achieves a constant-factor-approximation on graphs with PLB-U and  $\beta > 2$ .

Problem	General Graph	Graphs with PLB-U
Minimum Dominating Set	$\mathcal{O}(\ln \Delta)$ [25]	$\Theta_n(1)$ [Theorem 5.5]
Minimum Vertex Cover	$\mathcal{O}(\ln \Delta)$ [34]	$\Theta_n(1)$ [Theorem 5.8]
Maximum Independent Set	$\mathcal{O}(\Delta)$ [17]	$\Theta_n(1)$ [Theorem 5.7]

## 2 Related Work

MDS, MVC and MIS are well studied NP-complete problems. It is known that MDS cannot be approximated within a factor of  $(1 - \varepsilon) \ln |V|$  for any  $\varepsilon > 0$  [19] unless  $\text{NP} \subseteq \text{DTIME}(|V|^{\log \log |V|})$  and not to within a factor of  $\ln \Delta - c \ln \ln \Delta$  for some  $c > 0$  [14] unless  $\text{P} = \text{NP}$ , although a simple greedy algorithm achieves an approximation ratio of  $1 + \ln \Delta$  [25]. Even for sparse graphs, MDS cannot be approximated within a factor of  $o(\ln(n))$ , since we could have a graph with a star of  $n - \sqrt{n}$  nodes to which an arbitrary graph of the  $\sqrt{n}$  remaining nodes is attached [28].

MIS cannot be approximated within a factor of  $\Delta^\varepsilon$  for some  $\varepsilon > 0$  unless  $\text{P} = \text{NP}$  [7], although a simple greedy algorithm achieves an approximation factor of  $\frac{\Delta+2}{3}$  [23]. We also know from Turán's theorem that every graph with an average degree of  $\bar{d}$  has a maximum independent set of size at least  $\frac{n}{\bar{d}+1}$ . This lower bound can already be achieved by the same greedy algorithm [23, Theorem 1].

MVC cannot be approximated within a factor of  $10\sqrt{5} - 21 \approx 1.36$  unless  $\text{P} = \text{NP}$ , whereas the simple algorithm which greedily constructs a maximal matching achieves an approximation ratio of 2 [31]. The greedy algorithm based on node degrees only achieves an approximation factor of  $\ln \Delta$ .

All three problems have already been studied in the context of  $(\alpha, \beta)$ -Power Law Graphs. Ferrante et al. [21] showed that these problems remain NP-hard for  $\beta > 0$ . Shen et al. [33] proved that there is no  $\left(1 + \frac{1}{3120\zeta(\beta)3^\beta}\right)$ -approximation for MDS and no  $\left(1 + \frac{1}{1120\zeta(\beta)3^\beta} - \varepsilon\right)$ -approximation for MIS when  $\beta > 1$  unless  $\text{P} = \text{NP}$ , showing that in this case the problem is APX-hard. For MVC, Shen et al. [33] proved that there is no PTAS when  $\beta > 1$  under the Unique Games Conjecture. They also showed that the greedy algorithm achieves a constant approximation factor for  $\beta > 2$ . Gast et al. [22] also proved a logarithmic lower bound on the approximation factor when  $\beta \leq 2$  for MDS. Hauptmann et al. [24] gave the first non-constant bound on the approximation ratio for MIS when  $\beta \leq 1$ . In contrast to  $(\alpha, \beta)$ -Power Law Graphs the PLB-U property captures a wide range of real networks, making it possible to transfer our results to them.

## 3 Preliminaries and Notation

We generally consider undirected multigraphs  $G = (V, E)$  without loops, where  $V$  denotes the set of vertices and  $E$  the multiset of edges. If we consider simple graphs, we state so specifically. Throughout the paper we use  $\deg(v)$  to denote the degree of node  $v$ ,  $d_i$  for the set of nodes of degree  $i$ ,  $d_{\geq i}$  for the set of nodes of degree greater than or equal to  $i$ . We will also let  $b_i$  denote the set of nodes  $v \in V$  with  $\deg(v) \in [2^i, 2^{i+1})$  and for  $v \in V$  we let  $N^+(v)$  denote the *inclusive neighborhood* of  $v$  in  $G$ . We also use  $d_{\min}$  and  $\Delta$  to denote the minimum



■ **Table 2** Comparison of the approximation lower bounds for polynomial-time algorithms (assuming  $P \neq NP$ ) on networks with an upper (PLB-U) and lower (PLB-L) bound on the power-law degree distribution and with PLB neighborhoods (PLB-N) with the approximation lower bounds on general graphs. Even with the additional properties of PLB-L and PLB-N the problems on graphs with PLB-U remain APX-hard, i.e. these problems cannot admit a PTAS. Better lower bounds for each problem are in respective theorem,  $\Omega(1)$  hides the PLB-L parameters  $\beta, t$  and constant  $c_2$ .

Problem	General Graph	Graph with PLB-(U,L,N)
Minimum Dominating Set (MDS)	$\Omega(\ln \Delta)$ [14]	$1 + \Omega(1)$ [Theorem 6.9]
Minimum Vertex Cover (MVC)	$\geq 1.3606$ [16]	$1 + \Omega(1)$ [Theorem 6.10]
Maximum Independent Set (MIS)	$\Omega(\text{poly}(\Delta))$ [7]	$1 + \Omega(1)$ [Theorem 6.11]

and maximum degree of the graph respectively. For a set of nodes  $S \subseteq V$ , the volume of  $S$ , denoted by  $\text{VOL}(S)$  is the sum of degrees of vertices in  $S$ ,  $\text{VOL}(S) = \sum_{v \in S} \deg(v)$ . We denote the optimal value of an objective function  $f$  on input  $x$  by  $\text{OPT}_f(x)$ . If not stated otherwise  $\log$  denotes the logarithm of base 2.

Now we give a formal definition of the PLB properties for (multi-)graphs.

► **Definition 3.1** (PLB-U [10]). Let  $G$  be an undirected  $n$ -vertex graph and  $c_1 > 0$  be a universal constant. We say that  $G$  is power law bounded (PLB-U) for some parameters  $1 < \beta = \mathcal{O}(1)$  and  $t \geq 0$  if for every integer  $d \geq 0$ , the number of vertices  $v$ , such that  $\deg(v) \in [2^d, 2^{d+1})$  is at most

$$c_1 n (t+1)^{\beta-1} \sum_{i=2^d}^{2^{d+1}-1} (i+t)^{-\beta}.$$

► **Definition 3.2** (PLB-L). Let  $G$  be an undirected  $n$ -vertex graph and  $c_2 > 0$  be a universal constant. We say that  $G$  is power law bounded PLB-L for some parameters  $1 < \beta = \mathcal{O}(1)$  and  $t \geq 0$  if for every integer  $\lfloor \log d_{\min} \rfloor \leq d \leq \lfloor \log \Delta \rfloor$ , the number of vertices  $v$ , such that  $\deg(v) \in [2^d, 2^{d+1})$  is at least

$$c_2 n (t+1)^{\beta-1} \sum_{i=2^d}^{2^{d+1}-1} (i+t)^{-\beta}.$$

Since the PLB-U property alone can capture a much broader class of networks, for example empty graphs and rings, this lower-bound is important to restrict networks to those with an actual (approximate) power-law degree distribution. In the definition of PLB-L  $d_{\min}$  is necessary because in real-world power law-networks the minimum degree is not always 1.

► **Definition 3.3** (PLB-N [10]). Let  $G$  be a PLB (multi-)graph with parameters  $\beta > 2$  and  $t \geq 0$ , and let  $c_2 > 0$  be a universal constant. We say that  $G$  has PLB neighborhoods (PLB-N) if for every vertex  $v$  of degree  $k$ , the number of neighbors of  $v$  of degree at least  $k$  is at most  $c_3 \max \left( \log n, (t+1)^{\beta-2} k \sum_{i=k}^{n-1} i(i+t)^{-\beta} \right)$ .

Note that throughout the paper we assume the parameters,  $c_i$ ,  $\beta$ , and  $t$ , of the above definitions to be constants.

► **Definition 3.4** (Graphical degree sequence). A graphical sequence is a sequence of numbers which can be the degree sequence of some graph.

## 4 Power-Law Random Graphs and the PLB properties

In this section we consider some well known power law random graph models and prove that w. h. p. or *almost surely* graphs generated by these models have PLB-U and PLB-N properties. We chose Chung-Lu Random Graphs, Geometric Inhomogeneous Random Graphs, and Hyperbolic Random Graphs, because they are common models and rather easy to analyze. Furthermore, they assume independence or some geometrically implied sparseness of edges, which is important for establishing the PLB-N property.

### 4.1 $(\alpha, \beta)$ -Power Law Graphs

► **Definition 4.1** ( $(\alpha, \beta)$ -Power Law Graph [3]). An  $(\alpha, \beta)$ -Power Law Graph is an undirected multigraph with the following degree distribution depending on two given values  $\alpha$  and  $\beta$ . For  $1 \leq i \leq \Delta = \lfloor e^{\alpha/\beta} \rfloor$  there are  $y_i = \lfloor \frac{e^\alpha}{i^\beta} \rfloor$  nodes of degree  $i$ .

► **Theorem 4.2.** *The  $(\alpha, \beta)$ -Power Law Graph with  $\beta > 1$  has the PLB-U property with  $c_1 = \frac{1}{\zeta(\beta)}$ ,  $t = 0$ , and exponent  $\beta$  and the PLB-L property with  $c_2 = \frac{1}{2\zeta(\beta)}$ ,  $t = 0$ , and exponent  $\beta$ .*

**Proof.** The number of nodes of degree  $i$  is exactly  $\lfloor \frac{e^\alpha}{i^\beta} \rfloor$ . It holds that the number of nodes of degree between  $2^d$  and  $2^{d+1} - 1$  is at most

$$e^\alpha \sum_{i=2^d}^{2^{d+1}-1} i^{-\beta} = \frac{n}{\zeta(\beta)} \sum_{i=2^d}^{2^{d+1}-1} i^{-\beta}$$

due to the definition of the degree distribution and the fact that  $n = \zeta(\beta)e^\alpha$  for  $\beta > 1$ . Furthermore, since  $i \leq \lfloor e^{\alpha/\beta} \rfloor$ ,  $\lfloor \frac{e^\alpha}{i^\beta} \rfloor$  is at least one. Therefore  $\lfloor \frac{e^\alpha}{i^\beta} \rfloor \geq \frac{1}{2} \frac{e^\alpha}{i^\beta}$ . It now holds that the number of nodes of degree between  $2^d$  and  $2^{d+1} - 1$  is at least

$$\frac{e^\alpha}{2} \sum_{i=2^d}^{2^{d+1}-1} i^{-\beta} = \frac{n}{2\zeta(\beta)} \sum_{i=2^d}^{2^{d+1}-1} i^{-\beta}. \quad \blacktriangleleft$$

► **Corollary 4.3.** *A random  $(\alpha, \beta)$ -Power Law Graph with  $\beta > 1$  created with the Erased Configuration Model has the PLB-U and PLB-N properties with high probability.*

### 4.2 Geometric Inhomogeneous Random Graphs

► **Definition 4.4** (Geometric Inhomogeneous Random Graphs (GIRGs) [11]). For  $n \in \mathbb{N}$  let  $w = (w_1, \dots, w_n)$  be a sequence of positive weights. Let  $W = \sum_{i=1}^n w_i$  be the total weight. For any vertex  $v$ , draw a point  $x_v \in \mathbb{T}^d$  uniformly and independently at random. We connect vertices  $u \neq v$  independently with probability  $p_{uv} = p_{uv}(r)$ , which now depends not only on the weights  $w_u, w_v$  but also on the positions  $x_u, x_v$ , more precisely, on the distance  $r = \|x_u - x_v\|$ . We require for some constant  $\alpha > 1$  the following edge probability condition

$$p_{uv} = \Theta\left(\min\left\{\frac{1}{\|x_u - x_v\|^{\alpha d}} \left(\frac{w_u w_v}{W}\right)^\alpha, 1\right\}\right).$$

► **Definition 4.5** (General Power-law [11]). A weight sequence  $\vec{w}$  is said to follow a general power-law with exponent  $\beta > 2$  if  $w_{\min} := \min\{w_v \mid v \in V\} = \Omega(1)$  and if there is a  $\bar{w} = \bar{w}(n) \geq n^{\omega(1/\log \log n)}$  such that for all constants  $\eta > 0$  there are  $c_1, c_2 > 0$  with

$$c_1 \frac{n}{w^{\beta-1+\eta}} \leq |\{v \in V \mid w_v \geq w\}| \leq c_2 \frac{n}{w^{\beta-1-\eta}},$$

where the first inequality holds for all  $w_{\min} \leq w \leq \bar{w}$  and the second holds for all  $w \geq w_{\min}$ .



To prove that GIRGs fulfill PLB-U and PLB-N we need the following theorem and some auxiliary lemmas by Bringmann et al. [12]. For the sake of brevity these lemmas as well as the remaining proofs of this section can be found in the full version of the paper [13].

► **Theorem 4.6** ([12]). *Let  $G$  be a GIRG with a weight sequence that follows a general power-law with exponent  $\beta$  and average degree  $\Theta(1)$ . Then, with high probability the degree sequence of  $G$  follows a power law with exponent  $\beta$  and average degree  $\Theta(1)$ , i.e. there exist constants  $c_3, c_4 > 0$  such that w. h. p.*

$$c_3 \frac{n}{k^{\beta-1+\eta}} \leq |\{v \in V | \deg(v) \geq k\}| \leq c_4 \frac{n}{k^{\beta-1-\eta}},$$

where the first inequality holds for all  $1 \leq k \leq \bar{w}$  and the second holds for all  $k \geq 1$ .

► **Theorem 4.7.** *Let  $G$  be a GIRG whose weight sequence  $\vec{w}$  follows a general power-law with exponent  $\beta' > 2$  and an  $\eta$  with  $\beta' - \eta > 2$ . Then, w. h. p.  $G$  fulfills PLB-U and PLB-N with  $\beta = \beta' - \eta$ ,  $t = 0$  and some constants  $c_1$  and  $c_2$ .*

### 4.3 Hyperbolic Random Graphs

► **Definition 4.8.** (Hyperbolic Random Graph [26]) Let  $\alpha_H > 0$ ,  $C_H \in \mathbb{R}$ ,  $T_H > 0$ ,  $n \in \mathbb{N}$  and  $R = 2 \log n + C_H$ . Then the Hyperbolic Random Graph  $G_{\alpha_H, C_H, T_H}(n)$  is a graph with vertex set  $V = [n]$  and the following properties:

- Every vertex  $v \in [n]$  draws coordinates  $(r_v, \phi_v)$  independently at random, where the angle  $\phi_v$  is chosen uniformly at random in  $[0, 2\pi)$  and the radius  $r_v \in [0, R]$  is random according to density  $f(r) = \frac{\alpha_H \sinh(\alpha_H r)}{\cosh(\alpha_H R) - 1}$ .
- Every potential edge  $e = \{u, v\} \in \binom{[n]}{2}$  is present independently with probability

$$p_H(d(u, v)) = \left(1 + e^{\frac{d(u, v) - R}{2T_H}}\right)^{-1}.$$

► **Lemma 4.9** ([11]). *Hyperbolic random graphs are a special case of GIRGs.*

This lemma directly leads to the following consequence.

► **Theorem 4.10.** *Let  $G$  be a hyperbolic random graph with  $\alpha_H > \frac{1}{2}$ . Then, almost surely  $G$  fulfills PLB-U and PLB-N with  $\beta = 2\alpha_H + 1 - \eta$ ,  $t = 0$ , constant  $\eta > 0$  and some constants  $c_1$  and  $c_2$ .*

### 4.4 Chung-Lu Random Graphs

Chung-Lu Random Graphs [15] assume a sequence of expected degrees  $w_1, w_2, \dots, w_n$  and each edge  $(i, j)$  exists independently at random with probability  $\min(1, \frac{w_i \cdot w_j}{W})$ , where  $W = \sum_{i=1}^n w_i$ . Using exactly the same techniques as for Theorem 4.6 we can prove the theorem below.

► **Theorem 4.11.** *Let  $G$  be a Chung-Lu random graph whose weight sequence  $\vec{w}$  follows a general power-law with exponent  $\beta' > 2$  and an  $\eta$  with  $\beta' - \eta > 2$ . Then w. h. p.  $G$  fulfills PLB-U and PLB-N with  $\beta = \beta' - \eta$ ,  $t = 0$  and some constants  $c_1$  and  $c_2$ .*

## 5 Greedy Algorithms

In this section we try to understand why simple greedy algorithms work efficiently in practice.

► **Definition 5.1.** An algorithm is an  $\alpha$ -approximation for problem  $P$  if it produces a solution set  $S$  with  $\alpha \geq \frac{|S|}{|\text{OPT}|}$  if  $P$  is a minimization problem and with  $\alpha \geq \frac{|\text{OPT}|}{|S|}$  if  $P$  is a maximization problem.

### Greedy Algorithm on PLB-U Networks

In this section we state our main lemma, Lemma 5.2, and use it to derive bounds on the solution size and approximation ratio of covering problems.

► **Lemma 5.2 (Potential Volume Lemma).** Let  $G$  be a (multi-)graph with the PLB-U property for some  $\beta > 2$ , some constant  $c_1 > 0$  and some constant  $t \geq 0$ . Let  $S$  be a solution set for which we can define a function  $g: \mathbb{R}^+ \rightarrow \mathbb{R}$  continuously differentiable and  $h(x) := g(x) + C$  for some constant  $C$  such that

1.  $g$  non-decreasing,
2.  $g(2x) \leq c \cdot g(x)$  for all  $x \geq 2$  and some constant  $c > 0$ ,
3.  $g'(x) \leq \frac{g(x)}{x}$ ,

then it holds that  $\sum_{x \in S} h(\deg(x))$  is at most

$$\left( c \left( 1 + \frac{\beta - 1}{\beta - 2} \frac{1}{1 - \left( \frac{t+2}{t+1} \right)^{1-\beta}} \right) g \left( \left( c_1 \frac{\beta-1}{\beta-2} \frac{n}{M} \cdot 2^{\beta-1} \cdot (t+1)^{\beta-1} \right)^{\frac{1}{\beta-2}} \right) + C \right) \cdot |S|,$$

where  $M(n) \geq 1$  is chosen such that  $\sum_{x \in S} \deg(x) \geq M$ .

For the proof one can refer to the full version of the paper [13].

All our bounds are in terms of the following two constants, which stem from the Potential Volume Lemma and which we will define for the sake of brevity:

$$a_{\beta,t} := \left( 1 + \frac{\beta - 1}{\beta - 2} \frac{1}{1 - \left( \frac{t+2}{t+1} \right)^{1-\beta}} \right) \text{ and } b_{c_1,\beta,t} := \left( c_1 \frac{\beta-1}{\beta-2} \cdot 2^\beta \cdot (t+1)^{\beta-1} \right)^{\frac{1}{\beta-2}}.$$

#### 5.1 Minimum Dominating Set

The idea for lower-bounding the size of a dominating set is essentially the same as the one by Shen et al. [33] and by Gast et al. [22] in the context of  $(\alpha, \beta)$ -Power-Law Graphs: Every set of  $o(n)$  nodes in a power-law graph can dominate only  $o(n)$  many nodes. For graphs with PLB-U this is implied by our Potential Volume Lemma. Finally, we will show that

► **Theorem 5.3.** For a multigraph without loops and isolated vertices and with the PLB-U property with parameters  $\beta > 2$ ,  $c_1 > 0$  and  $t \geq 0$ , the minimum dominating set is of size at least

$$(2 \cdot a_{\beta,t} \cdot b_{c_1,\beta,t} + 1)^{-1} n = \Theta(n).$$

### 5.1.1 The Greedy Algorithm

Theorem 5.3 implies that taking all nodes already gives a constant approximation factor, but now we want to show that using the classical greedy algorithm actually guarantees an even better approximation factor.

The proof of the following theorem is an adaptation of the proof for the greedy SET COVER algorithm to the case of unweighted DOMINATING SET.

► **Theorem 5.4** ([25]). *Let  $S$  the solution of the greedy algorithm and  $\text{OPT}$  an optimal solution for DOMINATING SET. Then it holds that*

$$|C| \leq \sum_{x \in \text{OPT}} H_{\deg(x)+1},$$

where  $H_k$  is the  $k$ -th harmonic number.

The interested reader can find the proof of the above theorem in the full version [13] of the paper. By using the inequality from Theorem 5.4 together with the Potential Volume Lemma 5.2, we can derive the following approximation factor for the greedy algorithm.

► **Theorem 5.5.** *For a multigraph without loops and isolated vertices and with the PLB- $U$  property with parameters  $\beta > 2$ ,  $c_1 > 0$  and  $t \geq 0$ , the classical greedy algorithm for MINIMUM DOMINATING SET (cf. [17]) has an approximation factor of at most*

$$\log_3(5) \cdot a_{\beta,t} \cdot \ln(b_{c_1,\beta,t} + 1) + 1 = \Theta(1).$$

**Proof.** From the analysis of the greedy algorithm we know that for its solution  $C$  and an optimal solution  $\text{OPT}$  it holds that

$$|C| \leq \sum_{x \in \text{OPT}} H_{\deg(x)+1} \leq \sum_{x \in \text{OPT}} \ln(\deg(x) + 1) + 1,$$

where  $H_k$  denotes the  $k$ -th harmonic number. We can now choose  $h(x) = g(x) + 1$  with  $g(x) = \ln(x + 1)$ .  $g(x)$  satisfies (i), (ii) with  $c = \log_3(5)$  and (iii). As we assume there to be no nodes of degree 0, it holds that

$$\sum_{x \in \text{OPT}} \deg(x) \geq \frac{n}{2} =: M,$$

since all nodes have to be covered. We can now use Lemma 5.2 with  $S = \text{OPT}$  to derive that

$$|C| \leq (\log_3(5) \cdot a_{\beta,t} \cdot \ln(b_{c_1,\beta,t} + 1) + 1) |\text{OPT}|. \quad \blacktriangleleft$$

Note that in PLB networks the maximum degree can be  $\Delta = \Theta(n^{\frac{1}{\beta-1}})$ . That means the simple bound for the greedy algorithm gives us only an approximation ratio of  $\ln(\Delta + 1) = \Theta(\log n)$ .

## 5.2 Maximum Independent Set

For networks with the PLB-L property only, we can already derive the following lower bound on the size of an optimal solution.

► **Lemma 5.6.** *A graph with the PLB-L property with parameters  $\beta > 2$ ,  $c_2 > 0$  and  $t \geq 0$ , has an independent set of size at least  $\frac{c_2(t+1)^{\beta-1}}{(t+d_{\min})^\beta(d_{\min}+1)} \cdot n$  or of size at least  $\frac{c_2}{(t+1)} \cdot n$  if we assume  $G$  to be connected and  $d_{\min} = 1$ .*

We can even go a step further and show that *all* maximal independent sets have to be quite big, even if we only have the PLB-U property. Since the PLB-U property with  $\beta > 2$  induces a constant average degree, this already gives us a constant approximation factor for MAXIMUM INDEPENDENT SET on networks with this property due to Turán's theorem. Although we can not give better bounds for the maximum independent set, Theorem 5.3 immediately implies a lower bound for the size of *all maximal independent sets*:

► **Theorem 5.7.** *In a multigraph without loops and isolated vertices and with the PLB-U property with parameters  $\beta > 2$ ,  $c_1 > 0$  and  $t \geq 0$ , every maximal independent set is of size at least*

$$(2 \cdot a_{\beta,t} \cdot b_{c_1,\beta,t} + 1)^{-1} n = \Theta(n).$$

Especially, it holds that computing any maximal independent set gives an approximation factor of at most  $2 \cdot a_{\beta,t} \cdot b_{c_1,\beta,t} + 1$ . The above theorem holds since every maximal independent set is also a dominating set. It is easy to see that these lower bounds do not hold in sparse graphs in general, since in a star the center node also constitutes a maximal independent set.

### 5.3 Vertex Cover

From the results we know about DOMINATING SET, we can also derive some results about VERTEX COVER in graphs without isolated vertices.

► **Theorem 5.8.** *In a multigraph without loops and isolated vertices and with the PLB-U property with parameters  $\beta > 2$ ,  $c_1 > 0$  and  $t \geq 0$ , the minimum vertex cover is of size at least*

$$(2 \cdot a_{\beta,t} \cdot b_{c_1,\beta,t} + 1)^{-1} n.$$

The above theorem follows because every vertex cover in a graph without isolated vertices is also a dominating set. Again, the theorem immediately implies an approximation factor of at most  $2 \cdot a_{\beta,t} \cdot b_{c_1,\beta,t} + 1$ .

## 6 Approximation Hardness for Simple Graphs

To show the actual non-approximability and APX hardness, we use the embedding framework by Shen et al. [33].

► **Definition 6.1** (Embedded-Approximation-Preserving Reduction [33]). Given an optimal substructure problem  $O$ , a reduction from an instance on graph  $G = (V, E)$  to another instance on a (power law) graph  $G' = (V', E')$  is called *embedded approximation-preserving* if it satisfies the following properties:

1.  $G$  is a subset of maximal connected components of  $G'$ ;
2. The optimal solution of  $O$  on  $G'$ ,  $\text{OPT}(G')$ , is upper bounded by  $C \cdot \text{OPT}(G)$  where  $C$  is a constant correspondent to the growth of the optimal solution.

► **Theorem 6.2** ([6]). MINIMUM VERTEX COVER, MAXIMUM INDEPENDENT SET and MINIMUM DOMINATING SET are APX-complete for cubic simple graphs.

Having shown an embedded-approximation-preserving reduction, we can use the following lemma to show hardness of approximation.

► **Lemma 6.3** ([33]). *Given an optimal substructure problem  $O$ , if there exists an embedded-approximation-preserving reduction from a graph  $G$  to another graph  $G'$  and if  $O$  is  $\varepsilon$ -inapproximable on  $G$ , then  $O$  is  $\delta$ -inapproximable on  $G'$ , where  $\delta$  is lower bounded by  $\frac{\varepsilon C}{(C-1)\varepsilon+1}$  if  $O$  is a maximization problem and by  $\frac{\varepsilon+C-1}{C}$  if  $O$  is a minimization problem.*

► **Theorem 6.4** ([6, 14]). *In 3-bounded simple graphs it is NP-hard to approximate MDS within a factor of  $\frac{391}{390}$ .*

► **Theorem 6.5** ([6, 9]). *In 3-bounded simple graphs it is NP-hard to approximate MIS within a factor of  $\frac{140}{139} - \gamma$  for any  $\gamma > 0$ .*

► **Theorem 6.6** ([16, 20]). *In regular simple graphs MVC is hard to approximate within a factor of  $10\sqrt{5} - 21 \approx 1.3606$  unless  $P = NP$ .*

We will use this framework as follows: First, we show how to embed cubic graphs into simple graphs with PLB-U, PLB-L and PLB-N. Then, we derive the value of  $C$  as in Definition 6.1 for each problem we consider. Last, we use Lemma 6.3 together with the known inapproximability results for the considered problems on cubic graphs to derive the approximation hardness on graphs with PLB-U, PLB-L and PLB-N.

We start by showing the embedding of cubic simple graphs into simple graphs with PLB-U, PLB-L and PLB-N. To this end we use stars as the gadgets for our embeddings. The following is a simple observation and is therefore stated without a formal proof.

► **Lemma 6.7.** *A star of size  $n$  has a minimum dominating set and a minimum vertex cover of size 1 and maximum independent set of size  $n - 1$ . Also, these can be computed in polynomial time.*

► **Lemma 6.8.** *Any cubic simple graph  $G$  can be embedded into a simple graph  $G_{PLB}$  having the PLB-U, PLB-L and PLB-N properties for any  $\beta > 2$  and any  $t \geq 0$ .*

**Proof.** Suppose we are given  $\beta$  and  $t$ . Again, we want to determine  $c_1$  and  $c_2$  of PLB-U and PLB-L respectively. Let  $n$  be the number of nodes in graph  $G$  and let  $N = cn$  be the number of nodes in  $G_{PLB}$  for some constant  $c$  to be determined. Like in Lemma 6.3 we have to ensure a number of conditions to get a graphical degree sequence. To hide a cubic graph in the respective bucket of  $G_{PLB}$ , we need

$$c_1 N (t+1)^{\beta-1} \sum_{i=2}^3 (i+t)^{-\beta} = c_1 N (t+1)^{\beta-1} \left( \frac{1}{(2+t)^\beta} + \frac{1}{(3+t)^\beta} \right) \geq n.$$

As we will see, we can choose the constant  $c_1$  arbitrarily large, so the former condition is no real restriction. Then we choose the maximum degree  $\Delta$  such that

$$d_{max}(G_{PLB}) = (cn)^{\frac{1}{\beta-1}}.$$

In our embedding we just fill each bucket  $i \geq 2$  with the number of stars of size  $2^i + 1$  it needs to reach its lower bound. Bucket 1 can get up to  $n$  nodes, since we hide the graph  $G$  in it and bucket 0 gets all the degree-one nodes of our star gadgets. By filling a bucket (other than buckets 0 and 1) we might deviate by at most one from the lower bound of that bucket. Then, we add additional stars within the bounds of our buckets until we have exactly  $N$  nodes. If we only need one more node, we just add it and connect it to an arbitrary star. This does not change the properties of the star or the degree of its center enough to make it change its bucket.

In order for this to be possible we need to ensure that after filling all buckets to their lower bound, there is still some slack until we reach  $N$ . This is the case if the following inequality holds true

$$\begin{aligned}
 & n + \sum_{i=0}^{\lfloor \log \Delta \rfloor} \left( (2^i + 1) \left( 1 + c_2 N (t+1)^{\beta-1} \sum_{j=2^i}^{2^{i+1}-1} (j+t)^{-\beta} \right) \right) \\
 & \leq \frac{N}{c} + \log N^{\frac{1}{\beta-1}} + \frac{c_2}{(t+1)} N + \frac{c_2}{\beta-1} N + 2\Delta + c_2 N + \frac{c_2}{\beta-2} N (t+1) \\
 & \leq N \left( \frac{1}{c} + \eta + \frac{c_2}{t+1} + \frac{c_2}{\beta-1} + \eta + c_2 + \frac{c_2}{\beta-2} (t+1) \right) \\
 & \leq N,
 \end{aligned}$$

where in the second line we used the inequalities

$$\begin{aligned}
 & \sum_{i=0}^{\lfloor \log \Delta \rfloor} \left( 1 + c_2 N (t+1)^{\beta-1} \sum_{j=2^i}^{2^{i+1}-1} (j+t)^{-\beta} \right) \leq \log N^{\frac{1}{\beta-1}} + \frac{c_2}{(t+1)} N + \frac{c_2}{\beta-1} N, \\
 & \sum_{i=0}^{\lfloor \log \Delta \rfloor} \left( 2^i \left( 1 + c_2 N (t+1)^{\beta-1} \sum_{j=2^i}^{2^{i+1}-1} (j+t)^{-\beta} \right) \right) \leq 2\Delta + c_2 N + \frac{c_2}{\beta-2} N (t+1)
 \end{aligned}$$

and choose a constant  $\eta > 0$  arbitrarily small.

From this last condition we can derive

$$c \geq 1 + \frac{\eta' + c_2 \left( \frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1 \right)}{1 - \eta' - c_2 \left( \frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1 \right)},$$

since  $\eta$  and therefore  $\eta'$  can be arbitrarily small. We choose  $\eta' = c_2 \left( \frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1 \right)$  to get

$$c = 1 + \frac{2c_2 \left( \frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1 \right)}{1 - 2c_2 \left( \frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1 \right)} = \frac{1}{1 - 2c_2 \left( \frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1 \right)}.$$

The use of star gadgets means we also have to guarantee that  $c_1$  is big enough for all degree-one nodes to fit into bucket 0. Since  $c_1$  can be arbitrarily large, this is no problem.

Now we can essentially choose  $c_1$  arbitrarily large and  $c_2$  arbitrarily small, guaranteeing  $c > 1$  and a large enough gap to have a graphical degree sequence. At the same time our choice of  $c$  guarantees that we can fill the graph with exactly  $N$  nodes. Furthermore, since every node has a constant number of neighbors of equal or higher degree,  $G_{PLB}$  also fulfills PLB-N, which always allows us at least  $c_3 \log N$  many neighbors. ◀

## 6.1 Dominating Set

► **Theorem 6.9.** *For every  $\beta > 2$  and every  $t \geq 0$  MINIMUM DOMINATING SET cannot be approximated to within a factor of  $1 + \left( 130 \cdot \left( 4 \frac{1 - \frac{c_2}{t+1}}{1 - 2c_2 \left( \frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1 \right)} + 1 \right) \right)^{-1}$  on simple graphs with PLB-U, PLB-L and PLB-N unless  $P = NP$ .*

**Proof.** Lemma 6.8 gives us an  $L$ -reduction from a cubic graph  $G$  to a simple graph  $G_{PLB}$  with the PLB-U, PLB-L and PLB-N properties. The  $L$ -reduction from a cubic graph  $G$  to a simple graph  $G_{PLB}$  together with Theorem 6.2 implies that MDS is APX hard for simple graphs with PLB-(U,L,N). Let  $\text{OPT}(G)$  and  $\text{OPT}(G_{PLB})$  denote the size of a minimum dominating set for  $G$  and  $G_{PLB}$  respectively. Let  $b_i$  be the set of nodes in PLB bucket  $i$ , i.e. the set of nodes  $v \in V$  with  $\deg(v) \in [2^i, 2^{i+1} - 1]$ . We know that  $\text{OPT}(G) \geq \frac{n}{4}$  and from Lemma 6.7 we can derive  $\text{OPT}(G_{PLB} \setminus G) = N - n - |b_0|$ . It now holds that

$$\begin{aligned} \text{OPT}(G_{PLB}) &= \text{OPT}(G) + \text{OPT}(G_{PLB} \setminus G) \\ &= \text{OPT}(G) + N - n - |b_0| \\ &\leq \text{OPT}(G) + N - n - \frac{c_2}{t+1}N \\ &= \text{OPT}(G) + \left(c - 1 - c \frac{c_2}{t+1}\right)n \\ &\leq \text{OPT}(G) + \left(c - 1 - c \frac{c_2}{t+1}\right)4\text{OPT}(G) \\ &= \left(4c \left(1 - \frac{c_2}{t+1}\right) - 3\right)\text{OPT}(G). \end{aligned}$$

In the context of Definition 6.1 and Lemma 6.3 this means  $C = 4c \left(1 - \frac{c_2}{t+1}\right) - 3$ . Due to Theorem 6.4 it also holds that  $\varepsilon = \frac{391}{390}$  in the context of Lemma 6.3. This gives us an approximation hardness of

$$\begin{aligned} 1 + \frac{\varepsilon - 1}{C} &= 1 + \frac{3}{390 \cdot \left(4c \left(1 - \frac{c_2}{t+1}\right) - 3\right)} \\ &= 1 + \frac{3}{390 \cdot \left(4 \frac{1 - \frac{c_2}{t+1}}{1 - 2c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1\right)} + 1\right)} \\ &= 1 + \left(130 \cdot \left(4 \frac{1 - \frac{c_2}{t+1}}{1 - 2c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1\right)} + 1\right)\right)^{-1} \end{aligned}$$

due to our choice of  $c$  in Lemma 6.8. ◀

By using similar arguments as for Theorem 6.9 we can prove Theorem 6.10 and Theorem 6.11.

► **Theorem 6.10.** *For every  $\beta > 2$  and every  $t \geq 0$  MINIMUM VERTEX COVER cannot be approximated to within a factor of  $1 + \frac{(1 - 2c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1\right))(10\sqrt{5} - 22)}{2c_2 \left(\frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1\right) + 1}$  on simple graphs with PLB-U, PLB-L and PLB-N unless  $P = NP$ .*

► **Theorem 6.11.** *For every  $\beta > 2$  and every  $t \geq 0$  MAXIMUM INDEPENDENT SET cannot be approximated to within a factor of  $1 + \frac{\left(\frac{1}{139} - \gamma\right) \left((t+1) \left(1 - 2c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1\right)\right)\right)}{4c_1 \left(\frac{140}{139} - \gamma\right) + (t+1) \left(1 - 2c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} + \frac{t+1}{\beta-2} + 1\right)\right)}$  for any  $\gamma > 0$  on simple graphs with PLB-U, PLB-L and PLB-N unless  $P = NP$ .*

---

## References

- 1 Lada A. Adamic, Orkut Buyukkokten, and Eytan Adar. A social network caught in the web. *First Monday*, 8(6), 2003.

- 2 William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *32nd Symp. Theory of Computing (STOC)*, pages 171–180, 2000.
- 3 William Aiello, Fan Chung, and Linyuan Lu. A random graph model for power law graphs. *Experiment. Math.*, 10(1):53–66, 2001.
- 4 William Aiello, Fan R.K. Chung, and Linyuan Lu. A random graph model for massive graphs. In *32nd Symp. Theory of Computing (STOC)*, pages 171–180, 2000.
- 5 Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.
- 6 Paola Alimonti and Viggo Kann. *Hardness of approximating problems on cubic graphs*, pages 288–298. Springer Berlin Heidelberg, 1997.
- 7 Noga Alon, Uriel Feige, Avi Wigderson, and David Zuckerman. Derandomized graph products. *Computational Complexity*, 5(1):60–75, 1995.
- 8 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
- 9 Piotr Berman and Marek Karpinski. On some tighter inapproximability results. In *26th Intl. Coll. Automata, Languages and Programming (ICALP)*, pages 200–209, 1999.
- 10 Paweł Brach, Marek Cygan, Jakub Lacki, and Piotr Sankowski. Algorithmic complexity of power law networks. In *27th Symp. Discrete Algorithms (SODA)*, pages 1306–1325, 2016.
- 11 Karl Bringmann, Ralph Keusch, and Johannes Lengler. Geometric inhomogeneous random graphs. *CoRR*, abs/1511.00576, 2015. URL: <http://arxiv.org/abs/1511.00576>.
- 12 Karl Bringmann, Ralph Keusch, and Johannes Lengler. Average distance in a general class of scale-free networks with underlying geometry. *CoRR*, abs/1602.05712, 2016. URL: <http://arxiv.org/abs/1602.05712>.
- 13 Ankit Chauhan, Tobias Friedrich, and Ralf Rothenberger. Greed is good for deterministic scale-free networks. *CoRR*, abs/1610.04217, 2016. URL: <http://arxiv.org/abs/1610.04217>.
- 14 Miroslav Chlebík and Janka Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Inf. Comput.*, 206(11):1264–1275, 2008.
- 15 F. Chung and L. Lu. Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, 6(2):125–145, 2002.
- 16 Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162:439–485, 2005.
- 17 Ding-Zhu Du, Ker-I Ko, and Xiaodong Hu. *Design and Analysis of Approximation Algorithms*. Springer Publishing Company, Incorporated, 2011.
- 18 Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *Symp. Communications Architectures and Protocols (SIGCOMM)*, pages 251–262, 1999.
- 19 Uriel Feige. A threshold of  $\ln n$  for approximating set cover. *J. ACM*, 45(4):634–652, 1998.
- 20 Uriel Feige. Vertex cover is hardest to approximate on regular graphs. Technical report, Citeseer, 2003.
- 21 Alessandro Ferrante, Gopal Pandurangan, and Kihong Park. On the hardness of optimization in power-law graphs. *Theoretical Computer Science*, 393(1–3):220–230, 2008.
- 22 Mikael Gast, Mathias Hauptmann, and Marek Karpinski. Inapproximability of dominating set in power law graphs. *CoRR*, abs/1212.3517, 2012. URL: <http://arxiv.org/abs/1212.3517>.
- 23 Magnús M. Halldórsson and Jaikumar Radhakrishnan. Greed is good: approximating independent sets in sparse and bounded-degree graphs. In *26th Symp. Theory of Computing (STOC)*, pages 439–448, 1994.
- 24 Mathias Hauptmann and Marek Karpinski. On the approximability of independent set problem on power law graphs. *CoRR*, abs/1503.02880, 2015.



- 25 M. Y. Kao. *Encyclopedia of Algorithms*. Encyclopedia of Algorithms. Springer, 2008.
- 26 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguná. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010.
- 27 Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the web for emerging cyber-communities. *Computer Networks*, 31(11-16):1481–1493, 1999.
- 28 Christoph Lenzen and Roger Wattenhofer. Minimum dominating set approximation in graphs of bounded arboricity. In *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, pages 510–524, 2010.
- 29 M. E. J. Newman. Random graphs as models of networks. In *Handbooks of Graphs and Networks*, pages 35–68. Wiley-VCH, 2003.
- 30 M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.
- 31 Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.
- 32 Arun G. Phadke and James S. Thorp. *Computer Relaying for Power Systems*. John Wiley & Sons, Ltd, 2009.
- 33 Yilin Shen, Dung T. Nguyen, Ying Xuan, and My T. Thai. New techniques for approximating optimal substructure problems in power-law graphs. *Theoretical Computer Science*, 447:107–119, 2012.
- 34 Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.



# Independent-Set Reconfiguration Thresholds of Hereditary Graph Classes

Mark de Berg<sup>1</sup>, Bart M. P. Jansen<sup>2</sup>, and Debankur Mukherjee<sup>3</sup>

- 1 Eindhoven University of Technology, Department of Computer Science, Eindhoven, The Netherlands  
[m.t.d.berg@tue.nl](mailto:m.t.d.berg@tue.nl)
- 2 Eindhoven University of Technology, Department of Computer Science, Eindhoven, The Netherlands  
[b.m.p.jansen@tue.nl](mailto:b.m.p.jansen@tue.nl)
- 3 Eindhoven University of Technology, Department of Mathematics, Eindhoven, The Netherlands  
[d.mukherjee@tue.nl](mailto:d.mukherjee@tue.nl)

---

## Abstract

Traditionally, reconfiguration problems ask the question whether a given solution of an optimization problem can be transformed to a target solution in a sequence of small steps that preserve feasibility of the intermediate solutions. In this paper, rather than asking this question from an algorithmic perspective, we analyze the combinatorial structure behind it. We consider the problem of reconfiguring one independent set into another, using two different processes: (1) exchanging exactly  $k$  vertices in each step, or (2) removing or adding one vertex in each step while ensuring the intermediate sets contain at most  $k$  fewer vertices than the initial solution. We are interested in determining the minimum value of  $k$  for which this reconfiguration is possible, and bound these threshold values in terms of several structural graph parameters. For hereditary graph classes we identify structures that cause the reconfiguration threshold to be large.

**1998 ACM Subject Classification** G.2.1 Combinatorics

**Keywords and phrases** reconfiguration, independent set, Token Addition Removal, Token Sliding

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.34

## 1 Introduction

Over the past decade, reconfiguration problems have drawn a lot of attention of researchers in algorithms and combinatorics [4, 5, 9, 13, 15, 16, 18, 22, 24]. In this framework, one asks the following question: Given two solutions  $I, J$  of a fixed optimization problem, can  $I$  be transformed into  $J$  by a sequence of small steps that maintain feasibility for all intermediate solutions? Such problems are practically motivated by the fact it may be impossible to adapt a new production strategy instantaneously if it differs too much from the strategy that is currently in use; changes have to be made in small steps, but production has to keep running throughout. From a theoretical perspective, the study of reconfiguration problems provides deep insights into the structure of the solution space. One of the well-studied examples is when the solution space consists of all the independent sets of a graph (optionally all having a prescribed size). In this case, three types of reconfiguration rules have been considered. These are naturally explained using *tokens* on vertices of the graph. In *Token Addition Removal* (TAR) [16, 22], there is a token on every vertex of the initial independent set, and there is a buffer of tokens, initially empty. A step consists of removing a token from a vertex



© Mark de Berg, Bart M. P. Jansen, and Debankur Mukherjee;  
licensed under Creative Commons License CC-BY

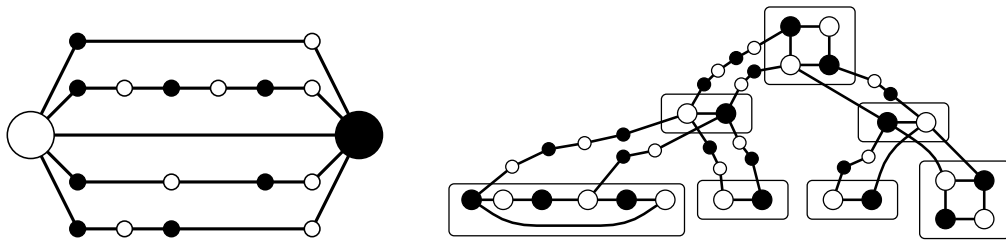
36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 34; pp. 34:1–34:15



Leibniz International Proceedings in Informatics

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



(a) A pumpkin of size 18.

(b) A graph of treewidth two with a complete binary tree  $T$  of depth two as a bipartite topological double minor.

■ **Figure 1** The bipartite structures responsible for large MTJ and TAR reconfiguration thresholds, respectively. A *pumpkin* consists of odd-length vertex-disjoint paths between two vertices. The special form of topological *minor* represents each vertex of the tree  $T$  by an edge or even cycle in  $G$ , and each edge of  $T$  by two odd-length paths connecting vertices in opposite partite sets in  $G$ .

and placing it in the buffer, or placing a buffer token onto a vertex of the graph. The set of vertices with tokens must form an independent set at all times, and the goal is to move the tokens from the initial to the target independent set while ensuring the buffer size never exceeds a given threshold. In *Token Sliding* (TS) [18, 15], a step consists of replacing one vertex  $v$  in the independent set by a neighbor of  $v$  (the token slides along an edge). In *Token Jumping* (TJ) [18] a step also consists of replacing a single vertex, but the newly added vertex need not have any neighboring relation with the replaced vertex (the token jumps). Token jumping reconfiguration is equivalent to TAR reconfiguration with a buffer of size one.

These models have been analyzed in detail in the recent literature on algorithms [4, 5, 9, 13, 14, 21], complexity theory [15, 16, 18, 22], combinatorics [6, 12], and even statistical physics [17, 19, 23]. It is known that the reconfiguration problem under all the above three rules is PSPACE-complete for general graphs, perfect graphs, and planar graphs [15, 18, 16]. The TJ and TAR reconfiguration problems are PSPACE-complete even for bounded bandwidth graphs [24]. Further analyses on the complexity can be found in [4, 5, 9, 13, 21]. The constrained token-moving problems are related to pebbling games that have been studied in the literature, with applications to robot motion planning [1, 6, 12, 14].

As mentioned, the goal in reconfiguring independent sets is to go from one given independent  $I$  to another one  $J$  by a sequence of small steps. In the TS and TJ models, a step involves moving a single token. This is ideal, but unfortunately reconfiguration is often impossible in the TS or TJ model. Reconfiguration in the TAR model is always possible if one makes the buffer size sufficiently large. However, a large buffer size is undesirable. We are interested in determining the minimum buffer size that is sufficient to ensure any independent set in a given graph  $G$  can be reconfigured to any target independent set of the same size. We call this minimum the TAR *reconfiguration threshold* (precise definitions in Section 2). Our aim is to bound the threshold in terms of properties of the graph, and to identify the structures contained in hereditary graph classes that cause large thresholds. We also generalize the TJ model to *Multiple Token Jumping* (MTJ), where in each step a prescribed number of tokens may be moved simultaneously. In the MTJ model, the question becomes: What is the minimum number of simultaneously jumping tokens needed to ensure any reconfiguration is possible? This quantity is called the MTJ *reconfiguration threshold*.

**Our contribution.** We provide upper and lower bounds on the MTJ and TAR reconfiguration thresholds in terms of several graph parameters. Our bounds apply to the reconfiguration

thresholds of hereditary *graph classes*. The threshold of a graph class is the supremum of the threshold values of the graphs in that class: it is the smallest value  $k$  such that for any graph  $G$  in the class, any source independent set  $I$  in  $G$  can be reconfigured into any target independent set  $J$  using steps of size  $k$  (for MTJ) or a buffer of size  $k$  (for TAR).

The MTJ reconfiguration threshold of graphs that are structurally very simple, may nevertheless be very large. For example, an even cycle with  $2n$  vertices can be partitioned into two independent sets  $I$  and  $J$  of size  $n$  each. Any MTJ reconfiguration of  $I$  into  $J$  requires a jump of  $n$  vertices, and this is trivially sufficient. Since a cycle has a feedback vertex set (FVS, see Section 2) of size one, the MTJ threshold cannot be bounded in terms of the size of a minimum feedback vertex set. However, we prove that the threshold is upper-bounded by the size of a minimum vertex cover of  $G$ . Although this bound is tight in the worst case, there are many graph classes with a small MTJ threshold even though they require a large vertex cover. Trees for example have MTJ threshold at most one. We therefore introduce the notion of *pumpkin*, which consists of two nodes connected by at least two vertex-disjoint paths of odd length (Figure 1a). The *size* of a pumpkin is its total number of vertices. We characterize the MTJ reconfiguration threshold of a hereditary graph class  $\Pi$  in terms of the size of the largest pumpkin it contains: the MTJ reconfiguration threshold is upper- and lower-bounded in terms of the largest pumpkin contained in a bipartite graph in  $\Pi$ .

TAR reconfiguration is more versatile than MTJ reconfiguration. In the concrete example of a  $2n$ -cycle discussed above, its MTJ threshold is  $n$  while any pair of independent sets can be reconfigured in the TAR model using a buffer of size two. Moreover, we show that any graph that has a feedback vertex set of size  $k$  has TAR reconfiguration threshold at most  $k + 1$ , and reconfiguring one side of the complete bipartite graph  $K_{n,n}$  to the other side shows that this is tight. Our main result concerning TAR reconfiguration states that the TAR reconfiguration threshold of any graph is upper-bounded by its pathwidth. Somewhat surprisingly, there are graphs of constant treewidth (treewidth 2 suffices) for which the TAR reconfiguration threshold is arbitrarily large. We also introduce the concept of *bipartite topological double minor* (BTD-minor), see Figure 1b, and show using an isoperimetric inequality that any hereditary graph class containing a graph having a complete binary tree of depth  $d$  as a BTD-minor, has TAR reconfiguration threshold  $\Omega(d)$ . We conjecture that the TAR reconfiguration threshold can also be upper-bounded in terms of the depth of the largest complete binary tree BTD-minor, but we have not been able to prove this (see Section 6).

**Applications.** The MTJ and TAR reconfiguration thresholds play an important role in statistical physics and wireless communication networks. To understand the importance of the TAR reconfiguration threshold, consider the following process: In a graph  $G$ , nodes are trying to become *active* (transmit information) at some rate, independently of each other in a distributed manner. When a potential activation occurs at a node, it can only become active if none of its neighboring nodes are active at that moment (otherwise the transmissions would interfere). An active node deactivates at some rate independent of the other processes. At any point in time, the set of active nodes in this process forms an independent set of the graph. In statistical physics, this process is known as Glauber dynamics with *hard-core interaction*. This activity process on graphs has applications in various fields of study. Loosely speaking, when the activation rate is large, in the long run the above process always tries to stay in a maximum independent set. For graphs with more than one maximum independent set, it is interesting to study the time this process takes to reach a target independent set, starting from some specific independent set. This time depends crucially upon what we call the TAR reconfiguration threshold of the underlying

graph [23]. In particular, the mixing time of the Glauber dynamics on a graph increases exponentially with its TAR reconfiguration threshold, and hence the Glauber dynamics on the graph is fast mixing if and only if the TAR reconfiguration threshold is small.

## 2 Preliminaries

In this section we give the most important graph-theoretic definitions. Notions not defined here can be found in standard textbooks [7, 10]. Due to space restriction, proofs of statements marked (★) have been omitted; they can be found in the full version of the paper [8].

A graph is a pair  $G = (V, E)$ , where  $V$  is the set of vertices, and  $E$  is the set of edges. We also use  $V(G)$  and  $E(G)$  to refer to the vertex and edge set of  $G$ , when convenient. All graphs we consider are finite, simple, and undirected. For  $U \subseteq V$  we denote by  $G - U$  the graph obtained from  $G$  by removing the vertices in  $U$  and their incident edges. A set  $U \subseteq V$  is an *independent set* of  $G$  if  $\{u, v\} \notin E$  for any  $u, v \in U$ . The *symmetric difference* of two sets  $U$  and  $U'$  is  $U \Delta U' := (U_1 \setminus U_2) \cup (U_2 \setminus U_1)$ . A set  $U \subseteq V$  is a *vertex cover* of  $G$  if every edge in  $E$  is incident with a vertex in  $U$ . The minimum cardinality of a vertex cover of  $G$  is denoted by  $\text{vc}(G)$ . A set  $U \subseteq V$  is a *feedback vertex set* if  $G - U$  is acyclic (a *forest*). The minimum cardinality of a feedback vertex set of  $G$  is denoted  $\text{fvs}(G)$ . For a vertex  $v$ , denote by  $N_G(v)$  the set of its neighbors (excluding  $v$  itself). The *neighborhood* of a set  $U \subseteq V$  is  $N_G(U) := \bigcup_{s \in U} N_G(s) \setminus U$ . We omit the subscript when it is clear from the context. A graph  $G' = (V', E')$  is said to be a *subgraph* of  $G$ , if  $V' \subseteq V$ , and  $E' \subseteq E$ . It is an *induced subgraph* of  $G$  if  $V' \subseteq V$  and for any  $u, v \in V'$  we have  $\{u, v\} \in E$  if and only if  $\{u, v\} \in E'$ . The subgraph of  $G$  induced by  $U \subseteq V$  is denoted  $G[U]$ . A *graph class* is a (possibly infinite) collection of graphs. A graph class  $\Pi$  is said to be *hereditary* if given any graph  $G \in \Pi$ , any induced subgraph of  $G$  belongs to the class  $\Pi$  as well. A graph is *bipartite* if its vertex set can be partitioned into two independent sets  $I$  and  $J$ , which are also called the *partite sets*. We sometimes denote such a bipartite graph by  $G = (I \cup J, E)$ . A bipartite graph is *balanced* if  $|I| = |J|$ . A *matching* is a set of edges that do not share any endpoints. A matching is *perfect* if it spans the entire vertex set. A vertex  $v$  is a *cutvertex* in graph  $G$  if the removal of  $v$  increases the number of connected components. A *biconnected component* of  $G$  is a maximal connected subgraph of  $G$  that does not contain a cutvertex: removal of a single vertex from a biconnected component leaves the component connected.

We use the definitions of (nice) path decompositions as given in [7, §7.2]. For any path decomposition  $\mathcal{P} = (X_1, X_2, \dots, X_r)$  of  $G = (V, E)$ , and any vertex  $v \in V$ , define  $l_{\mathcal{P}}(v) = \min\{i : v \in X_i\}$  and  $r_{\mathcal{P}}(v) = \max\{i : v \in X_i\}$ , that is,  $l_{\mathcal{P}}(v)$  and  $r_{\mathcal{P}}(v)$  respectively denote the index of the first and last bag containing  $v$ . Note that if  $\mathcal{P}$  is nice, then  $l_{\mathcal{P}}(\cdot)$  and  $r_{\mathcal{P}}(\cdot)$  are injective maps over the set of vertices.

## 3 Definitions and Basic Facts for Reconfiguration

**Multiple Token Jump (MTJ).** Given any two independent sets  $I$  and  $J$ , with  $|I| = |J|$ , we say that  $I$  can be *k-MTJ reconfigured* to  $J$ , if there exists a finite sequence of independent sets  $(I = W_0, W_1, W_2, \dots, W_n, W_{n+1} = J)$  for some  $n \geq 0$ , such that for all  $i \in \{0, \dots, n+1\}$  the set  $W_i$  is an independent set,  $|W_i| = |I| = |J|$ , and  $|W_{i+1} \setminus W_i| \leq k$ . A step  $W_i \rightarrow W_{i+1}$  in the reconfiguration process with  $|W_i \setminus W_{i+1}| = k$  is called a *k-TJ move*. Given a graph  $G = (V, E)$ , define  $\text{MTJ}(G, s)$  as the minimum value of  $k$  such that any two independent sets of size  $s$  in  $G$  can be *k-MTJ reconfigured* to each other. Now define  $\text{MTJ}(G) := \max_{1 \leq s \leq |V|} \text{MTJ}(G, s)$ . Our goal is to characterize the value of  $\text{MTJ}(G)$  in terms of certain parameters of the graph  $G$ .

We call  $\text{MTJ}(G)$  the *MTJ reconfiguration threshold* of the graph  $G$ . The MTJ reconfiguration threshold of a graph class  $\Pi$  is defined as  $\text{MTJ}(\Pi) := \sup_{G \in \Pi} \text{MTJ}(G)$ .

**Token Addition Removal (TAR).** Given any two independent sets  $I$  and  $J$ , with  $|I| = |J|$ , we say that  $I$  can be *k-TAR reconfigured* to  $J$ , if there exists a finite sequence of independent sets  $(I = W_0, W_1, W_2, \dots, W_n, W_{n+1} = J)$  for some  $n \geq 0$ , such that  $W_i$  is an independent set,  $|I| - |W_i| \leq k$ , and  $|W_{i-1} \Delta W_i| \leq 1$  for all  $i \in \{0, \dots, n+1\}$ . We refer to the quantity  $B_i := |I| - |W_i|$  as the *buffer size* at step  $i$ : the tokens that were on the initial independent set and are not on the current independent set  $W_i$ , are placed in the buffer. Define  $\text{TAR}(G, s)$  to be the smallest buffer size  $k$  such that any two independent sets of size  $s$  can be *k-TAR reconfigured* to each other. Define  $\text{TAR}(G) := \max_{1 \leq s \leq |V|} \text{TAR}(G, s)$ . As before, we call  $\text{TAR}(G)$  the *TAR reconfiguration threshold* of the graph  $G$ , and extend the terminology to graph classes  $\Pi$  by defining  $\text{TAR}(\Pi) := \sup_{G \in \Pi} \text{TAR}(G)$ .

**Facts on Reconfiguration.** Observe that for any graph  $G$ , it holds that  $\text{MTJ}(G) = 1$  if and only if  $\text{TAR}(G) = 1$ . In general, the TAR reconfiguration threshold is at most the MTJ reconfiguration threshold. Indeed, each *k-TJ* move can be thought of as a sequence of  $2k$  steps with maximum buffer size  $k$ . First, sequentially remove the tokens of the  $k$  vertices from which we are jumping, placing their tokens in the buffer; then sequentially place the buffer tokens on the  $k$  new vertices in the independent set.

► **Proposition 1 (★).** *Let  $G$  be a graph with independent sets  $I$  and  $J$  of equal size. If  $I \setminus J$  can be *k-TAR reconfigured* (resp. *k-MTJ reconfigured*) to  $J \setminus I$  in the graph  $G[I \Delta J]$ , then  $I$  can be *k-TAR reconfigured* (resp. *k-MTJ reconfigured*) to  $J$  in  $G$ .*

Proposition 1 shows that to upper-bound the TAR or MTJ reconfiguration threshold, it suffices to do so in balanced bipartite graphs where the source and target configurations are disjoint; note that  $G[I \Delta J]$  is balanced bipartite and  $I \setminus J$  and  $J \setminus I$  are disjoint. We will frequently exploit this in our proofs. For any graph class  $\Pi$ , let  $\Pi_{\text{bip}}$  denote the set of bipartite graphs in  $\Pi$ . The following proposition shows that the reconfiguration threshold of a hereditary graph class is determined by the behavior of the bipartite graphs in the class. Note that for hereditary classes  $\Pi$ , the class  $\Pi_{\text{bip}}$  is hereditary as well.

► **Proposition 2 (★).** *For any hereditary graph class  $\Pi$ , we have  $\text{MTJ}(\Pi) = \text{MTJ}(\Pi_{\text{bip}})$  and  $\text{TAR}(\Pi) = \text{TAR}(\Pi_{\text{bip}})$ .*

## 4 Thresholds for Multiple Token Jump Reconfiguration

We start our discussion of token jump reconfiguration by recalling the following known result.

► **Theorem 3** ([18, Theorem 7]). *Let the graph  $G = (V, E)$  be a forest. Then  $\text{MTJ}(G) \leq 1$ .*

The intuition behind this result is that since a forest does not contain any cycle, one can start reconfiguring from the leaf nodes or the isolated vertices, each of which has at most one neighbor from the target configuration. For arbitrary graphs, the above procedure does not work since there may not be any leaves or isolated vertices. But if a graph  $G$  has a small vertex cover, then its MTJ reconfiguration threshold is again small.

► **Theorem 4 (★).** *Let  $G = (V, E)$  be a graph. Then  $\text{MTJ}(G) \leq \max(\text{vc}(G), 1)$ .*



An even cycle of length  $2n$  has MTJ reconfiguration threshold  $n$ . Since its vertex cover number is  $n$ , Theorem 4 is best-possible. Long cycles are not the only graphs whose MTJ reconfiguration threshold equals half the size of the vertex set. Bistable graphs (introduced below), of which the pumpkin structure defined in the introduction is a special case, also have this property. We bound the MTJ reconfiguration threshold of any graph  $G$ , in terms of the size of the largest induced bistable subgraph. The resulting bounds on the MTJ reconfiguration threshold are tight, but can be hard to apply to specific graph classes: it may be difficult to estimate the size of the largest induced bistable graph, or even to determine whether a given graph is bistable or not. We will therefore relate the size of the largest induced bistable subgraph to the size of the largest pumpkin subgraph. This will result in upper- and lower bounds on the MTJ reconfiguration threshold in terms of the largest pumpkin structure contained in the graph (class), which is arguably a more insightful parameter. The resulting bound will not be best-possible, however.

► **Definition 5** (Bistable graphs). A graph is called *bistable* if it is connected, bipartite, and has exactly two distinct maximum independent sets formed by the two partite sets in its unique bipartition. The *rank* of a bistable graph is defined as the size of its maximum independent sets.

Let  $\text{BI}(G)$  denote the rank of the largest induced bistable subgraph of  $G$ . If  $G$  contains no induced bistable subgraphs (which can only occur if  $G$  has no edges), then we define  $\text{BI}(G)$  to be one. For a graph class  $\Pi$  we define  $\text{BI}(\Pi) := \sup_{G \in \Pi} \text{BI}(G)$ .

The pumpkin shown in Figure 1a forms an example of a bistable graph. Lemma 6 connects bistable graphs to independent-set reconfiguration. Consider the task of reconfiguring the  $J$ -partite set to the  $I$ -partite set in a balanced bipartite graph  $G = (I \cup J, E)$ . If we have a set  $S \subseteq I$  such that  $|S| \geq |N(S)|$ , then one way to make progress in the reconfiguration is to select  $|S|$  vertices from  $N(S) \subseteq J$  and jump their tokens onto the vertices in  $S$ , resulting in a new independent set of the same size. The following lemma shows that when we consider a set  $S$  that is *minimal* with respect to being at least as large as its neighborhood, then the induced subgraph  $G[N[S]]$  is bistable. Hence the cost of such a jump of  $|S|$  vertices is bounded by  $\text{BI}(G)$ , which will allow us to bound the MTJ reconfiguration threshold.

► **Lemma 6** (★). *Let  $G = (I \cup J, E)$  be a balanced bipartite graph without isolated vertices and let  $S \subseteq I$  be inclusion-wise minimal with the properties that  $|S| \geq |N(S)|$  and  $S$  is not empty. Then  $G[N[S]]$  is bistable.*

The next lemma states two key properties of bistable graphs. They will later be useful to relate the quantities  $\text{PUM}(G)$  and  $\text{BI}(G)$ .

► **Lemma 7** (★). *Let  $G = (I \cup J, E)$  be a bistable graph. Then the following holds:*

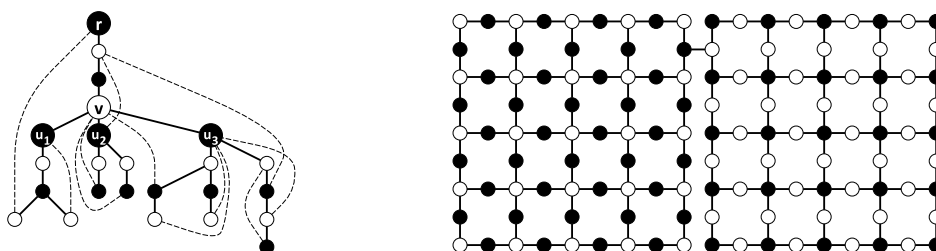
1.  $G$  has a perfect matching covering  $I$  (and hence  $J$ ).
2.  $G$  is biconnected.

► **Theorem 8** (★). *For any graph  $G$  it holds that  $\text{MTJ}(G) \leq \text{BI}(G)$ . Moreover, if  $G \neq K_1$ , then there exists an induced subgraph  $G'$  of  $G$  with  $\text{MTJ}(G') \geq \text{BI}(G) \geq \text{BI}(G')$ .*

The proof for the lower bound is straightforward, since by definition any induced bistable subgraph contains exactly two maximum independent sets. The upper bound follows by induction on the number of vertices in  $G$ , where in the induction step we make use of Lemma 6, and reconfigure the subgraph induced by the set  $N[S]$ .

The following corollary characterizes the MTJ reconfiguration threshold of hereditary graph classes. It follows easily from Theorem 8.





(a) DFS tree of a biconnected bipartite graph.

(b) Grid-like balanced bipartite graph with large treewidth and small TAR reconfiguration threshold.

■ **Figure 2** (2a) Depth-first search tree of a bipartite biconnected graph. Tree-edges are drawn solid, while the remaining edges of  $G$  are drawn with dotted lines. The three children  $u_1, u_2, u_3$  of  $v$  induce subtrees of types A, B, and C, respectively. (2b) Template for constructing graphs of large treewidth that can be TAR reconfigured with a buffer of size two. The treewidth is large due to the presence of a large grid minor.

► **Corollary 9** (★). *For any hereditary graph class  $\Pi \neq \{K_1\}$  it holds that  $\text{MTJ}(\Pi) = \text{BI}(\Pi)$ .*

We now formally introduce the pumpkin structure described in the introduction.

► **Definition 10** (Pumpkin). *A pumpkin is a graph consisting of two terminal vertices  $u$  and  $v$  linked by two or more vertex-disjoint paths with an odd number of edges, having no edges or vertices other than those on the paths. A path can consist of the single edge  $\{u, v\}$ . The size of the pumpkin is the total number of vertices.*

For a graph  $G$  we denote by  $\text{PUM}(G)$  the size of the largest (not necessarily induced) subgraph isomorphic to a pumpkin that is contained in  $G$ , or zero if  $G$  contains no pumpkin. For a graph class  $\Pi$  we define  $\text{PUM}(\Pi) := \sup_{G \in \Pi} \text{PUM}(G)$ .

The next theorem shows that the rank of the largest bistable induced subgraph of  $G$  can be upper-bounded in terms of the size of  $G$ 's largest pumpkin subgraph.

► **Theorem 11** (★). *For any bistable graph  $G$  we have  $\text{BI}(G) \leq f(\text{PUM}(G))$ , where the function  $f$  is defined as  $f(k) = (k^3 + k^2)^{k^2+1} + 1$ .*

**Proof sketch.** A bistable graph  $G$  is biconnected (Lemma 7). Biconnected graphs with a path of length more than  $L^2$  have a cycle of length more than  $L$  [11], which forms a pumpkin since  $G$  is bipartite. So if we set  $L := \text{PUM}(G)$ , graph  $G$  cannot have a path of length more than  $L^2$ . Consequently, if we build a DFS tree  $T$  rooted at an arbitrary vertex, its depth will be at most  $L^2$ . Next we claim that each vertex  $v$  in  $T$  has at most  $L^3 + L^2$  children, which in conjunction with the above bound on depth, yields the theorem. To prove the claim, we classify each child  $u$  of  $v$  into one of three types (Figure 2a) and prove that no type occurs often.

*Type A:* Some vertex in the subtree  $T_u$  rooted at  $u$ , has an edge in  $G$  to an ancestor  $w$  of  $v$  that does not belong to  $v$ 's partite set. Through each such subtree, we obtain an odd-length path from  $v$  to  $w$ . If there are more than  $L$  such paths from  $v$  to  $w$  then they form a pumpkin of size more than  $L$ , which gives a contradiction. Hence each ancestor of  $v$  receives an edge from less than  $L$  type-A child trees. There are  $\leq L^2$  ancestors, hence  $\leq L^3$  type-A children.

*Type B:* Vertex  $u$  is not of type A and the vertices in the subtree  $T_u$  are not evenly balanced over the two partite sets. Then any perfect matching in  $G$  (which exists by Lemma 7) matches

a vertex in  $T_u$  to an ancestor of  $v$ . Since the depth is at most  $L^2$ , while matching partners are all distinct, there are at most  $L^2$  type-B children.

*Type C:* Vertex  $u$  is not of type A and the vertices in the subtree  $T_u$  are evenly balanced over the two partite sets. If such a child exists, then we can build a maximum independent set that is not equal to either partite set in  $G$ : take the partite set that does not contain  $v$ , but replace its contents within  $T_u$  by the vertices from  $T_u$  in the other partite set. The case distinction ensures the result is a maximum independent set, contradicting bistability of  $G$ .

As this bounds the number of children of a vertex by  $L^3 + L^2$ , the theorem follows. ◀

The following theorem is our main result on the MTJ reconfiguration threshold. It bounds the MTJ reconfiguration threshold of a hereditary graph class  $\Pi$  in terms of the maximum size of a pumpkin subgraph of a graph in  $\Pi_{\text{bip}}$ , by combining Theorems 8 and 11. Recall that  $\Pi_{\text{bip}}$  contains the bipartite graphs in  $\Pi$ .

► **Theorem 12 (★).** *For any hereditary graph class  $\Pi$ , the following holds:*

$$g_1(\text{PUM}(\Pi_{\text{bip}})) \leq \text{MTJ}(\Pi) \leq g_2(\text{PUM}(\Pi_{\text{bip}})), \quad (1)$$

where  $g_1, g_2 : \mathbb{N} \rightarrow \mathbb{N}$  are positive non-decreasing functions defined as  $g_1(k) = k/2$  and  $g_2(k) = (k^3 + k^2)^{k^2+1} + 1$ . Moreover, for every graph  $G$  we have  $\text{MTJ}(G) \leq g_2(\text{PUM}(G))$ .

While the upper bound of Theorem 12 has room for improvement, the following proposition shows that the exponential dependency on the pumpkin size in the upper bound is unavoidable.

► **Proposition 13 (★).** *Let  $\Pi_{\text{PUM}}(k) := \{G : \text{PUM}(G) \leq k\}$  be the class of all graphs  $G$  whose largest pumpkin subgraph has size at most  $k$ . Then  $\text{MTJ}(\Pi_{\text{PUM}}(k)) = 2^{\Omega(k)}$ .*

## 5 Thresholds for Token Addition Removal Reconfiguration

In this section we study the model of token addition removal. First observe that when  $G$  is a forest, we have  $\text{MTJ}(G) \leq 1$  and therefore  $\text{TAR}(G) \leq 1$  as well. Also, from Theorem 4 we get  $\text{TAR}(G) \leq \max(\text{vc}(G), 1)$ . But the inequality  $\text{TAR}(G) \leq \text{MTJ}(G)$  tells us nothing about the behavior of the TAR reconfiguration threshold when the MTJ reconfiguration threshold is large. The next simple proposition immediately points towards this direction. Indeed, a large pumpkin (which has large MTJ reconfiguration threshold) can have a small feedback vertex set; this happens for even cycles, for example.

► **Proposition 14 (★).** *Let  $G = (V, E)$  be a graph. Then  $\text{TAR}(G) \leq \text{FVS}(G) + 1$ .*

The proof is fairly straightforward by noting that for any graph  $G$ , if the size of the minimum feedback vertex set is  $k$ , then by definition, deletion of  $k$  vertices leaves an acyclic subgraph. Hence, we can essentially apply Theorem 3. One can see that the above bound is tight, by considering the TAR reconfiguration threshold of a complete balanced bipartite graph. Indeed, for  $K_{n,n}$  the minimum size of a feedback vertex set is  $n - 1$ , and one can see that in order to include any one of the vertices of the target independent set, the reconfiguration must pass through the empty set. This shows that the TAR reconfiguration threshold is also  $n$ . As the main result of this section, we will show that the TAR reconfiguration threshold of a graph is also bounded in terms of its *pathwidth*. Before proving that statement, we present a structural lemma about path decompositions that will be useful in the proof.

► **Lemma 15 (★).** *Let  $G = (I \cup J, E)$  be a bipartite graph with a nice path decomposition  $\mathcal{P} = (X_1, \dots, X_r)$  of width  $k$ . Let  $S \subseteq J$  such that  $|N(S)| \leq |S|$  while no non-empty subset of  $S$  has this property. If we order the vertices in  $S$  as  $i_1, \dots, i_t$  such that  $r_{\mathcal{P}}(i_1) < r_{\mathcal{P}}(i_2) < \dots < r_{\mathcal{P}}(i_t)$ , then  $|N(\{i_1, \dots, i_{t'}\})| < t' + k$  for all  $1 \leq t' \leq t$ .*

Intuitively, the lemma says the following. Suppose a set  $S \subseteq J$  is inclusion-wise minimal with respect to being no smaller than its neighborhood. Then ordering  $S$  according to the right endpoints of the intervals representing  $S$  in the path decomposition, we are guaranteed that every prefix of  $S$  has a fairly small neighborhood compared to its size: the neighborhood size exceeds the size of the prefix by less than the pathwidth. Note that since the lemma deals with bipartite graphs only, no vertex of  $S$  can belong to the neighborhood of any prefix of  $S$ . The ordering of the vertices is uniquely defined since the path decomposition is nice. The bound of Lemma 15 is best-possible. Consider a complete bipartite graph  $K_{n,n}$ , with pathwidth  $n$ . In any optimal path decomposition, for  $t' = 1$  the first vertex in the ordering has a neighborhood of size  $n$  and so  $n < t' + n = 1 + n$ , but a better bound is not possible. Using Lemma 15 we bound the TAR reconfiguration threshold in terms of pathwidth.

► **Theorem 16.** *Let  $G = (V, E)$  be a graph. Then  $\text{TAR}(G) \leq \max(\text{PW}(G), 1)$ .*

**Proof.** We prove this theorem using induction on the number of vertices. By Proposition 1, it is enough to consider  $G = (V, E)$  and assume that the initial independent set  $I$  and target independent set  $J$  are such that  $|I| = |J|$ , and  $I \cup J = V$  and  $I \cap J = \emptyset$ . We will show that  $\text{PW}(G) \leq k$  implies that  $\text{TAR}(G) \leq k$ , using induction on the number of vertices  $n$ . For  $n = 1$ , the statement is trivially true. Now fix any  $k \geq 1$ , and assume the induction hypothesis that any graph  $G$  with  $n$  vertices satisfying  $\text{PW}(G) \leq k$  has  $\text{TAR}(G) \leq k$ .

Now let  $G$  be a graph of  $n + 1$  vertices having pathwidth at most  $k$ . Let  $S$  be an inclusion-minimal subset of  $J$  for which  $|S| \geq |N(S)|$ . Such a set exists since  $|J| = |I| \geq |N(J)|$ . We will show that if we reconfigure the set  $S$  in a suitable order by moving tokens from  $N(S)$  onto  $S$ , then the buffer size will not grow beyond  $k$ . There are enough vertices in  $S$  to accommodate all tokens on  $N(S)$ , and afterward we will invoke induction.

We first deal with a special case. If  $S = \{v\}$  is a singleton set, then it has degree at most one since  $|S| \geq |N(S)|$ . Move the token from the neighbor  $u$  of  $v$  (or from an arbitrary vertex  $u$ , if  $v$  has no neighbors) into the buffer, and then onto  $v$ . By induction there exists a TAR reconfiguration from  $I \setminus \{u\}$  to  $J \setminus \{v\}$  in  $G - \{u, v\}$  using a buffer of size at most  $\max(\text{PW}(G - \{u, v\}), 1) \leq \max(\text{PW}(G), 1)$ . When inserting the token move from  $u$  onto  $v$  at the beginning of this sequence, we get a TAR reconfiguration from  $I$  to  $J$  with the desired buffer size. In the remainder of the proof we can therefore assume  $|S| \geq 2$ . This implies that  $|S| = |N(S)|$ : if  $|S| > |N(S)|$  and  $|S| \geq 2$ , then we can remove a vertex  $v$  from  $S$  to obtain  $|S \setminus \{v\}| \geq |N(S \setminus \{v\})|$  for the nonempty set  $S \setminus \{v\}$ , contradicting minimality.

Let  $\mathcal{P} = (X_1, X_2, \dots, X_r)$  be a nice path decomposition of width at most  $k$ . If  $G$  has no edges, then  $S$  is a singleton set containing an isolated vertex. Since we already covered that case, we know  $G$  has at least one edge, so any path decomposition has width  $k \geq 1$ . Enumerate the vertices of  $S$  as  $i_1, \dots, i_m$  such that  $r_{\mathcal{P}}(i_1) < \dots < r_{\mathcal{P}}(i_m)$ . In other words, the vertices are ordered by increasing rightmost endpoint of the interval of bags containing it.

In order to describe the reconfiguration procedure we suitably group several TAR reconfiguration steps together as one step in the algorithm. In particular, one reconfiguration step in the algorithm described below will consist of a run of successive removals of nodes, followed by a single node addition.

We use the notion of a *buffer set*  $B_t$  at the  $t^{\text{th}}$  step of the reconfiguration, such that  $|B_t|$  will correspond to the number of tokens in the buffer at any particular time, and  $\max_t |B_t| + 1$  will correspond to the maximum buffer size of the corresponding TAR reconfiguration sequence. The buffer set is a subset of vertices, showing where the tokens in the buffer came from. At time step  $t = 0$ , define  $W_0 = I$  to be the independent set of vertices with a token, and let the buffer set  $B_0$  be empty. We will define intermediate independent sets  $W_i$  and buffer sets  $B_i$  representing the grouped reconfiguration steps. The algorithm stops

when  $W_m$  contains all vertices in  $S$ ; we will then invoke the induction hypothesis to finish the sequence. From the sequence  $(W_0, W_1, \dots, W_m)$  one obtains a formal reconfiguration sequence as defined in Section 3 by inserting “transitioning independent sets” in between  $W_i$  and  $W_{i+1}$  for all  $i$ . From  $W_i$ , repeatedly remove one vertex until arriving at  $W_{i+1} \setminus W_i$ , and then add the single vertex of  $W_{i+1} \setminus W_i$  to the resulting set.

For  $t \geq 1$ , the transition from  $t-1$  to  $t$  is obtained as follows. Let  $u_t$  be an arbitrary vertex from  $B_{t-1} \cup (N(i_t) \cap W_{t-1})$ . Intuitively, at step  $t$  we take the token from  $u_t$  (in the buffer set or on a neighbor of  $i_t$ ) and move it onto vertex  $i_t$ , causing  $u_t$  to disappear from the buffer and adding  $i_t$  to the independent set. To ensure the resulting set is independent, tokens on neighbors of  $i_t$  are moved into the buffer beforehand. Observe that the above step is valid only if  $B_{t-1} \cup (N(i_t) \cap W_{t-1})$  is nonempty. Below in Claim 17 we show that due to the choice of  $S$ , this is indeed the case for all  $t \leq m$ . Formally, we obtain the following:

► **Algorithm** (Reconfiguring graphs with small pathwidth). *Initialize with  $B_0 = \emptyset$  and  $W_0 = I$ . We now recursively define  $B_t$  and  $W_t$  for  $t \geq 1$ .*

1. *The neighbors of  $i_t$  that have tokens (that is, the neighbors that are in the current independent set) are removed from the previous independent set  $W_{t-1}$ , making room to add  $i_t$  to the new independent set:  $W_t = (W_{t-1} \setminus N(i_t)) \cup \{i_t\}$ .*
2. *The neighbors of  $i_t$  belonging to the previous independent set  $W_t$  move to the buffer, while  $u_t$  is removed from the buffer since its token has moved onto  $i_t$ :*

$$B_t = (B_{t-1} \cup (N(i_t) \cap W_{t-1})) \setminus \{u_t\}. \quad (2)$$

As mentioned earlier, a step from  $W_t$  to  $W_{t+1}$  can be thought as a sequence of successive removals of the nodes in  $N(i_{t+1}) \cap W_t$ , and then addition of the node  $i_{t+1}$ . During this successive TAR reconfiguration sequence corresponding to the step  $W_t$  to  $W_{t+1}$ , the maximum buffer size is given by  $|B_{t+1}| + 1$ , since the buffer size will be  $|B_{t-1} \cup (N(i_t) \cap W_{t-1})|$  just before the buffer token from  $u_t$  is moved onto  $i_t$ . Therefore, the maximum buffer size in the entire TAR reconfiguration sequence starting from  $W_0$  and ending at  $W_m$  is given by  $\max_{0 \leq t \leq m} |B_t| + 1$ . Also, at the end of the algorithm, all vertices from the set  $S$  will be in the independent set and no vertex in the buffer set. This can be seen as follows. Initially all tokens were on the vertices belonging to the set  $N(S) \subseteq I$ , since  $S \subseteq J$ . At each step of the algorithm, essentially one token is selected from  $N(S)$  as long as the number of such tokens is positive, and it is placed on some vertex in  $S$ . Now since  $|S| \geq |N(S)|$ , all the tokens in  $N(S)$  must eventually exhaust before the algorithm terminates placing one token at each vertex of  $S$ . For the validity of the above algorithm we claim the following, which in turn also characterizes the size of the buffer set at all intermediate time steps.

► **Claim 17.** *For all  $1 \leq t \leq m$  we have that  $B_{t-1} \cup (N(i_t) \cap W_{t-1})$  is nonempty, and that  $|B_t| = |N(\{i_1, \dots, i_t\})| - t$ .*

**Proof.** Suppose for a contradiction that there exists  $t' \leq m$ , such that  $B_{t'-1} \cup (N(i_{t'}) \cap W_{t'-1})$  is empty for the first time. If  $t' = 1$ , then  $B_{t'-1} \cup (N(i_{t'}) \cap W_{t'-1})$  is empty, and in particular  $N(i_{t'}) = \emptyset$ , so that  $i_{t'} = i_1$  is an isolated vertex. But since  $|S| \geq 2$  by our argument above, it follows that  $S' = \{i_1\}$  is a nonempty strict subset with  $|S'| \geq |N(S')|$ ; a contradiction. So in the remainder we consider  $t' > 1$ . We show that, for all  $t < t'$ ,  $|B_t| = |N(\{i_1, \dots, i_t\})| - t$ . Using this, we prove that  $2 \leq t' \leq m$  leads to a contradiction.

Observe that for any  $t < t'$ , after the  $t^{\text{th}}$  step of the algorithm, the total number of distinct vertices that have been added to the buffer set is given by  $|N(\{i_1, \dots, i_t\})|$ . Furthermore, for all  $t'' \leq t < t'$ , the set  $B_{t''-1} \cup (N(i_{t''}) \cap W_{t''-1})$  has always been nonempty. This implies that at each step, precisely one token has been removed from the buffer, thus reducing the

size of the buffer set by moving a buffer token onto a vertex that is added to the independent set. Therefore, in total  $t$  times the size of the buffer set reduces by one. Since initially the buffer set was empty, for any  $t < t'$  we have  $|B_t| = |N(\{i_1, \dots, i_t\})| - t$ .

Since we have assumed that  $B_{t'-1} \cup (N(i_{t'}) \cap W_{t'-1})$  is empty, we know  $B_{t'-1}$  is empty, and therefore from the above argument  $|B_{t'-1}| = |N(\{i_1, \dots, i_{t'-1}\})| - (t' - 1) = 0$ .

Defining  $S' := \{i_1, \dots, i_{t'-1}\} \subsetneq S$ , we have  $|N(S')| \leq |S'|$ . Since  $t' \geq 2$  the set  $S'$  is nonempty, contradicting the minimality of  $S$ . This proves the first part of the claim. Since the buffer does not become empty until after step  $t$ , the given argument then also proves the second part of the claim.  $\blacktriangleleft$

Note that in particular  $|B_m| = |N(\{i_1, \dots, i_m\})| - m = |N(S)| - |S| = 0$ ; the buffer empties for the first time only after reconfiguring the whole set.

It remains to show that throughout the process the buffer size will not grow beyond  $k$ , i.e.  $|B_t| \leq k - 1$ , for all  $t \leq m$ . Claim 17 (ii) implies that  $\max_{t \leq m} |B_t| \geq k$  if and only if there is a  $t$ , with  $t \leq m$ , such that  $|N(\{i_1, \dots, i_t\})| - t \geq k$ . But this is not possible due to Lemma 15. Hence, throughout the algorithm the buffer size will never exceed  $k$ .

Since the buffer set empties out after reconfiguring the set  $S$ , after the execution of the algorithm we have  $W_m \cap J = S$  and  $W_m \cap I \subset V \setminus (S \cup N(S))$ . Now define  $G' := G - (S \cup N(S))$ , and  $I' := I \cap W_m$  and  $J' := J \setminus S$ . Observe that  $G'$  has pathwidth at most  $k$ , and  $|I'| = |I \cap W_m| = |I| - |S| = |J'|$ . Furthermore, since  $S$  is non-empty,  $|V(G')| \leq n$ . By the induction hypothesis, there exists a TAR reconfiguration sequence from  $I'$  to  $J'$  in  $G'$  using a buffer of size at most  $k$ . Since  $N(S)$  is not in  $G'$ , any independent set in  $G'$  remains to be an independent set in  $G$  when augmented with the set  $S$ . Therefore we can first apply the given reconfiguration from  $N(S)$  to  $S$ , followed by the reconfiguration from  $I'$  to  $J'$ , to reconfigure  $I$  to  $J$  with a buffer of size at most  $k$ .  $\blacktriangleleft$

Observe by considering a complete balanced bipartite graph on  $2n$  vertices  $K_{n,n}$ , that in general the above bound is tight. Indeed,  $K_{n,n}$  has pathwidth equal to  $n$  [3], and as explained earlier, the TAR reconfiguration threshold is also  $n$ . Having proved Theorem 16, it is natural to ask whether pathwidth in some sense characterizes the TAR reconfiguration threshold: does large pathwidth of a graph imply that its TAR reconfiguration threshold is large? This is not the case: the pathwidth of a complete binary tree is proportional to its depth [20], but its reconfiguration threshold is 1 by Theorem 3. We now identify a graph structure which forces the TAR reconfiguration threshold to be large. First we formally introduce the special type of minor, illustrated in Figure 1b.

**► Definition 18** (Bipartite topological double minor). Let  $G = (I \cup J, E)$  be a bipartite graph and let  $H$  be an arbitrary graph. Then  $H$  is a *bipartite topological double minor* of  $G$ , if one can assign to every  $v \in V(H)$  a subgraph  $\varphi(v)$  of  $G$ , which is either an edge or an even cycle in  $G$ , and one can assign to each edge  $e = \{u, v\} \in E(H)$  a pair of odd-length paths  $\psi_1(e)$ ,  $\psi_2(e)$  in  $G$ , such that the following holds:

- For any  $u, v \in V(H)$  with  $u \neq v$  the subgraphs  $\varphi(u)$  and  $\varphi(v)$  are vertex-disjoint.
- For any  $v \in V(H)$  no vertex of  $\varphi(v)$  occurs as an interior vertex of a path  $\psi_1(e)$  or  $\psi_2(e)$ , for any  $e \in E(H)$ .
- For any  $e, e' \in E(H)$  the paths  $\psi_1(e)$  and  $\psi_2(e')$  are internally vertex-disjoint.
- For any  $e = \{u, v\} \in E(H)$  the paths  $\psi_1(e)$  and  $\psi_2(e)$  both have one endpoint in  $\varphi(v)$  and one endpoint in  $\varphi(u)$ .
- For any  $v \in V(H)$  and edge  $\{u, v\} \in E(H)$ , the attachment points of  $\psi_1(e)$  and  $\psi_2(e)$  in  $\varphi(v)$  belong to different partite sets.

The triple  $(\varphi, \psi_1, \psi_2)$  is a *BTD-minor model* of  $H$  in  $G$ . For an edge  $e \in E(H)$  we define  $\psi'_1(e), \psi'_2(e) \subseteq V(G)$  as the *interior* vertices of the paths  $\psi_1(e)$  and  $\psi_2(e)$ . Note that we can have  $\psi'_1(e) = \emptyset$  (and, similarly,  $\psi'_2(e) = \emptyset$ ) when  $\psi_1(e)$  (resp.  $\psi_2(e)$ ) consists of a single edge.

Intuitively,  $H$  occurs as a bipartite topological double minor (or *BTD-minor*) if each vertex of  $H$  can be realized by an edge or even cycle, and every edge of  $H$  can be realized by two odd-length paths that connect an  $I$ -vertex of  $\varphi(v)$  to a  $J$ -vertex of  $\varphi(u)$  and the other way around, in such a way that these structures are vertex-disjoint except for the attachment of paths to cycles. The definition easily extends to bipartite graphs whose bipartition is not given, since a BTD-minor is contained within a single connected component of the graph, which has a unique bipartition.

► **Proposition 19 (★)**. *Let  $G = (I \cup J, E)$  be a bipartite graph having a connected graph  $H$  as a BTD-minor model  $(\varphi, \psi_1, \psi_2)$ , such that each vertex of  $G$  is in the image of  $\varphi$ ,  $\psi_1$ , or  $\psi_2$ . Then  $G$  has a perfect matching with  $|I| = |J|$  edges, and for any independent set  $W$  in  $G$ :*

1. *For each vertex  $v$  of  $H$  we have  $|W \cap \varphi(v)| \leq |\varphi(v)|/2$ .*
  2. *For each edge  $e$  of  $H$  and  $i \in \{1, 2\}$  we have  $|W \cap \psi'_i(e)| \leq |\psi'_i(e)|/2$ .*
- For a maximum independent set  $W$ , equality holds in all cases.*

For a bipartite graph  $G$ , let  $\text{TREEMINOR}(G)$  denote the largest integer  $k$  for which  $G$  contains a complete binary tree of depth  $k$  as a BTD-minor. For a class  $\Pi$  of bipartite graphs we define  $\text{TREEMINOR}(\Pi) := \sup_{G \in \Pi} \text{TREEMINOR}(G)$ .

► **Theorem 20**. *There exists a real constant  $c > 0$  such that any hereditary graph class  $\Pi$  satisfies  $\text{TAR}(\Pi) \geq c \cdot \text{TREEMINOR}(\Pi_{\text{bip}})$ .*

**Proof.** As before, we consider a balanced bipartite graph  $G \in \Pi_{\text{bip}}$  with bipartition  $V(G) = I \cup J$  that has a complete binary tree  $T$  of depth  $d$  as a BTD-minor. Since the graph class is hereditary, for the lower bound we consider only the subgraph of  $G$  induced by  $\bigcup_{v \in V(T)} \{\varphi(v)\} \cup \left( \bigcup_{e \in E(T)} \{\psi_1(e) \cup \psi_2(e)\} \right)$ . With a slight abuse of notation we denote this subgraph by  $G$  from now on.

► **Fact 21** ([2]). *There is a universal constant  $c_1 > 0$  such that if  $T$  is a complete binary tree of depth  $d$ , then  $\max_{1 \leq i \leq |V(T)|} \min_{S \subseteq V(T); |S|=i} |N_T(S)| \geq c_1 \cdot d$ .*

The above implies that there exists  $i_0 \leq |V(T)|$ , such that any size- $i_0$  subset of  $V(T)$  has a neighborhood of size at least  $c_1 \cdot d$ . Let  $I \cup J$  be the unique bipartition of the connected graph  $G$ , and consider an arbitrary TAR reconfiguration sequence from  $I$  and  $J$ . In this sequence  $(I = W_0, W_1, \dots, W_t = J)$  of independent sets in  $G$ , look at the reconfiguration step when for the first time there exists  $S \subseteq V(T)$  with  $|S| = i_0$ , such that the intermediate independent set  $W$  at that step contains  $\bigcup_{v \in S} (\varphi(v) \cap J)$ , and for all  $v \notin S$  it satisfies  $(\varphi(v) \cap W \cap J) \subsetneq (\varphi(v) \cap J)$ . We will prove that  $|J| - |W| \geq c_1 \cdot d$ , implying that from the initial independent set of  $|I| = |J|$  tokens, at least  $c_1 \cdot d$  tokens must reside in the buffer.

To prove the theorem, consider the intermediate independent set  $W$ , and the set  $S \subseteq V(T)$  with  $|S| = i_0$  satisfying the above criteria. The following claim shows that for each vertex in  $N_T(S)$ , the independent set  $W$  uses at least one vertex fewer than the maximum independent set  $J$  does.

► **Claim 22**. *Consider an edge  $e = \{u, v\} \in E(T)$  with  $u \in S$  and  $v \notin S$ , and let  $Q_{e,v} \subseteq V(G)$  denote the vertices in  $\varphi(v) \cup \psi'_1(e) \cup \psi'_2(e)$ . The following holds:*

$$|W \cap Q_{e,v}| < |J \cap Q_{e,v}| = \frac{|Q_{e,v}|}{2}. \quad (3)$$



**Proof.** By Proposition 19, the maximum independent set  $J$  contains exactly half the vertices of  $Q_{e,v}$ . If  $|W \cap \psi'_i(e)| < |\psi'_i(e)|/2$  for some  $i \in \{1, 2\}$ , then we are done: by Proposition 19 the set  $W$  contains fewer vertices from  $\psi'_i(e)$  than the maximum independent set  $J$  does, and this cannot be compensated within the other parts of the structure since  $J$  contains half the vertices there and no independent set contains more. In the remainder, we can assume that  $W$  contains exactly half the vertices from  $\psi'_1(e)$  and  $\psi'_2(e)$ . Then the following are true:

- (i) All  $J$ -nodes of  $\varphi(u)$  are in  $W$  (by our choice of  $W$  and since  $u \in S$ ).
- (ii) Some  $J$ -node of  $\varphi(v)$  is not in  $W$  (by our choice of  $W$  and since  $v \notin S$ ).
- (iii) Some  $I$ -node of  $\varphi(v)$  is not in  $W$ . To see this, let  $i \in \{1, 2\}$  such that  $\psi_i(e)$  is an odd-length path from a  $J$ -node in  $\varphi(u)$  to an  $I$ -node in  $\varphi(v)$ , which exists by Definition 18, and orient it in that direction. Since the first vertex on the path is a  $J$ -node in  $\varphi(u)$ , it is contained in  $W$  as shown above. Hence the second vertex on the path, the first interior vertex, is not in  $W$ . Since exactly half the interior vertices from  $\psi_i(e)$  belong to  $W$ , every other interior vertex from  $\psi_i(e)$  is in  $W$ . Since the path has an even number of interior vertices and the first interior vertex is not in  $W$ , the last interior vertex must be in  $W$ . But this prevents its  $I$ -node neighbor in  $\varphi(v)$  from being in  $W$ .

Therefore, since  $\varphi(v)$  is either an edge or an even cycle, we have  $|W \cap \varphi(v)| < |\varphi(v)|/2$  by observing the following: the only independent sets in  $\varphi(v)$  of size  $|\varphi(v)|/2$  are  $\varphi(v) \cap I$  and  $\varphi(v) \cap J$ , but  $\varphi(v) \cap W$  is not equal to either of these sets since it avoids a  $J$ -node and an  $I$ -node. Hence  $|W \cap \varphi(v)| < |\varphi(v)|/2 = |J \cap \varphi(v)|$ , and Proposition 19 shows that this cannot be compensated in other parts of the minor model, implying  $|W \cap Q_{e,v}| < |J \cap Q_{e,v}|$ . ◀

Using Claim 22 we now finish the proof of Theorem 20. For each  $v \in N_T(S)$ , pick an edge  $e = \{u, v\}$  such that  $u \in S$ . By Claim 22 the set  $W$  contains less than half the vertices of  $Q_{e,v}$ , while the maximum independent set  $J$  contains exactly half. Note that the sets  $Q_{e,v}$  considered for different vertices  $v \in N_T(S)$  are disjoint, while Proposition 19 shows that from the other pieces of the minor model  $W$  cannot use more vertices than  $J$  does. It follows that  $|W| \leq |J| - |N_T(S)| \leq |J| - c_1 \cdot d$ . Hence the buffer contains at least  $c_1 \cdot d$  tokens. ◀

## 6 Conclusion

We considered two types of reconfiguration rules for independent set, involving simultaneously jumping tokens and reconfiguration with a buffer. For both models, we derived tight bounds on their reconfiguration thresholds in terms of several graph parameters like the minimum vertex cover size, the minimum feedback vertex set size, and the pathwidth. Many results in the literature concerning the parameter pathwidth can be extended to hold for the parameter treewidth as well. This is not the case here; the upper bound on the TAR reconfiguration threshold in terms of pathwidth (Theorem 16) cannot be strengthened to treewidth, since one can make arbitrarily deep complete binary trees as BTD-minors in bipartite graphs of treewidth only two (see Figure 1b). On the other hand, there are bipartite graphs of large treewidth with TAR reconfiguration threshold two (Figure 2b). To characterize the TAR reconfiguration threshold one therefore needs to combine graph connectivity (as measured by the width parameters) with notions that constrain the parity of the connections in the graph. This is precisely why we introduced BTD-minors. We conjecture that the converse of Theorem 20 holds, in the sense that any hereditary graph class having a large TAR reconfiguration threshold must contain a graph having a complete binary tree of large depth as a BTD-minor. Our belief is based partially on the fact that a BTD-minor model of a deep

complete binary tree is arguably the simplest graph of large pathwidth and feedback vertex number. Resolving this conjecture is our main open problem.

---

## References

- 1 Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. Efficient multi-robot motion planning for unlabeled discs in simple polygons. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pages 1–17, 2015. doi:10.1007/978-3-319-16595-0\_1.
- 2 B. V. Subramanya Bharadwaj and L. Sunil Chandran. Bounds on isoperimetric values of trees. *Discrete Mathematics*, 309(4):834–842, 2009. doi:10.1016/j.disc.2008.01.021.
- 3 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.
- 4 Paul Bonsma. Independent set reconfiguration in cographs. In *Graph-Theoretic Concepts in Computer Science: 40th International Workshop, WG 2014, Nouan-le-Fuzelier, France, June 25-27, 2014. Revised Selected Papers*, pages 105–116, 2014. doi:10.1007/978-3-319-12340-0\_9.
- 5 Paul Bonsma, Marcin Kamiński, and Marcin Wrochna. Reconfiguring independent sets in claw-free graphs. In *Algorithm Theory – SWAT 2014: 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, pages 86–97, 2014. doi:10.1007/978-3-319-08404-6\_8.
- 6 Gruiă Calinescu, Adrian Dumitrescu, and János Pach. Reconfigurations in graphs and grids. In *LATIN 2006: Theoretical Informatics: 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006. Proceedings*, pages 262–273, 2006. doi:10.1007/11682462\_27.
- 7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer Publishing Company, Incorporated, 2015. doi:10.1007/978-3-319-21275-3.
- 8 Mark de Berg, Bart M. P. Jansen, and Debankur Mukherjee. Independent set reconfiguration thresholds of hereditary graph classes. *arXiv:1610.03766*, 2016. arXiv:1610.03766.
- 9 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theoretical Computer Science*, 600:132–142, 2015. doi:10.1016/j.tcs.2015.07.037.
- 10 Reinhard Diestel. *Graph theory*. Springer-Verlag Berlin Heidelberg, 2010.
- 11 G. A. Dirac. Some theorems on abstract graphs. *Proceedings of the London Mathematical Society*, s3-2(1):69–81, 1952. doi:10.1112/plms/s3-2.1.69.
- 12 Adrian Dumitrescu and János Pach. Pushing squares around. *Graphs and Combinatorics*, 22(1):37–50, 2006. doi:10.1007/s00373-005-0640-1.
- 13 Eli Fox-Epstein, Duc A. Hoang, Yota Otachi, and Ryuhei Uehara. Sliding token on bipartite permutation graphs. In *Algorithms and Computation: 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, pages 237–247, 2015. doi:10.1007/978-3-662-48971-0\_21.
- 14 Gilad Goralý and Refael Hassin. Multi-color pebble motion on graphs. *Algorithmica*, 58(3):610–636, 2010. doi:10.1007/s00453-009-9290-7.
- 15 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.
- 16 Takehiro Ito, Erik D. Demaine, Nicholas J.A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.



- 17 Libin Jiang, Mathieu Leconte, Jian Ni, R. Srikant, and Jean Walrand. Fast mixing of parallel Glauber dynamics and low-delay CSMA scheduling. *IEEE Transactions on Information Theory*, 58(10):6541–6555, 2012. doi:10.1109/TIT.2012.2204032.
- 18 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.
- 19 Prashant Kataria and Rajarshi Roy. Parallel Glauber dynamics-based scheduling in static wireless grid networks with polynomially fading interference. *Electronics Letters*, 51(23):1948–1950, 2015. doi:10.1049/el.2014.3385.
- 20 Nancy G. Kinnersley and Michael A. Langston. Obstruction set isolation for the gate matrix layout problem. *Discrete Applied Mathematics*, 54(2-3):169–213, 1994. doi:10.1016/0166-218X(94)90021-3.
- 21 Daniel Lokshtanov, Amer E. Mouawad, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. In *Algorithms and Data Structures – 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, pages 506–517, 2015. doi:10.1007/978-3-319-21840-3\_42.
- 22 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. In *Parameterized and Exact Computation: 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, pages 281–294, 2013. doi:10.1007/978-3-319-03898-8\_24.
- 23 Francesca R. Nardi, Alessandro Zocca, and Sem C. Borst. Hitting time asymptotics for hard-core interactions on grids. *Journal of Statistical Physics*, 162(2):522–576, 2016. doi:10.1007/s10955-015-1391-x.
- 24 Marcin Wrochna. Reconfiguration in bounded bandwidth and treedepth. *arXiv:1405.0847*, 2014. arXiv:1405.0847.



# LZ77 Factorisation of Trees\*

Paweł Gawrychowski<sup>1</sup> and Artur Jeż<sup>2</sup>

1 University of Haifa, Israel

2 University of Wrocław, Poland

---

## Abstract

We generalise the fundamental concept of LZ77 factorisation from strings to trees. A tree is represented as a collection of edge-disjoint fragments that either consist of one node or has already occurred earlier (in the BFS order). Similarly as for strings, such a collection uniquely determines the tree, so by minimising the number of fragments we obtain a compressed representation of the tree. We show that our generalisation has several useful properties of the standard LZ77 factorisation: it can be computed in polynomial time and its simpler variant in linear time; its size is not larger than the smallest grammar for a tree; it can be transformed (in linear time) into a tree grammar of size  $\mathcal{O}(rg \log(n/(rg)))$ , where  $n$  is the size of the tree,  $g$  the size of the smallest grammar for this tree and  $r$  the maximal arity of the nodes in the tree, which matches a recent bound of Jeż and Lohrey [STACS 2014], but with a simpler and more modular proof.

**1998 ACM Subject Classification** F.4.2 Grammars and Other Rewriting Systems, F.2.2 Nonnumerical Algorithms and Problems, E.4 Data compaction and compression

**Keywords and phrases** Tree grammars, Grammar compression, LZ77, SLP, Tree compression

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.35

## 1 Introduction

Many popular compression text methods are based on the concept of block compression, where the input text is divided into blocks, each block being either a single letter or a longer block defined using the already compressed prefix (e.g. its arbitrary substring). Examples of such methods include LZ77, LZ78 and LZW algorithms. Another closely connected and particularly clean method is grammar compression, where we represent the input text as a CFG deriving exactly one word. Such a grammar is usually called a straight-line program (SLP). It is known that grammar compression captures most if not all block compression methods with small or no overhead in the size of the representation, yet it is much easier to operate on. This makes it particularly attractive given that the compressed data is often not only stored, but also accessed and processed on demand, and decompressing it defeats the purpose of having a small compressed representation in the first place. Thus, a recent trend is a design of algorithms working directly on a compressed representation of the data. Because grammar compression has an inductive definition, it is well-suited for such processing. In fact, similar algorithms for block compression routinely transform the the compressed representation into an equivalent SLP and only afterwards process it. The connection between grammar and block compression is used also in the other direction, for instance: computing a smallest SLP is NP-hard [8, 29, 7], but one can transform the LZ77 representation into an SLP, in this way we obtain an approximation algorithm with currently best known approximation ratio [25, 8, 19].

---

\* This work was supported under National Science Centre, Poland project number 2014/15/B/ST6/00615.



However, often the data has a more complex structure than a one-dimensional text. In particular, it is often the case that it has a natural hierarchical structure, which can be represented as a tree. Such a tree can be of course flattened and then treated as the input text, but this disregards the structure of the original data and hence might make processing the compressed representation (provably) difficult [10]. Hence it is desirable to design compression methods that retain the tree structure.

The folklore tree compression preserving the tree structure are directed acyclic graphs (DAGs), in which identical subtrees are shared. They yield up to exponential compression, processing DAGs is usually easy. The sharing mechanism of DAGs is limited, and it does not generalise the SLP: text  $a^n$ , when treated as a tree, is incompressible using DAGs. Tree SLP (*TSLP*) are a generalisation of SLP to the case of trees, which allow sharing of more general substructures. Moreover, DAG are subclass of TSLP and TSLP can be exponentially smaller than DAG. As in the case of SLP [8, 29], determining the size of the smallest TSLP is NP-hard, but there are approximation algorithms for this problem [17, 20] as well as heuristical ones [23, 5, 6].

TSLP became a *de facto* standard of tree compression in computer science, as several types of queries can be computed on TSLP in polynomial time [24] and there are various applications of TSLPs in other subfields. Most notably, they were used in problems related to context unification [13, 14, 15, 21], culminating in the recent proof that context unification is in PSPACE [18]. Moreover, studying matching problems for TSLP became an area on its own [11, 12, 27]. Other applications of TSLP (and SLP) can be found in a recent survey [22].

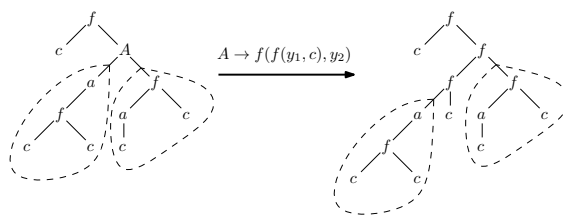
There are other approaches at tree grammar compression: Akutsu [2] generalised SLP to trees in a different (and incomparable) way. Bille et al. [3] proposed *top trees*, which are a variant of TSLP for unranked and unlabelled trees. Bojańczyk and Walukiewicz [4] proposed the model of forest algebras, which is designed specifically to handle unranked trees (while their model does not mention grammars but rather expressions, it can be easily trimmed to yield an TSLP-like formalism).

There were attempts at defining a variant of block compression designed for trees, but none of them became standard nor did they have ties to tree grammar compression. In particular, so far exploiting the connection between the grammar and block compression, so popular in text compression, did not occur for trees.

**Our contribution.** We propose a model of block compression for trees, inspired by both the notion of LZ77 and TSLPs. Similarly to LZ77, it represents the tree as a collection of trees (with “holes” that are placeholders for another trees), called *fragments*; each fragment is either an individual node or has already appeared earlier (in the BFS order) in the tree. Such factorisation can be compactly represented: each fragment is encoded by the BFS-addresses of the root and all the holes of its earlier occurrence. We give an  $\mathcal{O}(n^3)$  algorithm constructing the smallest such representation for the general case, and a specialised linear algorithm for the case when fragments have at most one hole. Such factorisations are related to TSLPs in a similar way as LZ77 are related to SLPs: it is not difficult to see each TSLP can be transformed into a factorisation without size increase. We prove that a factorisation can be transformed into a TSLP defining the same tree yielding an approximation algorithm for the smallest TSLP problem, the approximation ratio matches the best known [20].

## 2 Notions, definitions, basic results

**Trees.** We consider  $\Sigma$ -labeled rooted trees: The children of every node are ordered (by the usual left-to-right order) and each node is labeled with a letter from a fixed *ranked*



■ **Figure 1** A tree  $f(c, A(a(f(c, c)), f(a(c), c)))$  and the result of applying a rule  $A(y_1, y_2) \rightarrow f(f(y_1, c), y_2)$ .

alphabet  $\Sigma$ , i.e. every  $a \in \Sigma$  has arity  $\text{ar}(a)$ . The label of a node  $u$  is denoted by  $\text{label}(u)$  and determines its number of children  $\text{ar}(\text{label}(u))$ . By  $\text{children}(u)$  we denote the children of  $u$ , and by  $\text{parent}(u)$  its parent. We assume integer alphabet, that is,  $\Sigma = \{1, 2, \dots, n\}$ , where  $n$  is the size of the tree; thus, node labels can be sorted in  $\mathcal{O}(n)$  time using RadixSort. By  $r$  we denote the maximal arity of nodes in the tree. We often employ a BFS-order on the nodes of the tree, denoted by  $\leq_{\text{BFS}}$  or  $\leq$ , when this causes no confusion; it is a linear order. By *BFS-addresses* (or *addresses*) we denote the BFS-order numbers.

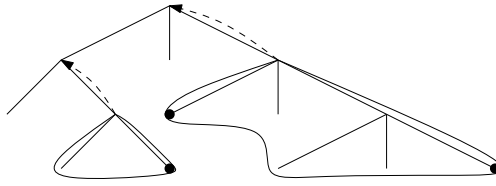
We consider also trees with *holes* that represent missing subtrees. The holes are represented by labels  $y, y_1, y_2, \dots$  from  $\mathbb{Y}$  that is disjoint with  $\Sigma$ . For the purposes of building trees holes have arity 0. Labels of different holes are different. As holes represent missing trees, we can substitute trees (or with trees with holes) for them: we delete the appropriate hole and replace it with the appropriate tree.

**LZ77.** As we model our tree factorisation on the LZ77 factorisations, let us recall the notion of LZ77 for strings: An LZ77 factorisation of a word  $w$  is a representation  $w = f_1 f_2 \dots f_\ell$ , where each  $f_i$  (called *phrase*) is either a single letter or it already occurred in the text, formally:  $f_i = w[j \dots j + |f_i| - 1]$  for some  $j \leq |f_1 \dots f_{i-1}|$ . If for some  $f_i$  as in the latter case it holds that  $j + |f_i| > |f_1 \dots f_{i-1}|$  (informally, the whole phrase occurred earlier), this factorisation is called *self-referencing*; it is called *non-self-referencing* otherwise. The self-referencing factorisations are in some cases more succinct than the non-self referencing ones, with the trivial example being string  $a^k$ . Each  $f_i$  can be represented as a single letter or as a pair  $(j, |f_i|)$ , where  $j$  is as in the explanation above, yielding a *compressed representation* of a LZ77 factorisation. The (compressed) *size* of a factorisation  $f_1 f_2 \dots f_\ell$  is  $\ell$ . Smallest LZ77 factorisation of a given text can be computed in linear time.

**Tree grammars.** Tree grammars extend CFG to trees: they have nonterminals (finite set  $N$ ), terminals (ranked alphabet  $\Sigma$ ), rules ( $P$ ) and a starting symbol ( $S$ ). Nonterminals generate trees with holes: each nonterminal  $A$  has an *arity*  $\text{ar}(A)$  and it generates a tree with  $\text{ar}(A)$  holes. Thus in the rules we treat  $A$  as  $\text{ar}(A)$ -ary function symbol and in the rule  $A \rightarrow t$  the tree  $t$  has  $\text{ar}(A)$  holes, labeled with  $y_1, \dots, y_{\text{ar}(A)}$ . To stress the dependence on holes, we sometimes write the rule as  $A(y_1, \dots, y_{\text{ar}(A)}) \rightarrow t$ .

We apply a rule  $A \rightarrow t$  as follows: given a tree  $s$  we can rewrite its subtree  $A(t_1, \dots, t_{\text{ar}(A)})$  by  $t$  with each  $y_i$  replaced by  $t_i$ . Intuitively, we replace an  $A$ -labeled node by  $t(y_1, \dots, y_{\text{ar}(A)})$  and identify the  $j$ -th child of  $A$  with the unique  $y_j$ -labeled node of  $t$ , see Fig. 1.

The *size* of a rule  $A(y_1, \dots, y_{\text{ar}(A)}) \rightarrow t$  is  $|t| - \text{ar}(A)$ , i.e. the number of non-holes nodes in  $t$ . This makes sense, as we can assume that BFS order of holes' labels in  $t$  is  $y_1, \dots, y_{\text{ar}(A)}$  and all of them occur in  $t$ , so we do not have to list them when describing  $t$ . The size of a tree grammar is the sum of sizes of its rules.



■ **Figure 2** BFS-factorisation: for simplicity, the node labels were suppressed, only one label of each arity is used. The fragments are enclosed in blobs, the holes are represented as full dots, other nodes are free. Note that nodes 7, 11, 14 are free nodes and holes in their fragments. The dashed arrows are definitions.

In *Tree SLP* (or *TSLP* for short) we additionally insist that for  $A \in N$  there is *exactly one* production  $A \rightarrow t$  and that nonterminals are linearly ordered such that  $S$  is the smallest nonterminal and if  $A \rightarrow t$  and  $B$  occurs in  $t$  then  $B > A$ . So, TSLP produces a unique tree.

**Factorisations of trees.** A *fragment*  $F$  is a connected subtree of a tree  $T$  in which each node *either* has the same label  $a$  as in  $T$  (and then has  $\text{ar}(a)$  children) *or* is labeled with some  $y \in \mathbb{Y}$ , in which case it is a leaf (in  $F$ ) and it is called a *hole*. The *arity* of a fragment is its total amount of holes, a fragment of arity  $k$  is called a *k-fragment*. We require that a fragment has at least 2 non-hole nodes. A *free node* is defined similarly as a fragment, but it has exactly one non-hole node, i.e. it has a non-hole root and all its children are holes. When referring to a free node rooted in  $v$  we call it simply  $v$ .

A *factorisation*  $\mathcal{F}$  of a tree  $T$ , denoted by  $(T, \mathcal{F})$ , is a collection of edge-disjoint fragments and free nodes, such that each node of  $T$  is either a non-hole node in some fragment or a free node. Such a factorisation is a *BFS-factorisation* if for each fragment  $F \in \mathcal{F}$ , which is rooted in  $v \in T$ , there is an isomorphic fragment  $F'$  of  $T$ , called the *definition* of  $F$ , rooted in  $v' <_{\text{BFS}} v$ ; note that  $F'$  is not necessarily in  $\mathcal{F}$  and that  $F$  and  $F'$  may overlap. See Fig. 2 for an example. Neither a factorisation, nor the BFS-factorisation of a tree is unique, similarly to the LZ77 factorisation of a string.

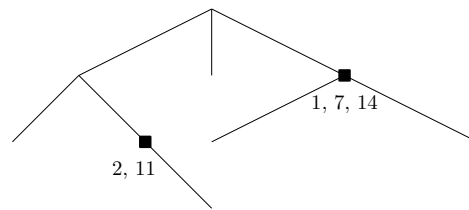
For  $v \in F$  the corresponding  $v' \in F'$  is the *definition* of  $v$ , denoted by  $\text{def}[v]$ , a link to this node is a *definition link*. The *arity* of a factorisation is the maximal arity of the fragments in it. An  $\ell$ -factorisation is a factorisation of arity at most  $\ell$ .

Note that nodes are shared, but in such a case in the “lower” fragment (or free node) this node is a root and in the “upper” a hole. In this way nodes in the tree can have two definitions: one as a definition of a hole and one as a definition of a root. We store only the latter but use a term “definition of a hole”, as it is useful in description and analysis.

We use the BFS order because the condition  $\text{def}[v] < v$  required for the root of the fragment is inherited by other nodes in this fragment. Thus, given a fragment, we can extend it by a node above the root (at the hole) as long as the labels are the same.

► **Lemma 1.** *Let  $F$  be a fragment in a BFS-factorisation of a tree,  $D$  a definition of  $F$  and  $u, v$  be two nodes in  $F$ . Then  $\text{def}[u] < u$  if and only if  $\text{def}[v] < v$ .*

In case of TSLP, the maximal arity of nonterminals in a grammar usually multiplicatively affects the running time and in general makes the reasoning about TSLP harder. Luckily, it is known that any TSLP can be converted to a one that uses only nonterminals of arity 0 and 1, with a polynomial size increase [24]. Similar result can be also obtained for the tree factorisations: a factorisation can be converted to a 1-factorisation, with a polynomial size increase:



■ **Figure 3** Compressed BFS-factorisation of the tree from Fig. 2. The fragments are depicted by square boxes, the addresses of their definitions and holes are beneath the boxes.

► **Lemma 2.** *In any factorisation an  $\ell$ -fragment can be (effectively) replaced with at most  $(\ell-1)(r-2)$  0-fragments,  $\ell-1$  free nodes and  $(2\ell-1)$  1-fragments. If the original factorisation was a BFS one, also the obtained one is.*

**Compressed representation.** The *compressed BFS-factorisation* for a BFS-factorisation  $(T, \mathcal{F})$  is given by a tree: each fragment and free node is represented as a single node; an edge between two fragments or free nodes means that they share a node in  $T$  (in other words: a hole in one of them is a root in the other). Each fragment-node stores addresses (in  $T$ ) of its root and holes. See Fig. 3.

A compressed BFS-factorisation is *valid* if it corresponds to some BFS-factorisation of a tree; not every compressed BFS-factorisation is valid, as it may use undefined addresses. Still, a valid compressed BFS-factorisation defines a unique tree, which can be computed by iteratively adding the nodes in the BFS order.

► **Lemma 3.** *A valid compressed BFS-factorisation corresponds to exactly one BFS-factorisation of a tree, which can be computed in  $\mathcal{O}(|T|)$  time, where  $T$  is the resulting tree.*

For the purpose of storing the compressed factorisation, we count the storage size of the factorisation in terms of memory cells rather than bits and assume that we can fit one address in  $\mathcal{O}(1)$  memory cells. Note that this is also the assumption for LZ77.

► **Lemma 4.** *The compressed BFS-factorisation of  $(T, \mathcal{F})$  uses  $\mathcal{O}(|\mathcal{F}|)$  memory cells.*

In the full version of this paper, we consider also *weighted size* of factorisations: the weight of an  $i$ -fragment is  $w_i$  and of a free node is 1; we assume  $1 \leq w_0 \leq w_1 \leq \dots$ . The (weighted) *size* of the factorisation is the sum of weights of its fragments. To streamline the presentation we do not consider the weights, though our algorithms work also in weighted case.

**Comparison with the text model.** A text can be seen as a tree using only nodes of arity 1, and the compressed BFS-factorisation of such a tree corresponds with a compressed LZ77 factorisation of this text: each LZ77 phrase can be extended to the right with an extra hole added at the end. The sizes and the descriptions of two such factorisations are similar.

In case of text we can distinguish between self-referencing and non-self-referencing factorisation. In case of trees we use only the self-referencing variant, as a clear and useful definition of a non-self-referencing one is elusive.

**Comparison with tree grammars.** In case of strings, the size of the LZ77 is a lower bound for the size of the grammar generating the same string [25, 8]; the same holds for trees; this observation is the first step in the approximation algorithm for the smallest tree grammar problem, presented in Section 4.



► **Lemma 5.** *The size of the smallest compressed BFS-factorisation for a tree  $T$  is not larger than the size of the smallest TSLP for  $T$ .*

### 3 Computing BFS-factorisations

In this section we first present a polynomial time dynamic programming algorithm **SimpleFactors**, which computes an optimal BFS-factorisation (of unbounded arity). Then, we show how to modify it, so that it computes optimal  $k$ -BFS-factorisations. Finally, using additional insight, we give a specialised algorithm for 1-BFS-factorisations that runs in linear time. We focus on describing how to calculate the number of fragments in an optimal BFS-factorisation, reconstructing the corresponding factorisation is straightforward.

**Unbounded arity.** Let  $T_u$  denote the tree rooted in  $u$  and  $opt(u)$  the number of fragments in an optimal factorisation of  $T_u$  into fragments occurring earlier in the whole tree  $T$  (so not necessarily in  $T_u$ ). To compute  $opt(u)$  **SimpleFactors** constructs a table  $T[u, v]$ , which stores the smallest size of a factorisation of  $T_u$ , such that  $v$  is the definition of  $u$  (or  $T[u, v] = +\infty$ , if  $label(u) \neq label(v)$ ). We fill  $T[u, v]$  in a reversed BFS order, thus when calculating  $T[u, v]$  the  $T[u', v']$  and  $opt(u')$  are known for every  $u' \in children(u)$  and  $v' < u'$ .

Let  $children(u) = (u_1, u_2, \dots, u_d)$  and  $children(v) = (v_1, v_2, \dots, v_d)$ . Then  $T[u, v]$  is 1 plus the number of fragments in subtrees rooted in  $u_i$ , for each  $i$ . If  $u_i$  is a hole this is  $opt(u_i)$ ; otherwise this is  $T[u_i, v_i] - 1$  (the ‘ $-1$ ’ is for the fragment rooted in  $u_i$  in  $T[u_i, v_i]$ , which is not counted in  $T[u, v]$ ). We take the minimum of those two numbers separately for each  $i$ . Thus  $T[u, v] = 1 + \sum_{i=1}^d \min(T[u_i, v_i] - 1, opt(u_i))$ . Similarly,  $opt(u) = \min(1 + \sum_{i=1}^d opt(u_i), \min_{v < u} T[u, v])$ , where the (outer) minimum represents the two possibilities: there is a free node or a fragment rooted in  $u$ .

► **Theorem 6.** *SimpleFactors computes the size of the smallest BFS-factorisation (of unbounded arity) using quadratic time and space.*

**Bounded arity.** We modify **SimpleFactors** into **BoundedFactors**, which computes smallest  $k$ -BFS-factorisation, by introducing another parameter. In particular, we again allow factorisations of  $T_u$  into fragments that occur earlier in  $T$ , not necessarily in  $T_u$ .

Let  $T[u, v, h]$  be the size of an optimal  $k$ -factorisation for  $T_u$  such that  $v < u$  is a definition of  $u$  and the fragment rooted in  $u$  has  $h$  holes. Let  $children(u) = (u_1, u_2, \dots, u_d)$  and  $children(v) = (v_1, v_2, \dots, v_d)$ . Any factorisation of  $T_u$ , when restricted to  $T_{u_i}$ , is either a free node with some fragments attached or an optimal factorisation of  $T_{u_i}$  of a cost  $T[u_i, v_i, h_i]$ ; moreover, the sum of  $h_i$  over all children is  $h$ , i.e.  $h_1 + h_2 + \dots + h_d = h$ , where  $h_i = 1$  if  $u_i$  is a hole. Thus **BoundedFactors** fills the table  $T[u, v, h]$  similarly as **SimpleFactors** the  $T[u, v]$  but computes an appropriate min-plus product of  $T[u_i, v_i, h_i] - 1$  instead of taking a simple minimum; then  $opt(u)$  is computed from  $T[u, v, h]$  as previously. A careful analysis yields  $\mathcal{O}(n^2 \min(k^2, n))$  bound on the running time.

► **Theorem 7.** *BoundedFactors computes the size of the smallest  $k$ -BFS-factorisation and runs in  $\mathcal{O}(n^2 \min(k^2, n))$  time and  $\mathcal{O}(n^2 k)$  space, where  $n$  is the size of the tree and  $k$  the maximal arity of fragments in the  $k$ -BFS-factorisation.*

### 1-BFS-factorisations

**BoundedFactors** runs in quadratic time in case of 1-BFS-factorisations. We now present a specialised version of **BoundedFactors**, called **1-Factors**, which achieves linear time.

► **Theorem 8.** *The algorithm 1-Factors runs in linear time and computes the size of the smallest 1-BFS-factorisation.*

**Outline.** Each factorisation of  $T_u$  has one of the following *types*: (F1) it is a single 0-fragment; (F2) the root  $u$  is a free node; (F3) there is a 1-fragment rooted in  $u$  such that its hole is a leaf. 1-Factors processes the nodes in a BFS-reversed order and for each node  $u$  computes for  $T_u$  the optimal factorisation of each type and takes the smallest among them.

Computing the F1 factorisations of all trees boils down to grouping the subtrees  $\{T_u\}_{u \in T}$  into isomorphism classes, which is explained in detail later on. F2 factorisation is easy to calculate, as we already know the optimal factorisation trees rooted in children of  $u$ . Lastly, we need to compute F3 factorisations only for nodes that do not have an F1 factorisation, as it is always smaller than F3.

► **Lemma 9.** *For every  $u$ , the size of F1 factorisations of  $T_u$  (if feasible) is at most the size of F3 factorisation of  $T_u$ .*

Call the nodes that have an F1 factorisation or are leaves *bottom part* and the remaining nodes the *top part*.

► **Lemma 10.** *Top part of the tree is a subtree rooted in the root of  $T$ ; it can be computed in linear time.*

Now, for F3 factorisations, we would like to also group 1-fragments into isomorphism classes, but there are  $\mathcal{O}(n^2)$  of them. Thus we limit their amount: we show that it is enough to consider maximal (in an appropriate sense) 1-fragments and prove that there are  $\mathcal{O}(n)$  of them. Furthermore, using additional insight we can compute in linear time a superset of all 1-factors used in an optimal 1-factorisation. Then it remains to calculate their actual cost, to see whether they should be used in the 1-factorisation. It turns out that those candidates are in some sense consecutive and we can process them in amortised constant time.

**F1 factorisations.** Two trees  $T_u$  and  $T_v$  are *isomorphic*, denoted by  $T_u \equiv T_v$ , if  $\text{label}(u) = \text{label}(v)$  and the subtrees rooted at the corresponding children of  $u$  and  $v$  are also isomorphic. Grouping subtrees of  $T$  into isomorphism classes in linear time is already folklore [1].

► **Lemma 11.** *Given a tree  $T$  we can calculate in total time  $\mathcal{O}(|T|)$  for every node  $u$ :*  
 (1) a number  $\text{id}(u) \in \{1, 2, \dots, |T|\}$ , such that  $T_u \equiv T_v \iff \text{id}(u) = \text{id}(v)$  (2) a node  $v < u$  such that  $\text{id}(u) = \text{id}(v)$ , whenever such  $v$  exists.

**F3 factorisations.** If  $F'$  is a definition of a fragment  $F$  then trees rooted in a node and its definition are mostly isomorphic: define for every node  $u$  a number  $\text{siblings-id}(u)$ , such that  $\text{siblings-id}(u) = \text{siblings-id}(v)$  if and only if the following conditions hold: (1)  $\text{label}(\text{parent}(u)) = \text{label}(\text{parent}(v))$ ; (2)  $u$  is the  $k$ -th child of  $\text{parent}(u)$  and  $v$  is the  $k$ -th child of  $\text{parent}(v)$ ; (3) for every  $i \in \{1, \dots, \text{ar}(\text{parent}(u))\} \setminus \{k\}$ , the trees rooted in  $i$ -th child of  $\text{parent}(u)$  and the  $i$ -th child of  $\text{parent}(v)$  are isomorphic. These can be efficiently computed, by appropriate grouping of nodes and its siblings by RadixSort.

► **Lemma 12.** *Given a tree  $T$  we can calculate in total  $\mathcal{O}(|T|)$  time for every node  $u$  its number  $\text{siblings-id}(u) \in \{1, 2, \dots, |T|\}$ .*

Consider a 1-fragment  $F$  rooted at  $u_1$  and a hole at  $u_{k+1}$ : the path from  $u_1$  to  $u_{k+1}$ , denoted as  $u_1, \dots, u_{k+1}$ , is the *spine* of  $F$ ; a spine is earlier than other if its hole is earlier.

Given a path  $u_1, \dots, u_{k+1}$  we call it a *spine* without explicitly mentioning the 1-fragment. The *length* of the spine  $u_1, \dots, u_{k+1}$  is  $k$ . A spine  $u_1, \dots, u_{k+1}$  is *isomorphic* to a spine  $v_1, \dots, v_{k+1}$  if and only if  $\text{siblings-id}(u_i) = \text{siblings-id}(v_i)$  for  $i = 2, \dots, k+1$ . Then 1-fragments are isomorphic if and only if their spines are. This characterisation is used to find isomorphism classes of 1-fragments. Given such classes, we can easily determine, whether a fragment  $F$  can be used in a BFS-factorisation (i.e. whether it has an earlier definition).

► **Lemma 13.** *1-fragments are isomorphic if and only if their spines are. In particular, a 1-fragment  $F$  can be used in a 1-BFS-factorisation if and only if there is an earlier spine isomorphic to the spine of  $F$ .*

A subsequence of a spine is also a spine, thus we extend the spines up as far as possible: a spine  $u_1, \dots, u_{k+1}$  is *up-maximal* if and only if it has an earlier isomorphic spine and  $\text{parent}(u_1), u_1, \dots, u_{k+1}$  does not. A 1-fragment is up-maximal when its spine is and 1-factorisation is up-maximal if each of its 1-fragment is. Any optimal 1-BFS-factorisation can be converted into an up-maximal one without increasing its size; thus we consider only up-maximal 1-factorisations.

Up-maximal spines can be compactly represented: for every node  $v$  let  $\text{spine}(v)$  be its highest ancestor such that the path starting at  $\text{spine}(v)$  and ending at  $v$  is an up-maximal spine; if there is no such an ancestor then  $\text{spine}(v) = v$ . All  $\text{spine}(v)$  and the lengths of the corresponding spines can be calculated in linear-time using a suffix tree of a tree [28].

► **Lemma 14.** *Given a tree  $T$ , we can find in  $\mathcal{O}(|T|)$  total time for every node  $v$  its  $\text{spine}(v)$  as well as the length of the spine beginning at  $\text{spine}(v)$  and a hole at  $v$ .*

Now, F3 factorisation are computed as follows: we process the tree in the reversed BFS order. When we are in a non-leaf node  $u$  we *activate* it, a node  $u$  is *deactivated* when we processed  $\text{spine}(u)$ . When we are in node  $u$  from a top part, we choose from its active descendent  $v$  with minimal  $\text{opt}(v)$ , if such  $v$  exists. We call such a query a minad (*minimal active descendent*) query, and denote by  $\text{minad}(u)$ . Then the cost of the smallest F3 factorisation of  $T_u$  is  $1 + \text{opt}(\text{minad}(u))$ , or undefined, if  $\text{minad}(u)$  is not defined. Note that  $\text{opt}(\text{minad}(u))$  is already known, as  $\text{minad}(u)$  is processed. The data structure presented below answers minad queries in amortised  $\mathcal{O}(1)$  time.

**Data structure for minad queries.** We exploit the structure of spines in the top part of the tree. Firstly, the top part can be represented as union of vertex disjoint paths: A *snake* originates in a node from a top part; if  $u$  is on a snake and it has a unique son  $u'$  that is not in a bottom part then  $u'$  is also on a snake; each snake is maximal: it cannot be extended neither up nor down. A spine of a node in a top part is contained in a single snake, perhaps except the hole.

► **Lemma 15.** *Snakes are vertex-disjoint paths, all snakes can be constructed in linear time.*

*Let  $u$  be in a top part and let  $T_u$  have an  $F_4$  factorisation. Then there are: an optimal up-maximal  $F_4$  factorisation of  $T_u$  and a snake  $u_1, u_2, \dots, u_s$  such that  $u = u_i$  and the hole of fragment rooted in  $u$  is either at some  $u_j$ , where  $j > i$ , or at a child of  $u_s$ .*

For a snake  $u_1, \dots, u_s$  we ask queries  $\text{minad}(u_s), \text{minad}(u_{s-1}), \dots, \text{minad}(u_1)$  in this order. Our data structures are constructed for a snake and updated when we process consecutive nodes of a snake. These data structures are called  $S1$  and  $S2$  later on, the former stores the active children of  $u_s$  and the other the active nodes from the snake. A node  $v$  is represented in these data structures as a pair  $(i, \text{opt}(v))$ , where  $\text{spine}(v) = u_i$ . These data structures

support three operations (on a set of elements  $S$ ):  $\text{insert}(x', y')$  inserts a new pair  $(x', y')$  into  $S$  such that  $x' \leq x$  for all  $(x, y) \in S$ ;  $\text{remove}(x')$  removes from  $S$  all pairs  $(x, y)$  such that  $x \geq x'$ ;  $\text{min}$  returns the pair  $(x, y) \in S$  with the smallest value of  $y$ . A data structure supporting those operations is standard.

► **Lemma 16.** *A set  $S$  of pairs  $(x, y)$  can be maintained in linear space, so that insert, remove, min operations take amortised constant time.*

This data structure is used in our setting as follows: when we begin to process a snake  $u_1, \dots, u_s$ , we insert into  $S_1$  each child  $u$  of  $u_s$ , whenever  $\text{spine}(u) \neq u$  (so  $\text{spine}(u) = u_j$  for some  $j$ ); this can be done in linear time using RadixSort. We also initialize  $S_2$  as empty. After processing  $u_i$  we create a pair  $(j, \text{opt}(u_i))$ , where  $\text{spine}(u_i) = u_j$ , insert it into  $S_2$  and remove from  $S_1, S_2$  elements with first coordinate less or equal to  $i$ . To calculate  $\text{minad}(u_i)$  we take minima from  $S_1, S_2$  and return the minimum of their second coordinates.

Note that  $\text{insert}(x', y')$  assumes  $x' \leq x$  for all  $(x, y) \in S$ . This is guaranteed for  $S_1$  as we sort the pairs by their first coordinate before inserting, for  $S_2$  this requires a simple proof.

#### 4 Application: approximation of the smallest TSLP

In the string case, the algorithm transforming the LZ77 representation into a grammar [25, 8, 26, 19] (with a  $\mathcal{O}(\log(n/\ell))$  size increase, where  $\ell$  is the size of the LZ77 representation and  $n$  the length of the text) has profound implications. On one hand, this is still the best approximation algorithms for the smallest SLP construction, in particular it returns a grammar at most quadratic in size of the smallest one, which is for instance usually good enough for computational complexity considerations. On the other hand, this algorithm is used as a first step in algorithms processing LZ77 compressed text, see for instance [16].

In this section we show a similar result for our factorisations of trees: We give an algorithm that transform a tree factorisation of size  $\ell$  into a TSLP of size  $\mathcal{O}(\ell r + \ell r \log(n/\ell g))$ , where  $r$  is the maximal arity on nodes in the tree and  $n$  the size of the tree. In particular, the approximation ratio is  $\mathcal{O}(r g \log(n/(r g)))$ , which matches a recent best known bound [20]; we call our algorithm **FactToG**. This algorithm generalises the approach of [19] from the string case to tree case.

► **Theorem 17.** *FactToG runs in linear time and returns a TSLP of size  $\mathcal{O}(r g + r g \log(n/r g))$ , where  $n$  is the size of the input tree  $T$ ,  $g$  the size of the smallest TSLP for  $T$  and  $r$  the maximal arity of nodes in  $T$ .*

#### 4.1 Idea

**FactToG** repeatedly applies two operations: *leaf compression* and *unary nodes compression*. Given a tree  $T$  leaf compression deletes all leaves in  $T$  and relabels their parents in a consistent way: the new label  $f'$  of the node  $v$  uniquely determines the previous label of  $v$  (say  $f$ ), on which positions  $v$  had leaf-children (say  $i_1, i_2, \dots, i_\ell$ ) and what were their labels (say  $c_1, \dots, c_\ell$ ). The “reverse” of leaf compression corresponds to a TSLP rule of a form

$$f'(y_1, \dots, y_{i_1-1}, y_{i_1+1}, \dots, y_{i_2-1}, y_{i_2+1}, \dots, y_{i_\ell-1}, y_{i_\ell+1}, \dots) \rightarrow f(y_1, \dots, y_{i_1-1}, c_1, y_{i_1+1}, \dots, y_{i_2-1}, c_2, y_{i_2+1}, \dots, y_{i_\ell-1}, c_\ell, y_{i_\ell+1}, \dots) \quad (1a)$$

Unary nodes compression chooses some neighbouring unary nodes (say labelled with  $a, b$ ) deletes the lower of them and relabels the remaining one (say with  $c$ ); the new label uniquely

determines the original labels. The corresponding TSLP rule is

$$c(y) \rightarrow a(b(y)) . \quad (1b)$$

These operations are iterated until a single-node tree is obtained.

We want to perform several unary nodes compression in parallel. As pairs of unary nodes may overlap, we choose a set of non-overlapping pairs. For appropriate choice the leaf compression followed by (parallel) unary nodes compression reduces the size of the tree by a constant factor. We call this a *phase* of **FactToG**. One phase can be implemented in linear time, assuming that **RadixSort** for grouping nodes; this yields a total linear running time.

As we end **FactToG** when the tree is reduced to a single node, the TSLP rules produced on the way yield an TSLP generating the input tree. It remains to bound its size in terms of the smallest TSLP for the input tree, thus yielding a bound on the approximation ratio. To this end we use the 1-BFS-factorisations: firstly, we show that the size of the smallest 1-BFS-factorisation is  $\mathcal{O}(rg)$ , where  $r$  is the maximal arity of nodes in the input tree and  $g$  the size of the smallest TSLP for it. Then we show that if  $T$  has a 1-factorisation of size  $m$  then rules introduced during one phase of **FactToG** have size  $\mathcal{O}(m)$  and that the resulting tree also has a 1-factorisation of size  $m$ . As there are  $\mathcal{O}(\log n)$  phases, this yields  $\mathcal{O}(rg \log n)$  bound on the size of the TSLP, better estimations yield the actual bound in Theorem 17.

The bound on the size of the produced rules is rather straightforward, but the construction of a 1-factorisation of the resulting tree is involved. To this end ensure that when we perform a compression operation on a fragment, we perform the same operation also on its definition. In particular, each compression operation should be performed wholly within or wholly outside a fragment (or definition of a fragment). As a result, way the factorisation of the resulting tree is “inherited” from the original tree.

To guarantee this property on one hand we choose the pairs of unary nodes to compress with the help of the factorisation, and on the other we modify the factorisation by “popping” the nodes that are compressed with nodes outside the fragments: we remove such a node from the fragment and make it a free node. This sometimes introduces new fragments, but they are of specific form and are under control. We call the (restrictions of) fragments that were present in the input *original fragments* and the created ones the *introduced fragments*.

Due to compression operations we cannot guarantee that we keep a BFS-factorisation. Instead, we store an order “ $<$ ” on the nodes and require that  $def[v] < v$  (as well as some other technical conditions). The actual order is obtained in a natural way: it is the original BFS order restricted to the nodes in the current tree.

## 4.2 Representation and basic operations

**Factorisation.** The factorisation depends on an order  $<$ , we refer to the factorisation as  $(T, \mathcal{F}, <)$  and call it a *proper factorisation* if (1)  $def[v] < v$ ; (2)  $parent[v] < v$  and (3) a hole of a fragment is not a root of the same fragment. The first condition ensures that a proper factorisation is a generalisation of a BFS-factorisation, second ensures that the order on the vertices is meaningful and the third that there are no useless fragments. Initially  $<$  is the BFS order on the input tree and clearly for BFS-order the 1-BFS-factorisation is proper. We often process the tree by considering nodes in the stored order, then we say that we *traverse the tree*.

Each node  $v$  in a fragment (except for the hole), has a definition link  $def[v]$  to its definition. We also store bitflags telling whether a node is a root or a hole of a fragment. A  $(T, \mathcal{F}, <)$  stores the order  $<$ .

**Popping nodes.** During the algorithm we modify the factorisation by *popping* nodes:

- *Popping a unary node up.* When a root  $v$  of a fragment is a unary node, we make  $v$  a free node and its unique child a new root.
- *Popping a unary node down.* When a parent  $v$  of a hole is a unary node, we perform symmetrical operations: we make  $v$  a free node and make it a hole of this fragment.
- *Popping a nullary node.* When a whole 0-fragment consist of a single nullary node  $v$ , we turn it into a free node.
- *Popping a node down.* When  $v$  is a parent of a hole we remove the current hole from the fragment, make  $v$  a new hole, make  $v$  a free node and create new 0-fragments rooted in each child of  $v$ , except the ex-hole. This generalises the popping of a unary node down.

When the variant is clear from the context, we simply refer to *popping a node  $v$* . A simple case inspection shows that popping nodes turns a proper factorisation into a proper one.

► **Lemma 18.** *Popping a node in a proper factorisation results in a proper factorisation.*

### 4.3 Compression operations and the algorithm

When we perform the compression operations, both the fragment and its definition should be modified in the same way. We now explain how to ensure this.

**Leaf compression** Both affected nodes (the deleted leaf and its parent) should either be both within a fragment or both outside it, the same applies to the definition of the fragment. The following conditions ensure this:

(L1) A leaf is neither a hole nor a definition of a hole.

(L2) A leaf is not a whole fragment.

Ensuring (L1–L2) is done by **PairLeaves**: if a leaf is a root of a fragment (so it violates (L2)), we turn it into a free node (so pop it). To ensure (L1) we look at each hole, if it violates (L1) then we pop its parent from the fragment; in this way we may create 0-fragments violating (L2), we pop nodes from such fragments as well.

► **Lemma 19.** *PairLeaves applied to a proper factorisation returns a proper factorisation satisfying (L1–L2). It introduces at most  $r$  free nodes per 1-fragment and 1 per 0-fragment.*

Then we perform the compression **CompLeaves**: traverse the tree, if the node  $v$  has a leaf child, then change its label: If this node is within a fragment, copy the label from the definition; if it is a free node then assign it a new label. If the read node is a leaf then delete it. The new order is a restriction of the order to the nodes that were not deleted.

► **Lemma 20.** *CompLeaves applied to a proper factorisation satisfying (L1–L2) returns a proper factorisation. The size of created rules is twice the number of removed free leaves.*

**Unary nodes compression.** Since we replace pairs of nodes with new ones, this compression boils down to pairing (of unary nodes): for each node we should determine, whether it is a top (*top*) or bottom one (*bottom*) in a pair or perhaps that it is unpaired (*none*). We construct a pairing satisfying the following properties, which are a slight modifications of conditions for a similar algorithm for in the string case [19]:

(P1) There are no two neighbouring unary nodes that are both unpaired.

(P2) If a root of a fragment is unary then it is not paired with its parent, if a node above the hole is unary then it is paired with its parent;

(P3)  $v$  and  $def[v]$  are paired in the same way (so either both are unpaired, both are top-nodes in a pair or both bottom nodes in a pair).

(P1) guarantees that compression of chosen pairs decreases the length of each sequence of unary nodes by a constant factor, (P3) says that the fragment is paired exactly in the same way as its definition and (P2) ensures that pairing for a fragment is done within this fragment, in particular that we can inherit the factorisation after the compression.

The computation of the pairing is technical and it is deferred to Section 4.4, let us first give the procedure `CompUnary` that uses this pairing: We traverse the tree. If the read node  $v$  is *top* and within a fragment then we replace its label with the label of its definition; if it is *top* and a free node then we assign it a fresh label. If the read node  $v$  is a *bottom* node, then let  $u$  be its unique child: we change father of  $u$  to current father of  $v$  and delete  $v$ .

► **Lemma 21.** *CompUnary applied to a proper factorisation satisfying (P1)–(P3) returns a proper factorisation. The size of introduced rules is at most twice the number of free unary nodes removed from the tree.*

`FactToG` first computes an optimal 1-factorisation. Then it transforms this factorisation into a TSLP in phases, until the tree is reduced into a single node. In each phase it first computes the pairing for the unary nodes and replaces those pairs and then modifies the factorisation so that the leaves compression can be applied and applies the leaves compression.

#### 4.4 Pairing unary nodes

**Removing circular references.** Imagine a unary root of a fragment  $v$  has  $\text{def}[v] = \text{parent}[v]$ . Then no pairing satisfying (P1–P3) can be devised, as all nodes in this fragment should be paired in the same way (i.e. all as top or all as bottom nodes). However, all nodes in this fragment are labeled with the same (unary) label, thus we can pop the root and change the definition of this fragment: each node has the definition two nodes up, instead of one.

Such circular reference can be more complex: it could be that  $\text{def}^\ell[v] = \text{parent}[v]$  for some  $\ell > 1$ , which again makes the pairing impossible. Thus, in some sense, there are many sequences of nodes involved in this circular reference. The solution is in the same spirit as above: we isolate (in some fragment) the unary nodes with such a circular reference and create a new 1-fragment from them. Then we change their definition so that it goes two nodes up. The details of the procedure and the actual properties guaranteed after it are left for the full version.

► **Lemma 22.** *UnaryPreproc runs in linear time and pops  $\mathcal{O}(1)$  nodes per fragment.*

**Pairing of unary nodes.** For the pairing, we first pair the free nodes, so that if a fragment has a unary root then the two free nodes above are paired and if the parent of its hole is unary then the two nodes below are paired. Then we traverse the tree: for a node  $v$  we copy the pairing from its definition, with some exceptions: If  $v$  is a root and its definition is paired as *bottom* then we pop  $v$  and set its status as *none*, similarly, when  $v$  is father of a hole and its definition is *top* then we pop it and set its pairing as *none*. In this way we may violate (P1) (as neighbour of  $v$  is also paired as *none*), by case inspection and  $\mathcal{O}(1)$  additional pops we can find appropriate pairing.

► **Lemma 23.** *PairUnary applied to a proper factorisation returns a proper factorisation and a pairing satisfying (P1)–(P3). It introduces  $\mathcal{O}(1)$  new free nodes per fragment.*

#### 4.5 Analysis

**Size of the factorisation.** Using Lemma 5 and known properties of the TSLP we can show that the smallest 1-factorisation for  $T$  has  $\mathcal{O}(g)$  1-fragments and  $\mathcal{O}(rg)$  0-fragments and free nodes, where  $r$  is the maximal arity of nodes in  $T$ .



**Size of the tree.** One phase of FactToG reduces the size of the tree by a constant factor: if there are no unary nodes in the tree then we remove all leaves and so halve the size. If there are only unary nodes, then by (P1) there are no two consecutive unpaired ones, so the size of the tree also drops by  $1/4$ . The general argument is a mix of the two above.

► **Lemma 24.** FactToG applied to a tree  $T$  returns a tree of size at most  $\frac{3|T|}{4}$ .

**Running time.** The construction of the 1-BFS-factorisation takes linear time. All sub-procedures run in linear time, combined with Lemma 24 this yields a total linear running time.

**Size of rules and popped letters.** The size of new rules is covered by the number of removed free nodes, see Lemma 20, 21, so it is enough to count the number of introduced free nodes. From Lemmata 19 and 22 this is  $\mathcal{O}(r)$  per 1-fragment and  $\mathcal{O}(1)$  per 0-fragment per phase.

**Introduced fragments.** We now bound the number of letters popped from introduced fragments. On one hand we bound the number of introduced 0-fragments and on the other we give an amortised analysis for introduced 1-fragments.

When we introduce a 0-fragment, we *associate* it with the 1-fragment of which it used to be part of. It can be shown that introduced 1-fragments have no associated 0-fragments, furthermore, when new 0-fragment are associated with an original 1-fragment  $F$  then  $F$  has no fragments associated with it. Thus at any point there are  $\mathcal{O}(rg)$  introduced 0-fragments.

We also associate introduced 1-fragments with original 1-fragments; this association is involved and described in the full version. We can amortise the number of nodes they pop.

► **Lemma 25.** At any point there are  $\mathcal{O}(rg)$  introduced 0-fragments. Introduced 1-fragments pop in total amortised  $\mathcal{O}(g)$  nodes per phase.

**Size of the constructed TSLP.** Combining the lemmata above we get an upper bound of  $\mathcal{O}(rg + rg \log(n))$  on the size of the constructed TSLP. In a more detailed analysis we separately consider the computation before and after the moment in which the current tree has size  $rg$ . We apply the above analysis to the first part: there are only  $\mathcal{O}(\log(n/(rg)))$  phases and so the size of constructed part of SLP is  $\mathcal{O}(rg \log(n/rg))$ . In the second step we use a simple fact [20] that any reasonable grammar for a tree of size  $rg$  has size  $\mathcal{O}(rg)$ , which yields the total size  $\mathcal{O}(rg + rg \log(n/rg))$  and so also the claimed approximation ratio.

## 5 Open problems/Future work

String LZ77 can be constructed in LOGSPACE and is used for instance in approximation of the construction of the smallest SLP in the streaming model [9]. Can a similar construction be performed also for the LZ77 for trees?

In case of strings, the LZ77 compressed representation can be directly translated into an SLP [8, 25], yielding an approximation algorithm for SLP construction. Can a similar construction be carried out also for the tree variant of LZ77? This would allow to translate several known algorithms for TSLP to the case of tree factorisations.

---

### References

- 1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

- 2 Tatsuya Akutsu. A bisection algorithm for grammar-based compression of ordered trees. *Inf. Process. Lett.*, 110(18–19):815–820, 2010.
- 3 Philip Bille, Inge Li Gørtz, Gad M. Landau, and Oren Weimann. Tree compression with top trees. *Information and Computation*, 243:166–177, 2015. doi:10.1016/j.i.c.2014.12.012.
- 4 Mikołaj Bojańczyk and Igor Walukiewicz. Forest algebras. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 107–132. Amsterdam University Press, 2008.
- 5 Mireille Bousquet-Mélou, Markus Lohrey, Sebastian Maneth, and Eric Nöth. XML compression via directed acyclic graphs. *Theory Comput. Syst.*, 57(4):1322–1371, 2015. doi:10.1007/s00224-014-9544-x.
- 6 Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4–5):456–474, 2008.
- 7 Katrin Casel, Henning Fernau, Serge Gaspers, Benjamin Gras, and Markus L. Schmid. On the complexity of grammar-based compression over fixed alphabets. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP*, volume 55 of *LIPICs*, pages 122:1–122:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.122.
- 8 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.
- 9 Travis Gagie and Paweł Gawrychowski. Grammar-based compression in a streaming model. In Adrian Horia Dediu, Henning Fernau, and Carlos Martín-Vide, editors, *LATA*, volume 6031 of *LNCS*, pages 273–284. Springer, 2010. doi:10.1007/978-3-642-13089-2\_23.
- 10 Moses Ganardi, Danny Hucce, Markus Lohrey, and Eric Noeth. Tree compression using string grammars. In Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors, *LATIN*, volume 9644 of *LNCS*, pages 590–604. Springer, 2016. doi:10.1007/978-3-662-49529-2\_44.
- 11 Adria Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Context matching for compressed terms. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 93–102. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.17.
- 12 Adria Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Unification with singleton tree grammars. In Ralf Treinen, editor, *RTA*, volume 5595 of *LNCS*, pages 365–379. Springer, 2009. doi:10.1007/978-3-642-02348-4\_26.
- 13 Adria Gascón, Guillem Godoy, Manfred Schmidt-Schauß, and Ashish Tiwari. Context unification with one context variable. *J. Symb. Comput.*, 45(2):173–193, 2010. doi:10.1016/j.jsc.2008.10.005.
- 14 Adria Gascón, Manfred Schmidt-Schauß, and Ashish Tiwari. Two-restricted one context unification is in polynomial time. In Stephan Kreutzer, editor, *CSL*, volume 41 of *LIPICs*, pages 405–422. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.CSL.2015.405.
- 15 Adria Gascón, Ashish Tiwari, and Manfred Schmidt-Schauß. One context unification problems solvable in polynomial time. In *LICS*, pages 499–510. IEEE, 2015. doi:10.1109/LICS.2015.53.
- 16 Paweł Gawrychowski. Pattern matching in Lempel-Ziv compressed strings: fast, simple, and deterministic. In Camil Demetrescu and Magnús M. Halldórsson, editors, *ESA*, volume 6942 of *LNCS*, pages 421–432. Springer, 2011. doi:10.1007/978-3-642-23719-5\_36.

- 17 Danny Hucke, Markus Lohrey, and Eric Noeth. Constructing small tree grammars and small circuits for formulas. In Venkatesh Raman and S. P. Suresh, editors, *FSTTCS*, volume 29 of *LIPICs*, pages 457–468. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICs.FSTTCS.2014.457.
- 18 Artur Jež. Context unification is in PSPACE. In Elias Koutsoupias, Javier Esparza, and Pierre Fraigniaud, editors, *ICALP*, volume 8573 of *LNCS*, pages 244–255. Springer, 2014. doi:10.1007/978-3-662-43951-7\_21.
- 19 Artur Jež. A really simple approximation of smallest grammar. *Theoretical Computer Science*, 616:141–150, 2016. doi:10.1016/j.tcs.2015.12.032.
- 20 Artur Jež and Markus Lohrey. Approximation of smallest linear tree grammar. In Ernst W. Mayr and Natacha Portier, editors, *STACS*, volume 25 of *LIPICs*, pages 445–457. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICs.STACS.2014.445.
- 21 Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. On the complexity of bounded second-order unification and stratified context unification. *Logic Journal of the IGPL*, 19(6):763–789, 2011. doi:10.1093/jigpal/jzq010.
- 22 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.
- 23 Markus Lohrey, Sebastian Maneth, and Roy Mennicke. XML tree structure compression using RePair. *Inf. Syst.*, 38(8):1150–1167, 2013. doi:10.1016/j.is.2013.06.006.
- 24 Markus Lohrey, Sebastian Maneth, and Manfred Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *J. Comput. Syst. Sci.*, 78(5):1651–1669, 2012. doi:10.1016/j.jcss.2012.03.003.
- 25 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.
- 26 Hiroshi Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *J. Discrete Algorithms*, 3(2-4):416–430, 2005. doi:10.1016/j.jda.2004.08.016.
- 27 Manfred Schmidt-Schauß. Linear compressed pattern matching for polynomial rewriting (extended abstract). In Rachid Echahed and Detlef Plump, editors, *TERMGRAPH*, volume 110 of *EPTCS*, pages 29–40, 2013. doi:10.4204/EPTCS.110.5.
- 28 Tetsuo Shibuya. Constructing the suffix tree of a tree with a large alphabet. In *Algorithms and Computation*, pages 225–236. Springer, 1999.
- 29 James A. Storer and Thomas G. Szymanski. The macro model for data compression. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *STOC*, pages 30–39. ACM, 1978.



# Finger Search in Grammar-Compressed Strings\*

Philip Bille<sup>1</sup>, Anders Roy Christiansen<sup>2</sup>, Patrick Høge Cording<sup>3</sup>,  
and Inge Li Gørtz<sup>4</sup>

1 Technical University of Denmark, DTU Compute, Lyngby, Denmark

2 Technical University of Denmark, DTU Compute, Lyngby, Denmark

3 Technical University of Denmark, DTU Compute, Lyngby, Denmark

4 Technical University of Denmark, DTU Compute, Lyngby, Denmark

---

## Abstract

Grammar-based compression, where one replaces a long string by a small context-free grammar that generates the string, is a simple and powerful paradigm that captures many popular compression schemes. Given a grammar, the random access problem is to compactly represent the grammar while supporting random access, that is, given a position in the original uncompressed string report the character at that position. In this paper we study the random access problem with the finger search property, that is, the time for a random access query should depend on the distance between a specified index  $f$ , called the *finger*, and the query index  $i$ . We consider both a static variant, where we first place a finger and subsequently access indices near the finger efficiently, and a dynamic variant where also moving the finger such that the time depends on the distance moved is supported.

Let  $n$  be the size the grammar, and let  $N$  be the size of the string. For the static variant we give a linear space representation that supports placing the finger in  $O(\log N)$  time and subsequently accessing in  $O(\log D)$  time, where  $D$  is the distance between the finger and the accessed index. For the dynamic variant we give a linear space representation that supports placing the finger in  $O(\log N)$  time and accessing and moving the finger in  $O(\log D + \log \log N)$  time. Compared to the best linear space solution to random access, we improve a  $O(\log N)$  query bound to  $O(\log D)$  for the static variant and to  $O(\log D + \log \log N)$  for the dynamic variant, while maintaining linear space. As an application of our results we obtain an improved solution to the longest common extension problem in grammar compressed strings. To obtain our results, we introduce several new techniques of independent interest, including a novel van Emde Boas style decomposition of grammars.

**1998 ACM Subject Classification** E.4 Coding and Information Theory, E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Compression, Grammars, Finger search, Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.36

## 1 Introduction

Grammar-based compression, where one replaces a long string by a small context-free grammar that generates the string, is a simple and powerful paradigm that captures many popular compression schemes including the Lempel-Ziv family [49, 48, 46], Sequitur [35], Run-Length Encoding, Re-Pair [32], and many more [40, 20, 29, 30, 47, 4, 2, 3, 26]. All of these are or can be transformed into equivalent grammar-based compression schemes with little expansion [38, 14].

---

\* A full version of the paper is available at <https://arxiv.org/abs/1507.02853>.



Given a grammar  $\mathcal{S}$  representing a string  $S$ , the *random access problem* is to compactly represent  $\mathcal{S}$  while supporting fast access queries, that is, given an index  $i$  in  $S$  to report  $S[i]$ . The random access problem is one of the most basic primitives for computation on grammar compressed strings, and solutions to the problem are a key component in a wide range of algorithms and data structures for grammar compressed strings [9, 10, 21, 22, 23, 8, 28, 42, 43, 6].

In this paper we study the random access problem with the *finger search property*, that is, the time for a random access query should depend on the distance between a specified index  $f$ , called the *finger*, and the query index  $i$ . We consider two variants of the problem. The first variant is *static finger search*, where we can place a finger with a **setfinger** operation and subsequently access positions near the finger efficiently. The finger can only be moved by a new **setfinger** operation, and the time for **setfinger** is independent of the distance to the previous position of the finger. The second variant is *dynamic finger search*, where we also support a **movefinger** operation that updates the finger such that the update time depends on the distance the finger is moved.

Our main result is efficient solutions to both finger search problems. To state the bounds, let  $n$  be the size the grammar  $\mathcal{S}$ , and let  $N$  be the size of the string  $S$ . For the static finger search problem, we give an  $O(n)$  space representation that supports **setfinger** in  $O(\log N)$  time and **access** in  $O(\log D)$  time, where  $D$  is the distance between the finger and the accessed index. For the dynamic finger search problem, we give an  $O(n)$  space representation that supports **setfinger** in  $O(\log N)$  time and **movefinger** and **access** in  $O(\log D + \log \log N)$  time. The best linear space solution for the random access problem uses  $O(\log N)$  time for **access**. Hence, compared to our result we improve the  $O(\log N)$  bound to  $O(\log D)$  for the static version and to  $O(\log D + \log \log N)$  for the dynamic version, while maintaining linear space. These are the first non-trivial bounds for the finger search problems.

As an application of our results we also give a new solution to the *longest common extension problem* on grammar compressed strings [9, 28, 36]. Here, the goal is to compactly represent  $\mathcal{S}$  while supporting fast lce queries, that is, given a pair of indices  $i, j$  to compute the length of the longest common prefix of  $S[i, N]$  and  $S[j, N]$ . We give an  $O(n)$  space representation that answers queries in  $O(\log N + \log^2 \ell)$ , where  $\ell$  is the length of the longest common prefix. The best  $O(n)$  space solution for this problem uses  $O(\log N \log \ell)$  time, and hence our new bound is always at least as good and better whenever  $\ell = o(N^\epsilon)$ .

## 1.1 Related Work

We briefly review the related work on the random access problem and finger search.

**Random Access in Grammar Compressed Strings.** First note that naively we can store  $S$  explicitly using  $O(N)$  space and report any character in constant time. Alternatively, we can compute and store the sizes of the strings derived by each grammar symbol in  $\mathcal{S}$  and use this to simulate a top-down search on the grammars derivation tree in constant time per node. This leads to an  $O(n)$  space representation using  $O(h)$  time, where  $h$  is the height of the grammar [25]. Improved succinct space representation of this solution are also known [15]. Bille et al. [10] gave a solution using  $O(n)$  and  $O(\log N)$  time, thus achieving a query time independent of the height of the grammar. Verbin and Yu [45] gave a near matching lower bound by showing that any solution using  $O(n \log^{O(1)} N)$  space must use  $\Omega(\log^{1-\epsilon} N)$  time. Hence, we cannot hope to obtain significantly faster query times within  $O(n)$  space. Finally, Belazzougui et al. [6] very recently showed that with superlinear space slightly faster query times are possible. Specifically, they gave a solution using  $O(n\tau \log_\tau N/n)$  space and

$O(\log_\tau N)$  time, where  $\tau$  is a trade-off parameter. For  $\tau = \log^\epsilon N$  this is  $O(n \log^\epsilon N)$  space and  $O(\log N / \log \log N)$  time. Practical solutions to this problem have been considered in [5, 34, 24].

The above solutions all generalize to support decompression of an arbitrary substring of length  $D$  in time  $O(t_{\text{access}} + D)$ , where  $t_{\text{access}}$  is the time for access (and even faster for small alphabets [6]). We can extend this to a simple solution to finger search (static and dynamic). The key idea is to implement `setfinger` as a random access and `access` and `movefinger` by decompressing or traversing, respectively, the part of the grammar in-between the two positions. This leads to a solution that uses  $O(t_{\text{access}})$  time for `setfinger` and  $O(D)$  time for `access` and `movefinger`.

Another closely related problem is the *bookmarking problem*, where a set of positions, called *bookmarks*, are given at preprocessing time and the goal is to support fast substring decompression from any bookmark in constant or near-constant time per decompressed character [16, 21]. In other words, bookmarking allows us to decompress a substring of length  $D$  in time  $O(D)$  if the substring crosses a bookmark. Hence, with bookmarking we can improve the  $O(t_{\text{access}} + D)$  time solution for substring decompression to  $O(D)$  whenever we know the positions of the substrings we want to decompress at preprocessing time. A key component in the current solutions to bookmarking is to trade-off the  $\Omega(D)$  time we need to pay to decompress and output the substring. Our goal is to support access without decompressing in  $o(D)$  time and hence this idea does not immediately apply to finger search.

**Finger Search.** Finger search is a classic and well-studied concept in data structures, see e.g., [7, 11, 13, 39, 17, 19, 27, 33, 31, 37, 41] and the survey [12]. In this setting, the goal is to maintain a dynamic dictionary data structure such that searches have the finger search property. Classic textbook examples of efficient finger search dictionaries include splay trees, skip lists, and level linked trees. Given a comparison based dictionary with  $n$  elements, we can support optimal searching in  $O(\log n)$  time and finger searching in  $O(\log d)$  time, where  $d$  is the rank distance between the finger and the query [12]. Note the similarity to our compressed results that reduce an  $O(\log N)$  bound to  $O(\log D)$ .

## 1.2 Our results

We now formally state our results. Let  $S$  be a string of length  $N$  compressed into a grammar  $\mathcal{S}$  of length  $n$ . Our goal is to support the following operations on  $\mathcal{S}$ .

**access**( $i$ ): return the character  $S[i]$

**setfinger**( $f$ ): set the finger at position  $f$  in  $S$ .

**movefinger**( $f$ ): move the finger to position  $f$  in  $S$ .

The static finger problem is to support `access` and `setfinger`, and the dynamic finger search problem is to support all three operations. We obtain the following bounds for the finger search problems.

► **Theorem 1.** *Let  $\mathcal{S}$  be a grammar of size  $n$  representing a string  $S$  of length  $N$ . Let  $f$  be the current position of the finger, and let  $D = |f - i|$  for some  $i$ . Using  $O(n)$  space we can support either:*

- (i) `setfinger`( $f$ ) in  $O(\log N)$  time and `access`( $i$ ) in  $O(\log D)$  time.
- (ii) `setfinger`( $f$ ) in  $O(\log N)$  time, `movefinger`( $i$ ) and `access`( $i$ ) both in  $O(\log D + \log \log N)$  time.



Compared to the previous best linear space solution, we improve the  $O(\log N)$  bound to  $O(\log D)$  for the static variant and to  $O(\log D + \log \log N)$  for the dynamic variant, while maintaining linear space. These are the first non-trivial solutions to the finger search problems. Moreover, the logarithmic bound in terms of  $D$  may be viewed as a natural grammar compressed analogue of the classic uncompressed finger search solutions. We note that Theorem 1 is straightforward to generalize to multiple fingers. Each additional finger can be set in  $O(\log N)$  time, uses  $O(\log N)$  additional space, and given any finger  $f$ , we can support  $\text{access}(i)$  in  $O(\log D_f)$  time, where  $D_f = |f - i|$ .

### 1.3 Technical Overview

To obtain Theorem 1 we introduce several new techniques of independent interest. First, we consider a variant of the random access problem, which we call the *fringe access problem*. Here, the goal is to support fast access close to the beginning or end (the fringe) of a substring derived by a grammar symbol. We present an  $O(n)$  space representation that supports fringe access from any grammar symbol  $v$  in time  $O(\log D_v + \log \log N)$ , where  $D_v$  is the distance from the fringe in the string  $S(v)$  derived by  $v$  to the queried position.

The main component in our solution to this problem is a new recursive decomposition. The decomposition resembles the classic van Emde Boas data structure [44], in the sense that we recursively partition the grammar into a hierarchy of depth  $O(\log \log N)$  consisting of subgrammars generating strings of lengths  $N^{1/2}, N^{1/4}, N^{1/8}, \dots$ . We then show how to implement fringe access via predecessor queries on special paths produced by the decomposition. We cannot afford to explicitly store a predecessor data structure for each special path, however, using a technique due to Bille et al. [10], we can represent all the special paths compactly in a tree and instead implement the predecessor queries as weighted ancestor queries on the tree. This leads to an  $O(n)$  space solution with  $O(\log D_v + (\log \log N)^2)$  query time. Whenever  $D_v \geq 2^{(\log \log N)^2}$  this matches our desired bound of  $O(\log D_v + \log \log N)$ . To handle the case when  $D_v \leq 2^{(\log \log N)^2}$  we use an additional decomposition of the grammar and further reduce the problem to weighted ancestor queries on trees of small weighted height. Finally, we give an efficient solution to weighted ancestor for this specialized case that leads to our final result for fringe access.

Next, we use our fringe access result to obtain our solution to the static finger search problem. The key idea is to decompose the grammar into heavy paths as done by Bille et al. [10], which has the property that any root-to-leaf path in the directed acyclic graph representing the grammar consists of at most  $O(\log N)$  heavy paths. We then use this to compactly represent the finger as a sequence of the heavy paths. To implement  $\text{access}$ , we binary search the heavy paths in the finger to find an exit point on the finger, which we then use to find an appropriate node to apply our solution to fringe access on. Together with a few additional tricks this gives us Theorem 1(i).

Unfortunately, the above approach for the static finger search problem does not extend to the dynamic setting. The key issue is that even a tiny local change in the position of the finger can change  $\Theta(\log N)$  heavy paths in the representation of the finger, hence requiring at least  $\Omega(\log N)$  work to implement  $\text{movefinger}$ . To avoid this we give a new compact representation of the finger based on both heavy path and the special paths obtained from our van Emde Boas decomposition used in our fringe access data structure. We show how to efficiently maintain this representation during local changes of the finger, ultimately leading Theorem 1(ii).

## 1.4 Longest Common Extensions

As application of Theorem 1, we give an improved solution to longest common extension problem in grammar compressed strings. The first solution to this problem is due to Bille et al. [9]. They showed how to extend random access queries to compute Karp-Rabin fingerprints. Combined with an exponential search this leads to a linear space solution to the longest common extension problem using  $O(\log N \log \ell)$  time, where  $\ell$  is the length of the longest common extension. We note that we can plug in any of the above mentioned random access solution. More recently, Nishimoto et al. [36] used a completely different approach to get  $O(\log N + \log \ell \log^* N)$  query time while using superlinear  $O(n \log N \log^* N)$  space. We obtain:

► **Theorem 2.** *Let  $\mathcal{S}$  be a grammar of size  $n$  representing a string  $S$  of length  $N$ . We can solve the longest common extension problem in  $O(\log N + \log^2 \ell)$  time and  $O(n)$  space where  $\ell$  is the length of the longest common extension.*

Note that we need to verify the Karp-Rabin fingerprints during preprocessing in order to obtain a worst-case query time. Using the result from Bille et al. [10] this gives a randomized expected preprocessing time of  $O(N \log N)$ . Theorem 2 improves the  $O(\log N \log \ell)$  solution to  $O(\log N + \log^2 \ell)$ . The new bound is always at least as good and asymptotically better whenever  $\ell = o(N^\epsilon)$  where  $\epsilon$  is a constant. The new result follows by extending Theorem 1 to compute Karp-Rabin fingerprints and use these to perform the exponential search from [9]. Due to lack of space the proof of Theorem 2 is deferred to the full version.

## 2 Preliminaries

**Strings and Trees.** Let  $S = S[1, |S|]$  be a string of length  $|S|$ . Denote by  $S[i]$  the character in  $S$  at index  $i$  and let  $S[i, j]$  be the substring of  $S$  of length  $j - i + 1$  from index  $i \geq 1$  to  $|S| \geq j \geq i$ , both indices included. Given a rooted tree  $T$ , we denote by  $T(v)$  the subtree rooted in a node  $v$  and the left and right child of a node  $v$  by  $left(v)$  and  $right(v)$  if the tree is binary. The *nearest common ancestor*  $nca(v, u)$  of two nodes  $v$  and  $u$  is the deepest node that is an ancestor of both  $v$  and  $u$ . A weighted tree has weights on its edges. A *weighted ancestor* query for node  $v$  and weight  $d$  returns the highest node  $w$  such that the sum of weights on the path from the root to  $w$  is at least  $d$ .

**Grammars and Straight Line Programs.** Grammar-based compression replaces a long string by a small context-free grammar (CFG). We assume without loss of generality that the grammars are in fact *straight-line programs* (SLPs). The lefthand side of a grammar rule in an SLP has exactly one variable, and the righthand side has either exactly two variables or one terminal symbol. In addition, SLPs are unambiguous and acyclic. We view SLPs as a directed acyclic graph (DAG) where each rule correspond to a node with outgoing ordered edges to its variables. Let  $\mathcal{S}$  be an SLP. As with trees, we denote the left and right child of an internal node  $v$  by  $left(v)$  and  $right(v)$ . The unique string  $S(v)$  of length  $N_v$  is produced by a depth-first left-to-right traversal of  $v$  in  $\mathcal{S}$  and consist of the characters on the leafs in the order they are visited. The corresponding parse tree for  $v$  is denoted  $T(v)$ . We will use the following results, that provides efficient random access from any node  $v$  in  $\mathcal{S}$ .

► **Lemma 3** ([10]). *Let  $S$  be a string of length  $N$  compressed into a SLP  $\mathcal{S}$  of size  $n$ . Given a node  $v \in \mathcal{S}$ , we can support random access in  $S(v)$  in  $O(\log N_v)$  time, and at the same time reporting the sequence of heavy paths and their entry- and exit points in the corresponding depth-first traversal of  $S(v)$ . The number of heavy paths visited is  $O(\log N_v)$ .*

### 3 Fringe Access

In this section we consider the *fringe access problem*. Here the goal is to compactly represent the SLP, such that for any node  $v$ , we can efficiently access locations in the string  $S(v)$  close to the start or the end of the substring. The fringe access problem is the key component in our finger search data structures. A straightforward solution to the fringe access problem is to apply a solution to the random access problem. For instance if we apply the random access solution from Bille et al. [10] stated in Lemma 3 we immediately obtain a linear space solution with  $O(\log N_v)$  access time, i.e., the access time is independent of the distance to the start or the end of the string. This is an immediate consequence of the central grammar decomposition technique of [10], and does not extend to solve fringe access efficiently. Our main contribution in this section is a new approach that bypasses this obstacle. We show the following result.

► **Lemma 4.** *Let  $\mathcal{S}$  be an SLP of size  $n$  representing a string of length  $N$ . Using  $O(n)$  space, we can support access to position  $i$  of any node  $v$ , in time  $O(\log(\min(i, N_v - i)) + \log \log N)$ .*

The key idea in this result is a van Emde Boas style decomposition of  $\mathcal{S}$  combined with a predecessor data structure on selected paths in the decomposition. To achieve linear space we reduce the predecessor queries on these paths to a weighted ancestor query. We first give a data structure with query time  $O((\log \log N)^2 + \log(\min(i, N_v - i)))$ . We then show how to reduce the query time to  $O(\log \log N + \log(\min(i, N_v - i)))$  by reducing the query time for small  $i$ . To do so we introduce an additional decomposition and give a new data structure that supports fast weighted ancestor queries on trees of small weighted height.

For simplicity and without loss of generality we assume that the access point  $i$  is closest to the start of  $S(v)$ , i.e., the goal is to obtain  $O(\log(i) + \log \log N)$  time. By symmetry we can obtain the corresponding result for access points close to the end of  $S(v)$ .

#### 3.1 van Emde Boas Decomposition for Grammars

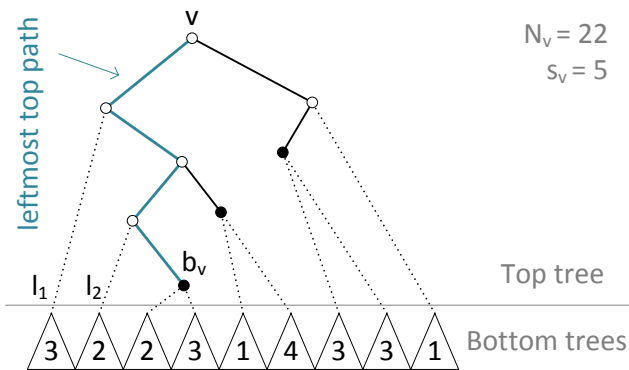
We first define the vEB decomposition on the parse tree  $T$  and then extend it to the SLP  $\mathcal{S}$ . In the decomposition we use the ART decomposition by Alstrup et al. [1].

**ART Decomposition.** The ART decomposition introduced by Alstrup et al. [1] decomposes a tree into a single top tree and a number of bottom trees. Each bottom tree is a subtree rooted in a node of minimal depth such that the subtree contains no more than  $x$  leaves and the top tree is all nodes not in a bottom tree. The decomposition has the following key property.

► **Lemma 5 ([1]).** *The ART decomposition with parameter  $x$  for a rooted tree  $T$  with  $N$  leaves produces a top tree with at most  $\frac{N}{x+1}$  leaves.*

**The van Emde Boas Decomposition.** We define the van Emde Boas Decomposition of a tree  $T$  as follows. The *van Emde Boas (vEB) decomposition* of  $T$  is obtained by recursively applying an ART decomposition: Let  $v = \text{root}(T)$  and  $x = \sqrt{N}$ . If  $N = O(1)$ , stop. Otherwise, construct an ART decomposition of  $T(v)$  with parameter  $x$ . For each bottom tree  $T(u)$  recursively construct a vEB decomposition with  $v = u$  and  $x = \sqrt{x}$ .

Define the *level* of a node  $v$  in  $T$  as  $\text{level}(v) = \lfloor \log \log N - \log \log N_v \rfloor$  (this corresponds to the depth of the recursion when  $v$  is included in its top tree).



■ **Figure 1** Example of the ART-decomposition and a leftmost top path. In the top, the nodes forming the top tree are drawn. In the bottom, triangles representing the bottom trees with a number that is the size of the bottom tree.  $v$ 's leftmost top path is shown as well, and the two trees hanging to the left of this path  $l_1$  and  $l_2$ .

Note that except for the nodes on the lowest level – which are not in any top tree – all nodes belong to exactly one top tree. For any node  $v \in T$  not in the last level, let  $T_{top}(v)$  be the top tree  $v$  belongs to. The *leftmost top path* of  $v$  is the path from  $v$  to the *leftmost leaf* of  $T_{top}(v)$ . See Figure 1.

Intuitively, the vEB decomposition of  $T$  defines a nested hierarchy of subtrees that decrease by at least the square root of the size at each step.

**The van Emde Boas Decomposition of Grammars.** Our definition of the vEB decomposition of trees can be extended to SLPs as follows. Since the vEB decomposition is based only on the length of the string  $N_v$  generated by each node  $v$ , the definition of the vEB decomposition is also well-defined on SLPs. As in the tree, all nodes belong to at most one top DAG. We can therefore reuse the terminology from the definition for trees on SLPs as well.

To compute the vEB decomposition first determine the level of each node and then remove all edges between nodes on different levels. This can be done in  $O(n)$  time.

### 3.2 Data Structure

We first present a data structure that achieves  $O((\log \log N)^2 + \log(i))$  time. In the next section we then show how to improve the running time to the desired  $O(\log \log(N) + \log(i))$  bound.

Our data structure contains the following information for each node  $v \in \mathcal{S}$ . Let  $l_1, l_2, \dots, l_k$  be the nodes hanging to the left of  $v$ 's leftmost top path (excluding nodes hanging from the bottom node).

- The length  $N_v$  of  $S(v)$ .
- The sum of the sizes of nodes hanging to the left of  $v$ 's leftmost top path  $s_v = |l_1| + |l_2| + \dots + |l_k|$ .
- A pointer  $b_v$  to the bottom node on  $v$ 's leftmost top path.
- A predecessor data structure over the sequence  $1, |l_1| + 1, |l_1| + |l_2| + 1, \dots, \sum_{i=1}^{k-1} |l_i| + 1$ .

We will later show how to represent this data structure.

In addition we also build the data structure from Lemma 3 that given any node  $v$  supports random access to  $S(v)$  in  $O(\log N_v)$  time using  $O(n)$  space.

**Query.** To perform an access query we proceed as follows. Suppose that we have reached some node  $v$  and we want to compute  $S(v)[i]$ . We consider the following five cases (when multiple cases apply take the first):

1. If  $N_v = O(1)$ . Decompress  $S(v)$  and return the  $i$ 'th character.
2. If  $i \leq s_v$ . Find the predecessor  $p$  of  $i$  in  $v$ 's predecessor structure and let  $u$  be the corresponding node. Recursively find  $S(u)[i - p]$ .
3. If  $i \leq s_v + N_{left(b_v)}$ . Recursively find  $S(left(b_v))[i - s_v]$ .
4. If  $i \leq s_v + N_{b_v}$ . Recursively find  $S(right(b_v))[i - s_v - N_{left(b_v)}]$ .
5. In all other cases, perform a random access for  $i$  in  $\mathcal{S}(v)$  using Lemma 3.

To see correctness, first note that case (1) and (5) are correct by definition. Case (2) is correct since when  $i \leq s_v$  we know the  $i$ 'th leaf must be in one of the trees hanging to the left of the leftmost top path, and the predecessor query ensures we recurse into the correct one of these bottom trees. In case (3) and (4) we check if the  $i$ 'th leaf is either in the left or right subtree of  $b_v$  and if it is, we recurse into the correct one of these.

**Compact Predecessor Data Structures.** We now describe how to represent the predecessor data structure. Simply storing a predecessor structure in every single node would use  $O(n^2)$  space. We can reduce the space to  $O(n)$  using ideas similar to the construction of the "heavy path suffix forest" in [10].

Let  $L$  denote the *leftmost top path forest*. The nodes of  $L$  are the nodes of  $\mathcal{S}$ . A node  $u$  is the parent of  $v$  in  $L$  iff  $u$  is a child of  $v$  in  $\mathcal{S}$  and  $u$  is on  $v$ 's leftmost top path. Thus, a leftmost top path  $v_1, \dots, v_k$  in  $\mathcal{S}$  is a sequence of ancestors from  $v_1$  in  $L$ . The weight of an edge  $(u, v)$  in  $L$  is 0 if  $u$  is a left child of  $v$  in  $\mathcal{S}$  and otherwise  $N_{left(v)}$ . Several leftmost top paths in  $\mathcal{S}$  can share the same suffix, but the leftmost top path of a node in  $\mathcal{S}$  is uniquely defined and thus  $L$  is a forest. A leftmost path ends in a leaf in the top DAG, and therefore  $L$  consists of  $O(n)$  trees each rooted at a unique leaf of a top dag. A predecessor query on the sequence  $1, |l_1| + 1, |l_1| + |l_2| + 1, \dots, \sum_{i=1}^{k-1} |l_i| + 1$  now corresponds to a weighted ancestor query in  $L$ . We plug in the weighted ancestor data structure from Farach-Colton and Muthukrishnan [18], which supports weighted ancestor queries in a forest in  $O(\log \log n + \log \log U)$  time with  $O(n)$  preprocessing and space, where  $U$  is the maximum weight of a root-to-leaf path and  $n$  the number of leaves. We have  $U = N$  and hence the time for queries becomes  $O(\log \log N)$ .

**Space and Preprocessing Time.** For each node in  $\mathcal{S}$  we store a constant number of values, which takes  $O(n)$  space. Both the predecessor data structure and the data structure for supporting random access from Lemma 3 take  $O(n)$  space, so the overall space usage is  $O(n)$ . The vEB decomposition can be computed in  $O(n \log \log N)$  time. The leftmost top paths and the information saved in each node can be computed in linear time. The predecessor data structure uses linear preprocessing time, and thus the total preprocessing time is  $O(n \log \log N)$ .

**Query Time.** Consider each case of the recursion. The time for case (1), (3) and (4) is trivially  $O(1)$ . Case (2) is  $O(\log \log N)$  since we perform exactly one predecessor query in the predecessor data structure.

In case (5) we make a random access query in a node of size  $N_v$ . From Lemma 3 we have that the query time is  $O(\log N_v)$ . We know  $\text{level}(v) = \text{level}(b_v)$  since they are on the same leftmost top path. From the definition of the level it follows for any pair of nodes  $u$  and  $w$  with the same level that  $N_u \geq \sqrt{N_w}$  and thus  $N_{b_v} \geq \sqrt{N_v}$ . From the conditions we have

$i > s_v + N_{b_v} \geq N_{b_v} \geq \sqrt{N_v}$ . Since  $\sqrt{N_v} < i \Leftrightarrow \log N_v < 2 \log i$  we have  $\log N_v = O(\log i)$  and thus the running time for case (5) is  $O(\log N_v) = O(\log i)$ .

Case (1) and (5) terminate the algorithm and can thus not happen more than once. Case (2), (3) and (4) are repeated at most  $O(\log \log N)$  times since the level of the node we recurse on increments by at least one in each recursive call, and the level of a node is at most  $O(\log \log N)$ . The overall running time is therefore  $O((\log \log N)^2 + \log i)$ .

In summary, we have the following result.

► **Lemma 6.** *Let  $\mathcal{S}$  be an SLP of size  $n$  representing a string of length  $N$ . Using  $O(n)$  space, we can support access to position  $i$  of any node  $v$ , in time  $O(\log i + (\log \log N)^2)$ .*

### 3.3 Improving the Query Time for Small Indices

The above algorithm obtains the running time  $O(\log i)$  for  $i \geq 2^{(\log \log N)^2}$ . We will now improve the running time to  $O(\log \log N + \log i)$  by improving the running time in the case when  $i < 2^{(\log \log N)^2}$ .

In addition to the data structure from above, we add another copy of the data structure with a few changes. When answering a query, we first check if  $i \geq 2^{(\log \log N)^2}$ . If  $i \geq 2^{(\log \log N)^2}$  we use the original data structure, otherwise we use the new copy.

The new copy of the data structure is implemented as follows. In the first level of the ART-decomposition let  $x = 2^{(\log \log N)^2}$  instead of  $\sqrt{N}$ . For the rest of the levels use  $\sqrt{x}$  as before. Furthermore, we split the resulting new leftmost top path forest  $L$  into two disjoint parts:  $L_1$  consisting of all nodes with level 1 and  $L_{\geq 2}$  consisting of all nodes with level at least 2. For  $L_1$  we use the weighted ancestor data structure by Farach-Colton and Muthukrishnan [18] as in the previous section using  $O(\log \log n + \log \log N) = O(\log \log N)$  time. However, if we apply this solution for  $L_{\geq 2}$  we end up with a query time of  $O(\log \log n + \log \log x)$ , which does not lead to an improved solution. Instead, we present a new data structure that supports queries in  $O(\log \log x)$  time.

► **Lemma 7** (see full version). *Given a tree  $T$  with  $n$  leaves where the sum of edge weights on any root-to-leaf path is at most  $x$  and the height is at most  $x$ , we can support weighted ancestor queries in  $O(\log \log x)$  time using  $O(n)$  space and preprocessing time.*

We reduce the query time for queries with  $i < 2^{(\log \log N)^2}$  using the new data structure. The level of any node in the new structure is at most  $O(1 + \log \log 2^{(\log \log N)^2}) = O(\log \log \log N)$ . A weighted ancestor query in  $L_1$  takes time  $O(\log \log N)$ . For weighted ancestor queries in  $L_{\geq 2}$ , we know any node  $v$  has height at most  $2^{(\log \log N)^2}$  and on any root-to-leaf path the sum of the weights is at most  $2^{(\log \log N)^2}$ . Hence, by Lemma 7 we support queries in  $O(\log \log 2^{(\log \log N)^2}) = O(\log \log \log N)$  time for nodes in  $L_{\geq 2}$ .

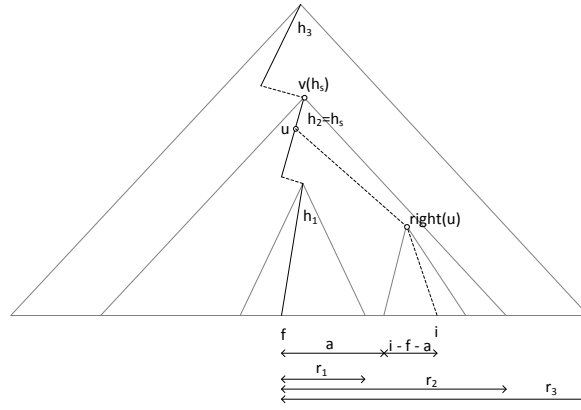
We make at most one weighted ancestor query in  $L_1$ , the remaining ones are made in  $L_{\geq 2}$ , and thus the overall running time is  $O(\log \log N + (\log \log \log N)^2 + \log i) = O(\log \log N + \log i)$ .

In summary, this completes the proof of Lemma 4.

## 4 Static Finger Search

We now show how to apply our solution to the fringe access to obtain a simple data structure for the static finger search problem. This solution will be the starting point for solving the dynamic case in the next section, and we will use it as a key component in our result for longest common extension problem.

Similar to the fringe search problem we assume without loss of generality that the access point  $i$  is to the right of the finger.



■ **Figure 2** Illustration of the data structure for a finger pointing at  $f$  and an access query at location  $i$ .  $h_1, h_2, h_3$  are the heavy paths visited when finding the finger.  $u$  corresponds to  $NCA(v_f, v_i)$  in the parse tree and  $h_s$  is the heavy path on which  $u$  lies, which we use to find  $u$ .  $a$  is a value calculated during the access query.

**Data Structure.** We store the random access data structure from [10] used in Lemma 3 and the fringe search data structures from above. Also from [10] we store the data structure that for any heavy path  $h$  starting in a node  $v$  and an index  $i$  of a leaf in  $T(v)$  gives the exit-node from  $h$  when searching for  $i$  in  $O(\log \log N)$  time and uses  $O(n)$  space.

To represent a finger the key idea is store a compact data structure for the corresponding root-to-leaf path in the grammar that allows us to navigate it efficiently. Specifically, let  $f$  be the position of the current finger and let  $p = v_1 \dots v_k$  denote the path in  $\mathcal{S}$  from the root to  $v_f$  ( $v_1 = \text{root}$  and  $v_k = v_f$ ). Decompose  $p$  into the  $O(\log N)$  heavy paths it intersects, and call these  $h_j = v_1 \dots v_{i_1}, h_{j-1} = v_{i_1+1} \dots v_{i_2}, \dots, h_1 = v_{i_{j-1}+1} \dots v_k$ . Let  $v(h_i)$  be the topmost node on  $h_i$  ( $v(h_j) = v_1, v(h_{j-1}) = v_{i_1}, \dots$ ). Let  $l_j$  be the index of  $f$  in  $\mathcal{S}(v(h_j))$  and  $r_j = N_{v(h_j)} - l_j$ . For the finger we store:

1. The sequence  $r_1, r_2, \dots, r_j$  (note  $r_1 \leq r_2 \leq \dots \leq r_j$ ).
2. The sequence  $v(h_1), v(h_2), \dots, v(h_j)$ .
3. The string  $F_T = S[f + 1, f + \log N]$ .

**Analysis.** The random access and fringe search data structures both require  $O(n)$  space. Each of the 3 bullets above require  $O(\log N)$  space and thus the finger takes up  $O(\log N)$  space. The total space usage is  $O(n)$ .

**Setfinger.** We implement  $\text{setfinger}(f)$  as follows. First, we apply Lemma 3 to make random access to position  $f$ . This gives us the sequence of visited heavy paths which exactly corresponds to  $h_j, h_{j-1}, \dots, h_1$  including the corresponding  $l_i$  values from which we can calculate the  $r_i$  values. So we update the  $r_i$  sequence accordingly. Finally, decompress and save the string  $F_T = S[f + 1, f + \log N]$ .

The random access to position  $f$  takes  $O(\log N)$  time. In addition to this we perform a constant number of operations for each heavy path  $h_i$ , which in total takes  $O(\log N)$  time. Decompressing a string of  $\log N$  characters can be done in  $O(\log N)$  time (using [10]). In total, we use  $O(\log N)$  time.

**Access.** To perform  $\text{access}(i)$  ( $i > f$ ), there are two cases. If  $D = i - f \leq \log N$  we simply return the stored character  $F_T[D]$  in constant time. Otherwise, we compute the node



$u = \text{nca}(v_f, v_i)$  in the parse tree  $T$  as follows. First find the index  $s$  of the successor to  $D$  in the  $r_i$  sequence using binary search. Now we know that  $u$  is on the heavy path  $h_s$ . Find the exit-nodes from  $h_s$  when searching for respectively  $i$  and  $f$  using the data structure from [10] - the topmost of these two is  $u$ . See Fig. 2. Finally, we compute  $a$  as the index of  $f$  in  $T(\text{left}(u))$  from the right and use the data structure for fringe search from Lemma 4 to compute  $S(\text{right}(u))[i - f - a]$ .

For  $D \leq \log N$ , the operation takes constant time. For  $D > \log N$ , the binary search over a sequence of  $O(\log N)$  elements takes  $O(\log \log N)$  time, finding the exit-nodes takes  $O(\log \log N)$  time, and the fringe search takes  $O(\log(i - f - a)) = O(\log D)$  time. Hence, in total  $O(\log \log N + \log D) = O(\log D)$  time.

## 5 Dynamic Finger Search

In this section we show how to extend the solution from Section 4 to handle dynamic finger search. The target is to support the `movefinger` operation that will move the current finger, where the time it takes is dependent on how far the finger is moved. Obviously, it should be faster than simply using the `setfinger` operation. The key difference from the static finger is a new decomposition of a root-to-leaf path into paths. The new decomposition is based on a combination of heavy paths and leftmost top paths, which we will show first. Then we show how to change the data structure to use this decomposition, and how to modify the operations accordingly. Finally we shortly describe how to generalize the solution to work when `movefinger/access` might both be to the left and right of the current finger.

Before we start, let us see why the data structure for the static finger cannot directly be used for dynamic finger. Suppose we have a finger pointing at  $f$  described by  $\Theta(\log N)$  heavy paths. It might be the case that after a `movefinger( $f + 1$ )` operation, it is  $\Theta(\log N)$  completely different heavy paths that describes the finger. In this case we must do  $\Theta(\log N)$  work to keep our finger data structure updated. This can for instance happen when the current finger is pointing at the right-most leaf in the left subtree of the root.

Furthermore, in the solution to the static problem, we store the substring  $S[f+1, f+\log N]$  decompressed in our data structure. If we perform a `movefinger( $f + \log N$ )` operation nothing of this substring can be reused. To decompress  $\log N$  characters takes  $\Omega(\log N)$  time, thus we cannot do this in the `movefinger` operation and still get something faster than  $\Theta(\log N)$ .

### 5.1 Left Heavy Path Decomposition of a Path

Let  $p = v_1 \dots v_k$  be a root-to-leaf path in  $\mathcal{S}$ . A subpath  $p_i = v_a \dots v_b$  of  $p$  is a *maximal heavy subpath* if  $v_a \dots v_b$  is part of a heavy path and  $v_{b+1}$  is not on the same heavy path. Similarly, a subpath  $p_i = v_a \dots v_b$  of  $p$  is a *maximal leftmost top subpath* if  $v_a \dots v_b$  is part of a leftmost top path and  $\text{level}(v_b) \neq \text{level}(v_{b+1})$ .

A *left heavy path decomposition* is a decomposition of a root-to-leaf path  $p$  into an arbitrary sequence  $p_1 \dots p_j$  of maximal heavy subpaths, maximal leftmost top subpaths and (non-maximal) leftmost top subpaths immediately followed by maximal heavy subpaths.

Define  $v(p_i)$  as the topmost node on the subpath  $p_i$ . Let  $l_j$  be the index of the finger  $f$  in  $\mathcal{S}(v(p_j))$  and  $r_j = N_{v(p_j)} - l_j$ . Let  $t(p_i)$  be the type of  $p_i$ ; either heavy subpath (*HP*) or leftmost top subpath (*LTP*).

A left heavy path decomposition of a root-to-leaf path  $p$  is not unique. The heavy path decomposition of  $p$  is always a valid left heavy path decomposition as well. The visited heavy paths and leftmost top paths during fringe search are always maximal and thus is always a valid left heavy path decomposition.

► **Lemma 8.** *The number of paths in a left heavy path decomposition is  $O(\log N)$ .*

**Proof.** There are at most  $O(\log N)$  heavy paths that intersects with a root-to-leaf path (Lemma 3). Each of these can at most be used once because of the maximality. So there can at most be  $O(\log N)$  maximal heavy paths. Each time there is a maximal leftmost top path, the level of the following node on  $p$  increases. This can happen at most  $O(\log \log N)$  times. Each non-maximal leftmost top path is followed by a maximal heavy path, and since there are only  $O(\log N)$  of these, this can happen at most  $O(\log N)$  times. Therefore the sequence of paths has length  $O(\log N + \log \log N + \log N) = O(\log N)$ . ◀

## 5.2 Data Structure

We use the data structures from [10] as in the static variant and the fringe access data structure with an extension. In the fringe access data structure there is a predecessor data structure for all the nodes hanging to the left of a leftmost top path. To support `access` and `movefinger` we need to find a node hanging to the left or right of a leftmost top path. We can do this by storing an identical predecessor structure for the accumulated sizes of the nodes hanging to the right of each leftmost top path. Again, the space usage for this predecessor structure can be reduced to  $O(n)$  by turning it into a weighted ancestor problem.

To represent a finger the idea is again to have a compact data structure representing the root-to-leaf path corresponding to the finger. This time we will base it on a left heavy path decomposition instead of a heavy path decomposition. Let  $f$  be the current position of the finger. For the root-to-leaf path to  $v_f$  we maintain a left heavy path decomposition, and store the following for a finger:

1. The sequence  $r_1, r_2, \dots, r_j$  ( $r_1 \leq r_2 \leq \dots \leq r_j$ ) on a stack with the last element on top.
2. The sequence  $v(p_1), v(p_2), \dots, v(p_j)$  on a stack with the last element on top.
3. The sequence  $t(p_1), t(p_2), \dots, t(p_j)$  on a stack with the last element on top.

**Analysis.** The fringe access data structure takes up  $O(n)$  space. For each path in the left heavy path decomposition we use constant space. Using Lemma 8 we have the space usage of this is  $O(\log N) = O(n)$ .

**Setfinger.** Use fringe access (Lemma 4) to access position  $f$ . This gives us a sequence of leftmost top paths and heavy paths visited during the fringe access which is a valid left heavy path decomposition. Calculate  $r_i$  for each of these and store the three sequences of  $r_i$ ,  $v(p_i)$  and  $t(p_i)$  on stacks.

The fringe access takes  $O(\log f + \log \log N)$  time. The number of subpaths visited during the fringe access cannot be more than  $O(\log f + \log \log N)$  and we only perform constant extra work for each of these.

**Access.** To implement `access(i)` for  $i > f$  we have to find  $u = \text{nca}(v_i, v_f)$  in  $T$ . Find the index  $s$  of the successor to  $D = i - f$  in  $r_1, r_2, \dots, r_j$  using binary search. We know  $\text{nca}(v_i, v_f)$  lies on  $p_s$ , and  $v_i$  is in a subtree that hangs of  $p_s$ . The exit-nodes from  $p_s$  to  $v_f$  and  $v_i$  are now found - the topmost of these two is  $\text{nca}(v_i, v_f)$ . If  $t(p_s) = HP$  then we can use the same data structure as in the static case, otherwise we perform the predecessor query on the extra predecessor data structure for the nodes hanging of the leftmost top path. Finally, we compute  $a$  as the index of  $f$  in  $S(\text{left}(u))$  from the right and use the data structure for fringe access from Lemma 4 to compute  $S(\text{right}(u))[i - f - a]$ .

The binary search on  $r_1, r_2, \dots, r_j$  takes  $O(\log \log N)$  time. Finding the exit-nodes from  $p_s$  takes  $O(\log \log N)$  in either case. Finally the fringe access takes  $O(\log(i-f-a) + \log \log N) = O(\log D + \log \log N)$ . Overall it takes  $O(\log D + \log \log N)$ . Note the extra  $O(\log \log N)$  time usage because we have not decompressed the first  $\log N$  characters following the finger.

**Movefinger.** To move the finger we combine the `access` and `setfinger` operations. Find the index  $s$  of the successor to  $D = i - f$  in  $r_1, r_2, \dots, r_j$  using binary search. Now we know  $u = \text{nca}(v_i, v_f)$  must lie on  $p_s$ . Find  $u$  in the same way as when performing `access`. From all of the stacks pop all elements above index  $s$ . Compute  $a$  as the index of  $f$  in  $S(\text{left}(u))$  from the right. The finger should be moved to index  $i - f - a$  in  $\text{right}(u)$ . First look at the heavy path  $\text{right}(u)$  lies on and find the proper exit-node  $w$  using the data structure from [10]. Then continue with fringe search from the proper child of  $w$ . This gives a heavy path followed by a sequence of maximal leftmost top paths and heavy paths needed to reach  $v_i$  from  $\text{right}(u)$ , push the  $r_j, v(p_j)$ , and  $t(p_j)$  values for these on top of the respective stacks.

We now verify the sequence of paths we maintain is still a valid left heavy path decomposition. Since fringe search gives a sequence of paths that is a valid left heavy path decomposition, the only problem might be  $p_s$  is no longer maximal. If  $p_s$  is a heavy path it will still be maximal, but if  $p_s$  is a leftmost top path then  $\text{level}(u)$  and  $\text{level}(\text{right}(u))$  might be equal. But this possibly non-maximal leftmost top path is always followed by a heavy path. Thus the overall sequence of paths remains a left heavy path decomposition.

The successor query in  $r_1, r_2, \dots, r_j$  takes  $O(\log \log N)$  time. Finding  $u$  on  $p_i$  takes  $O(\log \log N)$  time, and so does finding the exit-node on the following heavy path. Popping a number of elements from the top of the stacks can be done in  $O(1)$  time. Finally the fringe access takes  $O(\log(i-f-a) + \log \log N) = O(\log D + \log \log n)$  including pushing the right elements on the stacks. Overall the running time is therefore  $O(\log D + \log \log n)$ .

## 6 Moving/Access to the Left of the Dynamic Finger

Previously we have assumed  $i > f$ , we will now show how this assumption can be removed. It is easy to see we can mirror all data structures and we will have a solution that works for  $i < f$  instead. Unfortunately, we cannot just use a copy of each independently, since one of them only supports moving the finger to the left and the other only supports moving to the right. We would like to support moving the finger left and right arbitrarily. This was not a problem with the static finger since we could just make `setfinger` in both the mirrored and non-mirrored data structures in  $O(\log N)$  time.

Instead we extend our finger data structure. First we extend the left heavy path decomposition to a *left right heavy path decomposition* by adding another type of paths to it, namely *rightmost top paths* (the mirrored version of leftmost top paths). Thus a *left right heavy path decomposition* is a decomposition of a root-to-leaf path  $p$  into an arbitrary sequence  $p_1 \dots p_j$  of maximal heavy subpaths, maximal leftmost/rightmost top subpaths and (non-maximal) leftmost/rightmost top subpaths immediately followed by maximal heavy subpaths. Now  $t(p_i) = \text{HP|LTP|RTP}$ . Furthermore, we save the sequence  $l_1, l_2, \dots, l_j$  ( $l_j$  being the left index of  $f$  in  $T(v(p_i))$ ) on a stack like the  $r_1, r_2, \dots, r_j$  values, etc.

When we do `access` and `movefinger` where  $i < f$ , the subpath  $p_s$  where  $\text{nca}(v_f, v_i)$  lies can be found by binary search on the  $l_j$  values instead of the  $r_j$  values. Note the  $l_j$  values are sorted on the stack, just like the  $r_j$  values. The following heavy path lookup/fringe access should now be performed on  $\text{left}(u)$  instead of  $\text{right}(u)$ . The remaining operations can just be performed in the same way as before.

## References

- 1 Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Proc. 39th FOCS*, pages 534–543, 1998.
- 2 A. Apostolico and S. Lonardi. Some theory and practice of greedy off-line textual substitution. In *Proc. DCC*, pages 119–128, 1998.
- 3 A. Apostolico and S. Lonardi. Compression of biological sequences by greedy off-line textual substitution. In *Proc. DCC*, pages 143–152, 2000.
- 4 Alberto Apostolico and Stefano Lonardi. Off-line compression by greedy textual substitution. *Proceedings of the IEEE*, 88(11):1733–1744, 2000.
- 5 D. Belazzougui, T. Gagie, P. Gawrychowski, J. Karkkainen, A. Ordonez, S. J. Puglisi, and Y. Tabei. Queries on lz-bounded encodings. In *Proc. DCC*, pages 83–92, April 2015. doi:10.1109/DCC.2015.69.
- 6 Djamal Belazzougui, Patrick Hagge Cording, Simon J. Puglisi, and Yasuo Tabei. Access, rank, and select in grammar-compressed strings. In *Proc. 23rd ESA*, 2015.
- 7 Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Inform. Process. Lett.*, 5(3):82–87, 1976.
- 8 Philip Bille, Patrick Hagge Cording, and Inge Li Gørtz. Compressed subsequence matching and packed tree coloring. *Algorithmica*, pages 1–13, 2015. doi:10.1007/s00453-015-0068-9.
- 9 Philip Bille, Patrick Hagge Cording, Inge Li Gørtz, Benjamin Sach, Hjalte Wedel Vildhøj, and Søren Vind. Fingerprints in compressed strings. In *Proc. 13th SWAT*, 2013.
- 10 Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM J. Comput.*, 44(3):513–539, 2014. Announced at SODA 2011.
- 11 Guy E. Blelloch, Bruce M. Maggs, and Shan Leung Maverick Woo. Space-efficient finger search on degree-balanced search trees. In *Proc. 14th SODA*, pages 374–383, 2003.
- 12 Gerth Stølting Brodal. Finger search trees. In *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004.
- 13 Gerth Stølting Brodal, George Lagogiannis, Christos Makris, Athanasios K. Tsakalidis, and Kostas Tsichlas. Optimal finger search trees in the pointer machine. *J. Comput. Syst. Sci.*, 67(2):381–418, 2003. doi:10.1016/S0022-0000(03)00013-8.
- 14 M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shefat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005. Announced at STOC 2002 and SODA 2002.
- 15 Francisco Claude and Gonzalo Navarro. Self-indexed grammar-based compression. *Fund. Inform.*, 111(3):313–337, 2011.
- 16 Patrick Hagge Cording, Paweł Gawrychowski, and Oren Weimann. Bookmarks in grammar-compressed strings. In *Proc. 23rd SPIRE*, pages x–y, 2016.
- 17 Paul F. Dietz and Rajeev Raman. A constant update time finger search tree. *Inf. Process. Lett.*, 52(3):147–154, 1994.
- 18 Martin Farach and S. Muthukrishnan. Perfect hashing for strings: Formalization and algorithms. In *Proc. 7th CPM*, pages 130–140. Springer, 1996.
- 19 Rudolf Fleischer. A simple balanced search tree with  $O(1)$  worst-case update time. *Int. J. Found. Comput. Sci.*, 7(2):137–150, 1996. doi:10.1142/S0129054196000117.
- 20 P. Gage. A new algorithm for data compression. *The C Users J.*, 12(2):23–38, 1994.
- 21 Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. A faster grammar-based self-index. In *Proc. 6th LATA*, pages 240–251, 2012.
- 22 Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. LZ77-based self-indexing with faster pattern matching. In *Proc. 11th LATIN*, pages 731–742. Springer, 2014.

- 23 Travis Gagie, Pawel Gawrychowski, and Simon J. Puglisi. Approximate pattern matching in lz77-compressed texts. *J. Discrete Algorithms*, 32:64–68, 2015. doi:10.1016/j.jda.2014.10.003.
- 24 Travis Gagie, Christopher Hoobin, and Simon J. Puglisi. Block graphs in practice. In *Proc. ICABD*, pages 30–36, 2014.
- 25 Leszek Gąsieniec, Roman Kolpakov, Igor Potapov, and Paul Sant. Real-time traversal in grammar-based compressed files. In *Proc. 15th DCC*, page 458, 2005.
- 26 Keisuke Goto, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. LZD factorization: Simple and practical online grammar compression with variable-to-fixed encoding. In *Proc. 26th CPM*, pages 219–230. Springer, 2015.
- 27 Leonidas J. Guibas, Edward M. McCreight, Michael F. Plass, and Janet R. Roberts. A new representation for linear lists. In *Proc. 9th STOC*, pages 49–60, 1977.
- 28 Tomohiro I, Wataru Matsubara, Kouji Shimohira, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda, Kazuyuki Narisawa, and Ayumi Shinohara. Detecting regularities on grammar-compressed strings. *Inform. Comput.*, 240:74–89, 2015.
- 29 J. C. Kieffer and E. H. Yang. Grammar based codes: A new class of universal lossless source codes. *IEEE Trans. Inf. Theory*, 46(3):737–754, 2000.
- 30 J. C. Kieffer, E. H. Yang, G. J. Nelson, and P. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Trans. Inf. Theory*, 46(5):1227–1245, 2000.
- 31 S. Rao Kosaraju. Localized search in sorted lists. In *Proc. 13th STOC*, pages 62–69, New York, NY, USA, 1981. doi:10.1145/800076.802458.
- 32 N. Jesper Larsson and Alistair Moffat. Off-line dictionary-based compression. *Proc. IEEE*, 88(11):1722–1732, 2000.
- 33 Kurt Mehlhorn. A new data structure for representing sorted lists. In *Proc. WG*, pages 90–112, 1981.
- 34 Gonzalo Navarro and Alberto Ordóñez. Grammar compressed sequences with rank/select support. In *21st SPIRE*, pages 31–44. Springer, 2014.
- 35 Craig G. Nevill-Manning and Ian H. Witten. Identifying Hierarchical Structure in Sequences: A linear-time algorithm. *J. Artificial Intelligence Res.*, 7:67–82, 1997.
- 36 Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Fully dynamic data structure for LCE queries in compressed space. In *Proc. 41st MFCS*, pages 72:1–72:15, 2016. doi:10.4230/LIPIcs.MFCS.2016.72.
- 37 William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.
- 38 W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003.
- 39 Raimund Seidel and Cecilia R. Aragon. Randomized search trees. *Algorithmica*, 16(4/5):464–497, 1996.
- 40 Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Byte Pair encoding: A text compression scheme that accelerates pattern matching. *Technical Report DOI-TR-161, Dept. of Informatics, Kyushu University*, 1999.
- 41 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, July 1985.
- 42 Toshiya Tanaka, I Tomohiro, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing convolution on grammar-compressed text. In *Proc. 23rd DCC*, pages 451–460, 2013.
- 43 I Tomohiro, Takaaki Nishimoto, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Compressed automata for dictionary matching. *Theor. Comput. Sci.*, 578:30–41, 2015.
- 44 P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Theory Comput. Syst.*, 10(1):99–127, 1976.

- 45 Elad Verbin and Wei Yu. Data structure lower bounds on random access to grammar-compressed strings. In *Proc. 24th CPM*, pages 247–258, 2013.
- 46 Terry A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, 1984.
- 47 E. H. Yang and J. C. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform – part one: Without context models. *IEEE Trans. Inf. Theory*, 46(3):755–754, 2000.
- 48 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977.
- 49 Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, 1978.

# Characterization and Lower Bounds for Branching Program Size Using Projective Dimension

Krishnamoorthy Dinesh<sup>1</sup>, Sajin Koroth<sup>2</sup>, and Jayalal Sarma<sup>3</sup>

**1** Indian Institute of Technology Madras, Chennai, India

**2** Indian Institute of Technology Madras, Chennai, India

**3** Indian Institute of Technology Madras, Chennai, India

---

## Abstract

---

We study projective dimension, a graph parameter (denoted by  $\text{pd}(G)$  for a graph  $G$ ), introduced by Pudlák and Rödl (1992). For a Boolean function  $f$  (on  $n$  bits), Pudlák and Rödl associated a bipartite graph  $G_f$  and showed that size of the optimal branching program computing  $f$  (denoted by  $\text{bysize}(f)$ ) is at least  $\text{pd}(G_f)$  (also denoted by  $\text{pd}(f)$ ). Hence, proving lower bounds for  $\text{pd}(f)$  imply lower bounds for  $\text{bysize}(f)$ . Despite several attempts (Pudlák and Rödl (1992), Rónyai et.al, (2000)), proving super-linear lower bounds for projective dimension of explicit families of graphs has remained elusive.

We observe that there exist a Boolean function  $f$  for which the gap between the  $\text{pd}(f)$  and  $\text{bysize}(f)$  is  $2^{\Omega(n)}$ . Motivated by the argument in Pudlák and Rödl (1992), we define two variants of projective dimension – *projective dimension with intersection dimension 1* (denoted by  $\text{upd}(f)$ ) and *bitwise decomposable projective dimension* (denoted by  $\text{bpdim}(f)$ ). We show the following results:

- (a) We observe that there exist a Boolean function  $f$  for which the gap between  $\text{upd}(f)$  and  $\text{bysize}(f)$  is  $2^{\Omega(n)}$ . In contrast, we also show that the bitwise decomposable projective dimension characterizes size of the branching program up to a polynomial factor. That is, there exists a large constant  $c > 0$  and for any function  $f$ ,

$$\text{bpdim}(f)/6 \leq \text{bysize}(f) \leq (\text{bpdim}(f))^c.$$

- (b) We introduce a new candidate function family  $f$  for showing super-polynomial lower bounds for  $\text{bpdim}(f)$ . As our main result, we demonstrate gaps between  $\text{pd}(f)$  and the above two new measures for  $f$ :

$$\text{pd}(f) = O(\sqrt{n}) \quad \text{upd}(f) = \Omega(n) \quad \text{bpdim}(f) = \Omega\left(\frac{n^{1.5}}{\log n}\right).$$

- (c) Although not related to branching program lower bounds, we derive exponential lower bounds for two restricted variants of  $\text{pd}(f)$  and  $\text{upd}(f)$  respectively by observing that they are exactly equal to well-studied graph parameters – bipartite clique cover number and bipartite partition number respectively.

**1998 ACM Subject Classification** F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes

**Keywords and phrases** Projective Dimension, Lower Bounds, Branching Program Size.

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.37

## 1 Introduction

A central question in complexity theory – the P vs L problem – asks if a deterministic Turing machine that runs in polynomial time can accept any language that cannot be accepted



© Krishnamoorthy Dinesh, Sajin Koroth, and Jayalal Sarma;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 37; pp. 37:1–37:14



Leibniz International Proceedings in Informatics

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



by deterministic Turing machines with logarithmic space bound. A stronger version of the problem asks if  $P$  is separate from  $L/\text{poly}$  (deterministic logarithmic space given polynomial sized advice). The latter, recast in the language of circuit complexity theory, asks if there exists an *explicit* family of functions ( $f : \{0, 1\}^n \rightarrow \{0, 1\}$ ) computable in polynomial time (in terms of  $n$ ), such that any family of deterministic branching programs computing them has to be of size  $2^{\Omega(n)}$ . However, the best known non-trivial size lower bound against deterministic branching programs, due to Nechiporuk [11] in 1970s, is  $\Omega(\frac{n^2}{\log^2 n})$ .

Pudlák and Rödl [12] described a linear algebraic approach to show size lower bounds against deterministic branching programs. They introduced a linear algebraic parameter called *projective dimension* (denoted by  $\text{pd}_{\mathbb{F}}(f)$ , over a field  $\mathbb{F}$ ) defined on a natural graph associated with the Boolean function  $f$ . For a Boolean function  $f : \{0, 1\}^{2n} \rightarrow \{0, 1\}$ , fix a partition of the input bits into two parts of size  $n$  each, and consider the bipartite graph  $G_f(U, V, E)$  defined on vertex sets  $U = \{0, 1\}^n$  and  $V = \{0, 1\}^n$ , as  $(u, v) \in E$  if and only if  $f(uv) = 1$ . We call  $G_f$  as the bipartite realization of  $f$ . For a bipartite graph  $G(U, V, E)$ , the projective dimension of  $G$  over a field  $\mathbb{F}$ , denoted by  $\text{pd}_{\mathbb{F}}(G)$ , is defined as the smallest  $d$  for which there is a vector space  $W$  of dimension  $d$  (over  $\mathbb{F}$ ) and a function  $\phi$  mapping vertices in  $U, V$  to linear subspaces of  $W$  such that for all  $(u, v) \in U \times V$ ,  $(u, v) \in E$  if and only if  $\phi(u) \cap \phi(v) \neq \{0\}$ . We say that  $\phi$  *realizes* the graph  $G$ .

Pudlák and Rödl [12] showed that if  $f$  can be computed by a deterministic branching program of size  $s$ , then  $\text{pd}_{\mathbb{F}}(f) \leq s$  over any field  $\mathbb{F}$ . Thus, in order to establish size lower bounds against branching programs, it suffices to prove lower bounds for projective dimension of explicit family of Boolean functions.

By a counting argument, Pudlák and Rödl in [12] showed that for most Boolean functions  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$ ,  $\text{pd}_{\mathbb{R}}(f)$  is  $\Omega(\sqrt{\frac{2n}{n}})$ . In a subsequent work, the same authors [13] also established an upper bound  $\text{pd}_{\mathbb{R}}(f) = O(\frac{2n}{n})$  for all functions. More recently, Rónyai, Babai and Ganapathy [15] established the same lower bound over all fields. Over finite fields  $\mathbb{F}$ , Pudlák and Rödl [12] also showed (by a counting argument) that there exists a Boolean function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  such that  $\text{pd}_{\mathbb{F}}(f)$  is  $\Omega(\sqrt{2n})$ . However, till date, obtaining an explicit family of Boolean functions (equivalently graphs) achieving such lower bounds remain elusive. The best lower bound for projective dimension for an explicit family of functions is for the inequality function (on  $2n$  bits, the graph is the bipartite complement of the perfect matching) where a lower bound of  $\epsilon n$  for an absolute constant  $\epsilon > 0$  is known [12] over  $\mathbb{R}$ . For a survey on projective dimension and related linear algebraic techniques, refer [13, 9]. However, the best known size lower bound that was achieved using this framework is only  $\Omega(n)$  which is not better than trivial lower bounds.

**Our Results:** Our starting point is the observation that projective assignment appearing in the proof of [12] also has the property that the dimension of the intersection of two subspaces assigned to the vertices is exactly 1, whenever they intersect (See Proposition 2.2(2)). We denote, for a function  $f$ , the variant of projective dimension defined by this property as  $\text{upd}(f)$  (see Section 4). From the above discussion, for any Boolean function  $f$ ,  $\text{pd}(f) \leq \text{upd}(f) \leq \text{bysize}(f)$ . A natural question is whether this restriction helps in proving better lower bounds for the branching programs. By observing properties about the measure of projective dimension, choosing a new candidate function<sup>1</sup>, we demonstrate that the above restriction can help by proving the following quadratic gap between the two measures.

<sup>1</sup> The candidate function is in  $\text{NC}^2$  but unlikely to be in  $\text{NL}$ .

► **Theorem 1.1.** *For any  $d \geq 0$ , for the function  $\text{Sl}_d$  (on  $2d^2$  variables, see Definition 2.3), the projective dimension is exactly equal to  $d$ , while the projective dimension with intersection dimension 1 is  $\Omega(d^2)$ .*

However, this does not directly improve the known branching program size lower bound for  $\text{Sl}_d$ , since it leads to only a linear lower bound on  $\text{upd}(\text{Sl}_d)$ . We demonstrate the weakness of this measure by showing the existence of a function (although not explicit) for which there is an exponential gap between  $\text{upd}$  over any partition and the branching program size (Proposition 5.1). This motivates us to look for variants of projective dimension of graphs, which is closer to the optimal branching program size of the corresponding Boolean function. We observe more properties (see Proposition 2.2) about the subspace assignment from the proof of the upper bound from [12]. We call the projective assignments with these properties *bitwise decomposable projective assignment* and denote the corresponding dimension<sup>2</sup> as  $\text{bitpdim}(f)$  (See Definition 5.2). Thus, for any Boolean function  $f$ ,  $\text{pd}(f) \leq \text{bitpdim}(f)$ . We also show that  $\text{bitpdim}(f) \leq 6 \cdot \text{bysize}(f)$  (Lemma 5.3). To demonstrate the tightness of the definition, we first argue a converse with respect to this new parameter.

► **Theorem 1.2.** *There is an absolute constant  $c > 0$  such that if  $\text{bitpdim}(f_n) \leq d(n)$  for a function family  $\{f_n\}_{n \geq 0}$  on  $2n$  bits, then there is a deterministic branching program of size  $(d(n))^c$  computing it.*

Thus, super-polynomial size lower bounds for branching programs imply super-polynomial lower bounds for  $\text{bitpdim}(f)$ . The function  $\text{Sl}_d$  (on  $2d^2$  input bits – see Definition 2.3) is a natural candidate for proving  $\text{bitpdim}$  lower bounds as the corresponding language is hard<sup>3</sup> for the complexity class  $\text{C=L}$  under logspace Turing reductions.

However, the best known lower bound for branching program size for an explicit family of functions is  $\Omega\left(\frac{n^2}{\log^2 n}\right)$  by Nechiporuk [11] which uses a counting argument on the number of sub-functions. By Theorem 1.2,  $\text{bitpdim}(f)$  (for the same explicit function) is at least  $\Omega\left(\frac{n^{2/c}}{\log^{2/c} n}\right)$ . The constant  $c$  is large<sup>4</sup> and hence implies only weak lower bounds for  $\text{bitpdim}$ . Despite this weak connection, by combining the counting strategy with the linear algebraic structure of  $\text{bitpdim}$ , we show a super-linear lower bound for  $\text{Sl}_d$  matching the branching program size lower bound<sup>5</sup>.

► **Theorem 1.3 (Main Result).** *For any  $d > 0$ ,  $\text{bitpdim}(\text{Sl}_d)$  is at least  $\Omega\left(\frac{d^3}{\log d}\right)$ .*

Theorems 1.1 and 1.3 demonstrate gaps between the  $\text{pd}$  and the new measures considered. In particular, for  $n = d^2$ ,  $\text{pd}(\text{Sl}_d) = O(\sqrt{n})$ ,  $\text{upd}(\text{Sl}_d) = \Omega(n)$ , and  $\text{bitpdim}(\text{Sl}_d) = \Omega\left(\frac{n^{1.5}}{\log n}\right)$ . We remark that Theorem 1.3 implies a size lower bound of  $\Omega\left(\frac{n^{1.5}}{\log n}\right)$  for branching programs computing the function  $\text{Sl}_d$  (where  $n = d^2$ ). However, note that this can also be derived from Nechiporuk's method. For the Element Distinctness function, the above linear algebraic adaptation of Nechiporuk's method for  $\text{bitpdim}$  gives  $\Omega\left(\frac{n^2}{\log^2 n}\right)$  lower bounds (for  $\text{bitpdim}$  and hence for  $\text{bysize}$ ) which matches with the best lower bound that Nechiporuk's method can derive. This shows that our modification of approach in [12] can also achieve the best known lower bounds for branching program size.

<sup>2</sup> We do not use the property that intersection dimension is 1 and hence is incomparable with  $\text{upd}$ .

<sup>3</sup> Assuming  $\text{C=L} \not\subseteq \text{L/poly}$ ,  $\text{Sl}_d$  cannot be computed by deterministic branching programs of size  $\text{poly}(d)$ .

<sup>4</sup> However, the value of  $c$  can be shown to be at most 5. See proof of Theorem 1.2 in Section 5.1.

<sup>5</sup> A lower bound of  $\Omega\left(\frac{d^3}{\log d}\right)$  for the branching program size can also be obtained using Nechiporuk's method.

Continuing the quest for better lower bounds for projective dimension, we study two further restrictions. In these variants of  $\text{pd}$  and  $\text{upd}$ , the subspaces assigned to the vertices must be spanned by standard basis vectors. We denote the corresponding dimensions as  $\text{spd}(f)$  and  $\text{uspd}(f)$  respectively. It is easy to see that for any  $2n$ -bit function, both of these dimensions are upper bounded by  $2^n$ .

We connect these variants to some of the well-studied graph parameters. The *bipartite clique cover number* (denoted by  $bc(G)$ ) is the smallest collection of complete bipartite subgraphs of  $G$  such that every edge in  $G$  is present in some graph in the collection. If we insist that the bipartite graphs in the collection be edge-disjoint, the measure is called *bipartite partition number* denoted by  $bp(G)$ . By definition,  $bc(G) \leq bp(G)$ . These graph parameters are closely connected to communication complexity as well. More precisely,  $\log(bc(G_f))$  is exactly the non-deterministic communication complexity of the function  $f$ , and  $\log(bp(G_f))$  is a lower bound on the deterministic communication complexity of  $f$  (see [6]). In this context, we show the following:

► **Theorem 1.4.** *For any Boolean function  $f$ ,  $\text{spd}(f) = bc(G_f)$  and  $\text{uspd}(f) = bp(G_f)$ .*

Thus, if for a function family, the non-deterministic communication complexity is  $\Omega(n)$ , then we will have  $\text{spd}(f) = 2^{\Omega(n)}$ . Thus, both  $\text{spd}(\text{DISJ})$  and  $\text{uspd}(\text{DISJ})$  are  $2^{\Omega(n)}$ .

## 2 Preliminaries

In this section, we introduce the notations used in the paper. For definitions of basic complexity classes and computational models, we refer the reader to standard textbooks [6, 16].

Unless otherwise stated we work over the field  $\mathbb{F}_2$ . We remark that our arguments do generalize to any finite field. All subspaces that we talk about in this work are linear subspaces. Also  $\vec{0}$  and  $\{0\}$  denote the zero vector, and zero-dimensional space respectively. For a subspace  $U \subseteq \mathbb{F}^n$ , we call the ambient dimension of  $U$  as  $n$ . We denote  $e_i \in \mathbb{F}^n$  as the  $i^{\text{th}}$  standard basis vector with  $i^{\text{th}}$  entry being 1 and rest of the entries being zero.

For a graph  $G(U, V, E)$ , recall the definition of projective dimension of  $G$  over a field  $\mathbb{F}(\text{pd}_{\mathbb{F}}(G))$ , defined in the introduction. For a Boolean function  $f : \{0, 1\}^{2n} \rightarrow \{0, 1\}$ , fix a partition of the input bits into two parts of size  $n$  each, and consider the bipartite graph  $G_f$  defined on vertex sets  $U = \{0, 1\}^n$  and  $V = \{0, 1\}^n$ , as  $(u, v) \in E$  if and only if  $f(uv) = 1$ . A  $\phi$  is said to *realize* the function  $f$  if it *realizes*  $G_f$ . Unless otherwise mentioned, the partition is the one specified in the definition of the function. We denote by  $\text{bysize}(f)$  the number of vertices (including accept and reject nodes) in the optimal branching program computing  $f$ .

► **Theorem 2.1** (Pudlák-Rödl Theorem [12]). *For a Boolean function  $f$  computed by a deterministic branching program of size  $s$  and  $\mathbb{F}$  being any field,  $\text{pd}_{\mathbb{F}}(G_f) \leq s$ .*

The proof of this result proceeds by producing a subspace assignment for vertices of  $G_f$  from a branching program computing  $f$ . We derive the following proposition by a careful analysis of the aforementioned proof in [12].

► **Proposition 2.2.** *For a Boolean function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  computed by a deterministic branching program of size  $s$ , there is a collection of subspaces of  $\mathbb{F}^s$  denoted  $\mathcal{C} = \{U_i^a\}_{i \in [n], a \in \{0, 1\}}$  and  $\mathcal{D} = \{V_j^b\}_{j \in [n], b \in \{0, 1\}}$ , where we associate the subspace  $U_i^a$  with a bit assignment  $x_i = a$  and  $V_j^b$  with  $y_j = b$  such that if we define the map  $\phi$  assigning subspaces from  $\mathbb{F}^s$  to vertices of  $G_f(U, V, E)$  as  $\phi(x) = \text{span}_{1 \leq i \leq n} \{U_i^{x_i}\}$ ,  $\phi(y) = \text{span}_{1 \leq j \leq n} \{V_j^{y_j}\}$ , for  $x \in X, y \in Y$  then the following holds true. Let  $S = \{e_i - e_j \mid i, j \in [s], i \neq j\}$ .*

1. for all  $(u, v) \in U \times V$ ,  $\phi(u) \cap \phi(v) \neq \{0\}$  if and only if  $f(u, v) = 1$ .
2. for all  $(u, v) \in U \times V$ ,  $\dim(\phi(u) \cap \phi(v)) \leq 1$ .
3. For any  $W \in \mathcal{C} \cup \mathcal{D}$ ,  $\exists S' \subseteq S$  such that  $W = \text{span}\{S'\}$ .

We define the following family of functions and family of graphs based on subspaces of a vector space and their intersections.

► **Definition 2.3** ( $Sl_d, \mathcal{P}_d$ ). Let  $\mathbb{F}$  be a finite field. Denote by  $Sl_d$ , the Boolean function defined on  $\mathbb{F}^{d \times d} \times \mathbb{F}^{d \times d} \rightarrow \{0, 1\}$  as for any  $A, B \in \mathbb{F}^{d \times d}$   $Sl_d(A, B) = 1$  if and only if  $\text{rowspan}(A) \cap \text{rowspan}(B) \neq \{0\}$ . Note that the row span is over the field  $\mathbb{F}$  (which, in our case, is  $\mathbb{F}_2$ ). Denote by  $\mathcal{P}_d$ , the bipartite graph  $(U, V, E)$  where  $U$  and  $V$  are the set of all subspaces of  $\mathbb{F}^d$ . And for any  $(I, J) \in U \times V$ ,  $(I, J) \in E \iff I \cap J \neq \{0\}$

We collect the definitions of Boolean functions which we deal with in this work. For  $(x, y) \in \{0, 1\}^n \times \{0, 1\}^n$ ,  $IP_n(x, y) = \sum_{i=1}^n x_i y_i \pmod 2$ ,  $EQ_n(x, y)$  is 1 if  $\forall i \in [n]$   $x_i = y_i$  and is 0 otherwise,  $INEQ_n(x, y) = \neg EQ_n(x, y)$  and  $DISJ_n(x, y) = 1$  if  $\forall i \in [n]$   $x_i \wedge y_i = 0$  and is 0 otherwise. Note that all the functions discussed so far has branching programs of size  $O(n)$  computing them and hence have projective dimension  $O(n)$  by Theorem 2.1.

Let  $m \in \mathbb{N}$  and  $n = 2m \log m$ . The Boolean function, Element Distinctness, denoted  $ED_n$  is defined on  $2m$  blocks of  $2 \log m$  bits,  $x_1, \dots, x_m$  and  $y_1, \dots, y_m$  bits and it evaluates to 1 if and only if all the  $x_i$ s and  $y_i$ s take distinct values when interpreted as integers in  $[m^2]$ . Let  $q$  be a power of prime congruent to 1 modulo 4. Identify elements in  $\{0, 1\}^n$  with elements of  $\mathbb{F}_q^*$ . For  $x, y \in \mathbb{F}_q^*$ , the Paley function  $PAL_n^q(x, y) = 1$  if  $x - y$  is a quadratic residue in  $\mathbb{F}_q^*$  and 0 otherwise.

We observe for any induced subgraph  $H$  of  $G$ , if  $G$  is realized in a space of dimension  $d$ , then  $H$  can also be realized in a space of dimension  $d$ . For any  $d \in \mathbb{N}$ ,  $\mathcal{P}_d$  appears as an induced subgraph of the bipartite realization of  $Sl_d$ . Hence,  $\text{pd}(Sl_d) \geq \text{pd}(\mathcal{P}_d)$ .

### 3 Properties of Projective Dimension

In this section, we observe properties about projective dimension as a measure of graphs and Boolean functions. We start by proving closure properties of projective dimension under Boolean operations  $\wedge$  and  $\vee$ .

► **Lemma 3.1.** *Let  $\mathbb{F}$  be an arbitrary field. For any two functions  $f_1 : \{0, 1\}^{2n} \rightarrow \{0, 1\}$ ,  $f_2 : \{0, 1\}^{2n} \rightarrow \{0, 1\}$ ,  $\text{pd}_{\mathbb{F}}(f_1 \vee f_2) \leq \text{pd}_{\mathbb{F}}(f_1) + \text{pd}_{\mathbb{F}}(f_2)$  and  $\text{pd}_{\mathbb{F}}(f_1 \wedge f_2) \leq \text{pd}_{\mathbb{F}}(f_1) \cdot \text{pd}_{\mathbb{F}}(f_2)$*

The proof is based on direct sum and tensor product of vector spaces. The  $\vee$  part of the above lemma was also observed (without proof) in [13]. We remark that the construction for  $\vee$  is tight up to constant factors. Assume  $n$  is a multiple of 4. Consider the functions  $f(x_1, \dots, x_{\frac{n}{4}}, x_{\frac{n}{2}+1}, \dots, x_{\frac{3n}{4}})$  and  $g(x_{\frac{n}{4}+1}, \dots, x_{\frac{n}{2}}, x_{\frac{3n}{4}+1}, \dots, x_n)$  each of which performs inequality check on the first  $\frac{n}{4}$  and the second  $\frac{n}{4}$  variables. It is easy to see that  $f \vee g$  is the inequality function on  $\frac{n}{2}$  variables  $x_1, \dots, x_{\frac{n}{2}}$  and the next  $\frac{n}{2}$  variables  $x_{\frac{n}{2}+1}, \dots, x_n$ . By the fact that they are computed by  $n$  size branching programs and using Theorem 2.1 (Pudlák-Rödl theorem) we get that  $\text{pd}(f) \leq n$  and  $\text{pd}(g) \leq n$ . Hence by Lemma 3.1,  $\text{pd}(f \vee g) \leq \text{pd}(f) + \text{pd}(g) \leq 2n$ . Lower bound on projective dimension of inequality function comes from [12, Theorem 4], giving  $\text{pd}(f \vee g) \geq \epsilon \cdot \frac{n}{2}$  for an absolute constant  $\epsilon$ . This shows that  $\text{pd}(f \vee g) = \Theta(n)$ . We also cannot expect a general relation connecting  $\text{pd}_{\mathbb{R}}(f)$  and  $\text{pd}_{\mathbb{R}}(\neg f)$  since it is known [12] that  $\text{pd}_{\mathbb{R}}(INEQ_n)$  is  $\Omega(n)$  while  $\text{pd}_{\mathbb{R}}(EQ_n) = 2$ .

We now observe a characterization of bipartite graphs having projective dimension at most  $d$  over  $\mathbb{F}$ .

► **Lemma 3.2** (Characterization). *Let  $G$  be a bipartite graph with no two vertices having same neighborhood,  $\text{pd}(G) \leq d$  if and only if  $G$  is an induced subgraph of  $\mathcal{P}_d$ .*

It follows that  $\text{pd}(\mathcal{P}_d) \leq d$ . Observe that, in any projective assignment, the vertices with different neighborhoods should be assigned different subspaces. For  $\text{pd}(\mathcal{P}_d)$ , all vertices on either partitions have distinct neighborhoods. The number of subspaces of a vector space of dimension  $d - 1$  is strictly smaller than the number of vertices in  $\mathcal{P}_d$ . Thus, we conclude the following theorem.

► **Theorem 3.3.** *For any  $d \in \mathbb{N}$ ,  $\text{pd}(\mathcal{P}_d) = \text{pd}(\text{Sl}_d) = d$ .*

For an  $N$  vertex graph  $G$ , the number of vertices of distinct neighborhood can at most be  $N$ . Thus, the observation that we used to show the lower bound for the graph  $\text{pd}(\mathcal{P}_d)$  cannot be used to obtain more than a  $\sqrt{\log N}$  lower bound for  $\text{pd}(G)$ . Also, for many functions, the number of vertices of distinct neighborhood can be smaller.

We observe that by incurring an additive factor of  $2 \log N$ , any graph  $G$  on  $N$  vertices can be transformed into a graph  $G'$  on  $2N$  vertices such that all the neighborhoods of vertices in one partition are all distinct. Let  $f : \{0, 1\}^{2n} \rightarrow \{0, 1\}$  be such that the neighborhoods of  $G_f$  are not necessarily distinct. We consider a new function  $f'$  whose bipartite realization will have two copies of  $G_f$  namely  $G_1(A_1, B_1, E_1)$  and  $G_2(A_2, B_2, E_2)$  where  $A_1, A_2, B_1, B_2$  are disjoint and a matching connecting vertices in  $A_1$  to  $B_2$  and  $A_2$  to  $B_1$  respectively. Since the matching edges associated with every vertex is unique, the neighborhoods of all vertices are bound to be distinct. Applying Lemma 3.1 and observing that matching (i.e, equality function) has projective dimension at most  $n$ ,  $\text{pd}(f') \leq 2\text{pd}(f) + 2n$ . This shows that to show super-linear lower bounds on projective dimension for  $f$  where the neighborhoods may not be distinct, it suffices to show a super-linear lower bound for  $f'$ .

## 4 Projective Dimension with Intersection Dimension 1

Motivated by the proof of Theorem 2.1, we make the following definition.

► **Definition 4.1** (Projective Dimension with Intersection Dimension 1). A Boolean function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  with the corresponding bipartite graph  $G(U, V, E)$  is said to have projective dimension with intersection dimension 1 (denoted by  $\text{upd}(f)$ )  $d$  over field  $\mathbb{F}$ , if  $d$  is the smallest possible dimension for which there exists a vector space  $K$  of dimension  $d$  over  $\mathbb{F}$  with a map  $\phi$  assigning subspaces of  $K$  to  $U \cup V$  such that

- for all  $(u, v) \in U \times V$ ,  $\phi(u) \cap \phi(v) \neq \{0\}$  if and only if  $(u, v) \in E$ .
- for all  $(u, v) \in U \times V$ ,  $\dim(\phi(u) \cap \phi(v)) \leq 1$ .

By the properties observed in Proposition 2.2,

► **Theorem 4.2.** *For a Boolean function  $f$  computed by a deterministic branching program of size  $s$ ,  $\text{upd}_{\mathbb{F}}(f) \leq s$  for any field  $\mathbb{F}$ .*

Thus, it suffices to prove lower bounds for  $\text{upd}(f)$  in order to obtain branching program size lower bounds. We now proceed to show lower bounds on  $\text{upd}$ . Our approaches use the fact that the adjacency matrix of  $\mathcal{P}_d$  has high rank.

► **Lemma 4.3.** *Let  $M$  be the bipartite adjacency matrix of  $\mathcal{P}_d$ , then  $\text{rank}(M) \geq \binom{d}{d/2}_q \geq q^{\frac{d^2}{4}}$*

**Proof.** For  $0 \leq i \leq k \leq d$ , and subspace  $I, K \subseteq_s \mathbb{F}_q^d$  with  $\dim(I) = i, \dim(K) = k$ , define matrix  $\overline{W}_{ik}$  over  $\mathbb{R}$  as  $\overline{W}_{ik}(I, K) = 1$  if  $I \cap K = \{0\}$  and 0 otherwise. This matrix has dimension  $\binom{d}{i}_q \times \binom{d}{k}_q$ .

Consider the submatrix  $M_i$  of  $M$  with rows and columns indexed by subspaces of dimension exactly  $i$ . Observe that  $\overline{W_{ii}} = J - M_i$  where  $J$  is an all ones matrix of appropriate order. These matrices are well-studied (see [5]). Closed form expressions for eigenvalues are computed in [3, 10] and the eigenvalues are known to be non-zero. Hence for  $0 \leq i \leq d/2$  the matrix  $\overline{W_{ii}}$  has rank  $\binom{d}{i}_q$ . Since  $\overline{W_{ii}} = J - M_i$ ,  $\text{rank}(M_i) \geq \text{rank}(\overline{W_{ii}}) - 1$ . This shows that  $\text{rank}(M) \geq \text{rank}(M_i) = \binom{d}{i}_q$  for all  $i$  such that  $2i \leq d$ . Choosing  $i = d/2$  gives  $\text{rank}(M) \geq \binom{d}{d/2}_q - 1 \geq q^{\frac{d^2}{4}} - 1$ .  $\blacktriangleleft$

We now present two approaches for showing lower bounds on  $\text{upd}(f)$  – one using intersection families of vector spaces and the other using rectangle arguments on  $M_f$ .

**Lower Bound for  $\text{upd}(\mathcal{P}_d)$  using intersecting families of vector spaces:** To prove a lower bound on  $\text{upd}(\mathcal{P}_d)$  we define a matrix  $N$  from a projective assignment with intersection dimension 1 for  $\mathcal{P}_d$ , such that it is equal to  $(q-1)M$ . Let  $D = \text{upd}(\mathcal{P}_d)$ . We first show that  $\text{rank}(N)$  is at most  $1 + \binom{D}{1}_q$ . Then by Lemma 4.3 we get that  $\text{rank}(N)$  is at least  $q^{\frac{d^2}{4}}$ . Let  $\mathcal{G} = \{G_1, \dots, G_m\}$ ,  $\mathcal{H} = \{H_1, \dots, H_m\}$  be the subspace assignment with intersection dimension 1 realizing  $\mathcal{P}_d$  with dimension  $D$ .

► **Lemma 4.4.** *For any polynomial  $p$  in  $q^x$  of degree  $s$ , with matrix  $N$  of order  $|\mathcal{G}| \times |\mathcal{H}|$  defined as  $N[G_r, H_t] = p(\dim(G_r \cap H_t))$  with  $G_r \in \mathcal{G}$ ,  $H_t \in \mathcal{H}$ , then  $\text{rank}(N) \leq \sum_{i=0}^s \binom{D}{i}_q$*

**Proof.** This proof is inspired by the proof in [4] of a similar claim where a non-bipartite version of this lemma is proved. To begin with, note that  $p$  is a degree  $s$  polynomial in  $q^x$ , and hence can be written as a linear combination of polynomials  $p_i = \binom{x}{i}_q$ ,  $0 \leq i \leq s$ . Let the linear combination be given by  $p(x) = \sum_{i=0}^s \alpha_i p_i(x)$ . For  $0 \leq i \leq s$  define a matrix  $N_i$  with rows and columns indexed respectively by  $\mathcal{G}$ ,  $\mathcal{H}$  defined as  $N_i[G_r, H_s] = p_i(\dim G_r \cap H_s)$ . By definition of  $N_i$ ,  $N = \sum_{i \in [s]} \alpha_i N_i$ .

To bound the rank of  $N_i$ 's we introduce the following families of inclusion matrices. For any  $j \in [D]$ , consider two matrices  $\Gamma_j$  corresponding to  $\mathcal{G}$  and  $\Delta_j$  corresponding to  $\mathcal{H}$  defined as  $\Gamma_j(G, I) = 1$  if  $\dim(I) = j$ ,  $G \in \mathcal{G}$ ,  $I \subseteq_s G$  and 0 otherwise.  $\Delta_j(H, I) = 1$  if  $\dim(I) = j$ ,  $H \in \mathcal{H}$ ,  $I \subseteq_s H$  and 0 otherwise. Note that rank of these matrices are upper bounded by the number of columns which is  $\binom{D}{j}_q$ . We claim that for any  $i \in \{0, 1, \dots, s\}$ ,  $\text{rank}(N_i) \leq \binom{D}{i}_q$ . This completes the proof since  $N = \sum_{i \in [s]} \alpha_i N_i$ .

To prove the claim, let  $\mathcal{F}_i$  denote the set of all  $i$  dimensional subspace of  $\mathbb{F}_q^D$ . We show that  $N_i = \Gamma_i \Delta_i^T$ . Hence  $\text{rank}(N_i) \leq \min\{\text{rank}(\Gamma_i), \text{rank}(\Delta_i)\} \leq \binom{D}{i}_q$ . For  $(G_r, H_t) \in \mathcal{G} \times \mathcal{H}$ ,  $\Gamma_i \Delta_i^T(G_r, H_t) = \sum_{I \in \mathcal{F}_i} \Gamma_i(G_r, I) \Delta_i^T(I, H_t) = \sum_{I \in \mathcal{F}_i} \Gamma_i(G_r, I) \Delta_i(H_t, I) = \sum_{I \in \mathcal{F}_i} [I \subseteq_s G_r] \wedge [I \subseteq_s H_t] = \sum_{I \in \mathcal{F}_i} [I \subseteq_s G_r \cap H_t] = \binom{\dim(G_r \cap H_t)}{i}_q = N_i(G_r, H_t)$   $\blacktriangleleft$

We apply Lemma 4.4 on  $N$  defined via  $p(x) = q^x - 1$  with  $s = 1$ , to get  $q^{d^2/4} \leq \binom{d}{d/2}_q \leq 1 + \binom{D}{1}_q = 1 + (q^D - 1)/(q - 1)$ . By definition,  $\text{rank}(N) = \text{rank}(M)$ . This gives that  $D = \Omega(d^2)$  and proves Theorem 1.1.

**Lower Bound for  $\text{upd}(\mathcal{P}_d)$  from Rectangle Arguments:** We now give an alternate proof of for Theorem 1.1 using combinatorial rectangle arguments.

► **Lemma 4.5.** *For  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  with  $M_f$  denoting the bipartite adjacency matrix of  $G_f$ ,  $\text{rank}_{\mathbb{F}}(M_f) \leq q^{O(\text{upd}_{\mathbb{F}}(f))}$  where  $\mathbb{F}$  is a finite field of size  $q$ .*



**Proof.** Let  $\phi$  be a subspace assignment realizing  $f$  of dimension  $d$  with intersection dimension 1. Let  $S(v)$  for  $v \in \mathbb{F}_q^d$  denote  $\{(a, b) \in \{0, 1\}^n \times \{0, 1\}^n \mid \phi(a) \cap \phi(b) = \text{span}\{v\}\}$ . Also let  $M_v$  denote the matrix representation of  $S(v)$ . That is,  $M_v(a, b) = 1 \iff (a, b) \in S(v)$ . Consider all 1 dimensional subspaces which appear as intersection space for some input  $(x, y)$ . Fix a basis vector for each space and let  $T$  denote the collection of basis vectors of all the intersection spaces. Note that for any  $(x, y) \in f^{-1}(1)$ , there is a unique  $v \in \mathbb{F}_q^d$  (up to scalar multiples) such that  $(x, y) \in S(v)$  for otherwise intersection dimension exceeds 1. Then  $M_f = \sum_{v \in T} M_v$ . Now,  $\text{rank}(M_f) \leq \sum_{v \in T} \text{rank}(M_v)$ . Since  $\text{rank}(M_v) = 1$ ,  $\text{rank}(M_f) \leq |T|$ . The fact that the number of 1 dimensional spaces in  $\mathbb{F}^d$  can be at most  $\frac{q^d-1}{q-1}$  completes the proof. Note that the rank of  $M_f$  can be over any field (we choose  $\mathbb{R}$ ). ◀

We get an immediate corollary. Any function  $f$ , such that the adjacency matrix of  $M_f$  of the bipartite graph  $G_f$  is of full rank  $2^n$  over some field must have  $\text{upd}(f) = \Omega(n)$ . There are several Boolean functions with this property, well-studied in the context of communication complexity (see textbook [8]). Hence, we have for  $f \in \{\text{IP}_n, \text{EQ}_n, \text{INEQ}_n, \text{DISJ}_n, \text{PAL}_n^q\}$ ,  $\text{upd}_{\mathbb{F}}(f)$  is  $\Omega(n)$  for any finite field  $\mathbb{F}$ .

For arguing about  $\text{PAL}_n^q$ , it can be observed that the graph is strongly regular (as  $q \equiv 1 \pmod{4}$ ) and hence the adjacency matrix has full rank over  $\mathbb{R}$  [2]. Except for  $\text{PAL}_n^q$ , all the above functions have  $O(n)$  sized deterministic branching programs computing them and hence the Pudlák-Rödl theorem (Theorem 2.1) gives that  $\text{upd}$  for these functions (except  $\text{PAL}_n^q$ ) are  $O(n)$  and hence the above lower bound is indeed tight.

From Lemma 4.3, it follows that the function  $\text{Sl}_d$  also has rank  $2^{\Omega(d^2)}$ . To see this, it suffices to observe that  $\mathcal{P}_d$  appears as an induced subgraph in the bipartite realization of  $\text{Sl}_d$ . Thus,  $\text{upd}(\text{Sl}_d)$  is  $\Omega(d^2)$ . We proved in Theorem 3.3 that  $\text{pd}(\text{Sl}_d) = d$ . This establishes a quadratic gap between the two parameters. This completes the proof of Theorem 1.1.

Let  $D(f)$  denote the deterministic communication complexity of the Boolean function  $f$ . We observe that the rectangle argument used in the proof of Lemma 4.5 is similar to the matrix rank based lower bound arguments for communication complexity. This yields the Proposition 4.6. If  $\text{upd}(f) \leq D$ , the assignment also gives a partitioning of the 1s in  $M_f$  into at most  $\frac{q^D-1}{q-1}$  1-rectangles. However, it is unclear whether this immediately gives a similar partition of 0s into 0-rectangles as well. Notice that if  $D(f) \leq d$ , there are at most  $2^d$  monochromatic rectangles (counting both 0-rectangles and 1-rectangles) that cover the entire matrix. However, our proof does not exploit this difference.

► **Proposition 4.6.** *For a Boolean function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  and a finite field  $\mathbb{F}$ ,  $\text{upd}_{\mathbb{F}}(f) \leq 2^{D(f)}$  and  $D(f) \leq (\text{pd}_{\mathbb{F}}(f))^2 \log |\mathbb{F}|$*

**Proof.** We give a proof of the first inequality. Any deterministic communication protocol computing  $f$  of cost  $D(f)$ , partitions  $M_f$  into  $k$  rectangles where  $k \leq 2^{D(f)}$  rectangles. Define  $f_i : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  for each rectangle  $R_i$   $i \in [k]$ , such that  $f_i(x, y) = 1$  iff  $(x, y) \in R_i$ . Note that  $\text{upd}_{\mathbb{F}}(f_i) = 1$  and  $f = \bigvee_{i=1}^k f_i$ . For any  $(x, y) \in \{0, 1\}^n \times \{0, 1\}^n$  if  $f(x, y) = 1$ , there is exactly one  $i \in [k]$  where  $f_i(x, y) = 1$ . Hence for each  $j \in [k], j \neq i$ , the intersection vector corresponding to the edge  $(x, y)$  in the assignment of  $f_j$  is trivial. Hence the assignment obtained by applying Lemma 3.1, to  $f_1, \bigvee f_2 \vee \dots \vee f_k$  will have the property that for any  $(x, y)$  with  $f(x, y) = 1$ , the intersection dimension is 1. Hence  $\text{upd}_{\mathbb{F}}(f) \leq k \leq 2^{D(f)}$ . To prove the second inequality, consider the protocol where Alice sends the subspace associated with her input as a  $\text{pd}_{\mathbb{F}}(f) \times \text{pd}_{\mathbb{F}}(f)$  matrix. ◀

Note that the first inequality is tight, up to constant factors in the exponent. To see this, consider the function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  whose  $\text{pd}_{\mathbb{F}}(f) = \Omega(2^{n/2})$  [12, Proposition



1] and note that  $D(f)$  for any  $f$  is at most  $n$ . Tightness of second inequality is witnessed by  $Sl_d$  since by Lemma 4.3  $D(Sl_d) = \Omega(d^2)$  while  $\text{pd}(Sl_d) = d$ .

## 5 Bitwise Decomposable Projective Dimension

The restriction of intersection dimension being 1, although potentially useful for lower bounds for branching program size, does not capture the branching program size exactly. We start the section by demonstrating a function where the gap is exponential. We show the existence of a Boolean function  $f$  such that the size of the optimal branching program computing it is very high but has a very small projective assignment with intersection dimension 1 for any balanced partition of the input.

► **Proposition 5.1** (Implicit in Remark 1.30 of [6]). *There exist a function  $f : \{0, 1\}^n \times \{0, 1\}^n$  that requires size  $\Omega(\frac{2^n}{n})$  for any branching program computing  $f$  but the  $\text{upd}(f) \leq O(n)$  for any balanced partitioning of the input into two parts.*

The above proposition can be shown by adapting the counting argument presented in Remark 1.30 of [6].

### 5.1 A Characterization for Branching Program Size

Motivated by strong properties observed in Proposition 2.2, we make the following definition.

► **Definition 5.2** (Bitwise Decomposable Projective Dimension). Let  $f$  be a Boolean function on  $2n$  bits and  $G_f$  be its bipartite realization. The bipartite graph  $G_f(X, Y, E)$  is said to have *bit projective dimension*,  $\text{bitpdim}(G) \leq d$ , if there exists a collection of subspaces of  $\mathbb{F}_2^d$  denoted  $\mathcal{C} = \{U_i^a\}_{i \in [n], a \in \{0,1\}}$  and  $\mathcal{D} = \{V_j^b\}_{j \in [n], b \in \{0,1\}}$  where a projective assignment  $\phi$  is obtained by associating subspace  $U_i^a$  with a bit assignment  $x_i = a$  and  $V_j^b$  with  $y_j = b$  satisfying the conditions listed below.

1. for all  $(x, y) \in \{0, 1\}^n \times \{0, 1\}^n$ ,  $\phi(x) = \text{span}_{1 \leq i \leq n} \{U_i^{x_i}\}$ ,  $\phi(y) = \text{span}_{1 \leq j \leq n} \{V_j^{y_j}\}$  and  $f$  is realized by  $\phi$ .
2. Let  $S = \{e_i - e_j \mid i, j \in [d], i \neq j\}$ . For any  $W \in \mathcal{C} \cup \mathcal{D}$ ,  $\exists S' \subseteq S$  such that  $W = \text{span}\{S'\}$ .
3. for any  $S_1, S_2 \subseteq ([n] \times \{0, 1\})$  such that  $S_1 \cap S_2 = \emptyset$ ,  $\text{span}_{(i,a) \in S_1} \{U_i^a\} \cap \text{span}_{(j,b) \in S_2} \{U_j^b\} = \{0\}$ .

Same property must hold for subspaces in  $\mathcal{D}$ .

We show that the new parameter bitwise decomposable projective dimension ( $\text{bitpdim}$ ) tightly characterizes the branching program size, up to constants in the exponent.

► **Lemma 5.3.** *Suppose  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  has deterministic branching program of size  $s$  then  $\text{bitpdim}(f) \leq 6s$*

We show that given a  $\text{bitpdim}$  assignment for a function  $f$ , we can construct a branching program computing  $f$ .

► **Theorem 5.4** (Theorem 1.2 restated). *For a Boolean function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  with  $\text{bitpdim}(f) \leq d$ , there exists a deterministic branching program computing  $f$  of size  $d^c$  for some absolute constant  $c$ .*

**Proof.** Consider the subspace associated with the variables  $\mathcal{C}, \mathcal{D}$  of the  $\text{bitpdim}$  assignment as the advice string. These can be specified by a list of  $n$  basis matrices each of size  $d^2$ . Since  $d = \text{bitpdim}(f) = \text{poly}(n)$ , the advice string is  $\text{poly}(n)$  sized and depends only on  $n$ .

We construct a deterministic branching program computing  $f$  as follows. On input  $x, y$ , from the basis matrices in  $\mathcal{C}, \mathcal{D}$ , construct an undirected graph<sup>6</sup>  $G^*$  with all standard basis vectors in  $\mathcal{C}, \mathcal{D}$  as vertices and add an edge between two vertices  $u, v$  if  $e_u - e_v \in U_i^{x_i}$  or  $e_u - e_v \in V_j^{y_j}$  for  $i, j \in [n]$ . For input  $x, y$ ,  $f(x, y) = 1$  iff  $G^*$  has a cycle. To see this, let  $C = C_1 \cup C_2$  be a cycle in  $G^*$  where  $C_1$  consists of edges from basis matrices in  $\mathcal{C}$  and  $C_2$  contain edges from basis matrices in  $\mathcal{D}$ . Note that if one of  $C_1$  or  $C_2$  is empty then there is a cycle consisting only of vectors from  $\mathcal{C}$  which implies a linear dependence among vectors in  $\mathcal{C}$ . But this contradicts Property 3 of bitpdim assignment. Hence both  $C_1$  and  $C_2$  are non-empty. Then, it must be that  $\sum_{(u,v) \in C_1} e_u - e_v + \sum_{(w,z) \in C_2} e_w - e_z = 0$ . Hence  $\sum_{(u,v) \in C_1} e_u - e_v = -\sum_{(w,z) \in C_2} e_w - e_z$ . Hence we get a vector in the intersection which gives  $f(x, y) = 1$ . Note that if  $f(x, y) = 1$ , then clearly there is a non-zero intersection vector. If we express this vector in terms of basis, we get a cycle in  $G^*$ .

Hence, to check if  $f$  evaluates to 1, it suffices check if there is a cycle in  $G^*$  which is solvable in  $L$  using Reingold's algorithm [14]. The log-space algorithm can also be converted to an equivalent branching program of size  $n^c$  for a constant<sup>7</sup>  $c$ . ◀

Assuming  $C=L \not\subseteq L/\text{poly}$ , the function  $\text{Sl}_d$  (a language which is hard for  $C=L$  under Turing reductions) cannot be computed by deterministic branching programs of polynomial size. Thus, using Theorem 1.2, we conclude that the function  $\text{Sl}_d$  is a candidate function (under standard complexity theoretic assumptions) for super-polynomial bitpdim lower bounds.

## 5.2 Lower Bounds for Bitwise Decomposable Projective dimension

From the results of the previous section, it follows that size lower bounds for branching programs do imply lower bounds for bitwise decomposable projective dimension as well. As mentioned in the introduction, the lower bounds that Theorem 1.2 can give for bitwise decomposable projective dimension are only known to be sub-linear.

To prove super-linear lower bounds for bitwise decomposable projective dimension, we show that Nechiporuk's method [11] can be adapted to our linear algebraic framework (thus proving Theorem 1.3 from the introduction). The overall idea is the following: given a function  $f$  and a bitpdim assignment  $\phi$ , consider the restriction of  $f$  denoted  $f_\rho$  where  $\rho$  fixes all variables except the ones in  $T_i$  to 0 or 1 where  $T_i$  is some subset of variables in the left partition. For different restrictions  $\rho$ , we are guaranteed to get at least  $c_i(f)$  different functions. We show that for each restriction  $\rho$ , we can obtain an assignment from  $\phi$  realizing  $f_\rho$ . Hence the number of different bitpdim assignments for  $\rho$  restricted to  $T_i$  is at least the number of sub functions of  $f$  which is at least  $c_i(f)$ . Let  $d_i$  be the ambient dimension of the assignment when restricted to  $T_i$ . By using the structure of bitpdim assignment, we count the number of assignments possible and use this relation to get a lower bound on  $d_i$ . Now repeating the argument with disjoint  $T_i$ , and by observing that the subspaces associated with  $T_i$ 's are disjoint, we get a lower bound on  $d$  as  $d = \sum_i d_i$ .

► **Theorem 5.5.** *For a Boolean function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  on  $2n$  variables, let  $T_1, \dots, T_m$  are partition of variables to  $m$  blocks of size  $r_i$  on the first  $n$  variables. Let  $c_i(f)$  be the number of distinct sub functions of  $f$  when restricted to  $T_i$ , then  $\text{bitpdim}(f) \geq \sum_{i=1}^m \frac{\log c_i(f)}{\log(\log c_i(f))}$*

<sup>6</sup> Note that this is not a deterministic branching program.

<sup>7</sup> Using more space efficient methods than [14], the constant  $c$  can be estimated to be at most 5.

**Proof.** Let  $(x, y)$  denote the  $2n$  input variables of  $f$  and  $\rho : \{x_1, \dots, x_n, y_1, \dots, y_n\} \rightarrow \{0, 1, *\}$  be a map that leaves only variables in  $T_i$  unfixed. Let  $\phi$  be a bitpdim assignment realizing  $f$  and let  $G_f(X, Y, Z)$  denote the bipartite realization of  $f$ . Let  $\mathcal{C} = \{U_i^a\}_{i \in [n], a \in \{0,1\}}$ ,  $\mathcal{D} = \{V_j^b\}_{j \in [n], b \in \{0,1\}}$  be the associated collection of subspaces. Let  $\rho$  be a restriction that does not make  $f_\rho$  a constant and  $(x, y) \in \{0, 1\}^n \times \{0, 1\}^n$  which agrees with  $\rho$ . We use  $x, y$  to denote both variables as well as assignment. From now on, we fix an  $i$  and a partition  $T_i$ .

Define  $L = \text{span}_{i \in [n], \rho(i) \neq *} \{U_i^{\rho(i)}\}$  and  $R = \text{span}_{j \in [n]} \{V_j^{\rho(n+j)}\}$ . For any  $x \in \{0, 1\}^n$  that agrees with  $\rho$  on the first  $n$  bits, define  $Z^x = \text{span}_{j \in T_i} \{U_j^{x_i}\}$ . Note that for any  $(x, y)$ , which agrees with  $\rho$ , has  $\phi(x) = L + Z^x$  and  $\phi(y) = R$ . For any  $f_{\rho_1} \not\equiv f_{\rho_2}$ ,  $G_{f_{\rho_1}} \neq G_{f_{\rho_2}}$ . Hence the number of bitpdim assignments is at least the number of different sub functions. We need to give a bitpdim assignment for  $G_{f_\rho}(V_1, V_2, E)$  where  $V_1 = \{x \mid x \text{ agrees with } \rho\}$ ,  $V_2 = \{y\}$  where  $y = \rho_{[n+1, \dots, 2n]}$  and  $E = \{(x, y) \mid x \in V_1, y \in V_2, f(x, y) = 1\}$ . We use the following property to come up with such an assignment.

► **Property 5.6.** Let  $\rho$  be a restriction which does not make the function  $f$  constant and which fixes all the variables  $y_1, \dots, y_n$ . For all such  $\rho$  and  $\forall x, y \in \{0, 1\}^n$  which agrees with  $\rho$ , any non-zero  $w \in \phi(x) \cap \phi(y)$ , where  $w = u + v$  with  $u \in L$  and  $v \in Z^x$  must satisfy  $v \neq \vec{0}$ .

**Proof.** Let there exists an intersection vector  $w \in (L + Z^x) \cap R$  with  $w = u + v$ ,  $u \in L$  and  $v \in Z^x$  and  $v = \vec{0}$ . Since  $\vec{0} \in Z^{\hat{x}}$  for any  $\hat{x}$ ,  $w = u + \vec{0}$  is in  $L + Z^{\hat{x}}$  and  $R$ . Thus the function after restriction  $\rho$  is a constant. This contradicts the choice of  $\rho$ . ◀

The assignment  $\psi_\rho$  for  $G_{f_\rho}$  defined as:  $\psi_\rho(x) = Z^x$  and  $\psi_\rho(y) = \text{span}_{x \in V_1} \{\Pi_{Z^x}(R \cap (L + Z^x))\}$ . Note that for  $(x, y) \in V_1 \times V_2$ ,  $f_\rho(x) = f(x, y)$ . Following claim shows that  $\psi_\rho$  realize  $f_\rho$ .

► **Claim 5.7.** For any  $(x, y) \in V_1 \times V_2$ ,  $f(x, y) = 1$  if and only if  $\psi_\rho(x) \cap \psi_\rho(y) \neq \{0\}$ .

**Proof.** For any  $(x, y) \in X \times Y$ ,  $\phi(x) \cap \phi(y) \neq \{0\}$  if and only if  $f(x, y) = 1$ . Since  $V_1 \subseteq X$  and  $V_2 \subseteq Y$ , it suffices to prove:  $\forall (x, y) \in V_1 \times V_2$ ,  $\psi_\rho(x) \cap \psi_\rho(y) \neq \{0\} \iff \phi(x) \cap \phi(y) \neq \{0\}$ .

We first prove that  $\psi_\rho(x) \cap \psi_\rho(y) \neq \{0\}$  implies  $\phi(x) \cap \phi(y) \neq \{0\}$ . Let  $v$  be a non-zero vector in  $\psi_\rho(x) \cap \psi_\rho(y)$ . By definition of  $\psi_\rho(x)$ ,  $v \in Z^x$ . By definition of  $\psi_\rho(y)$ , there exists a non-empty  $J \subseteq V_1$  such that  $v = \sum_{\hat{x} \in J} v_{\hat{x}}$  where  $v_{\hat{x}} \in Z^{\hat{x}}$ . Also for every  $\hat{x} \in J$ , there exists a  $u_{\hat{x}} \in L$  such that  $w_{\hat{x}} = u_{\hat{x}} + v_{\hat{x}}$  and  $w_{\hat{x}} \in R$ . Define  $u$  to be  $\sum_{\hat{x} \in J} u_{\hat{x}}$ . Since each  $u_{\hat{x}}$  is in  $L$ ,  $u$  is also in  $L$ . Hence  $w = u + v$  is in  $L + Z^x$ . Substituting  $u$  with  $\sum_{\hat{x} \in J} u_{\hat{x}}$  and  $v$  with  $\sum_{\hat{x} \in J} v_{\hat{x}}$  we get that  $w = \sum_{\hat{x} \in J} u_{\hat{x}} + v_{\hat{x}} = \sum_{\hat{x} \in J} w_{\hat{x}}$ . Since each  $w_{\hat{x}} \in R$ ,  $w \in R$ . Hence  $w \in R \cap (L + Z^x)$  and  $w$  is non-zero as  $J$  is non-empty.

Now we prove that  $\phi(x) \cap \phi(y) \neq \{0\}$  implies  $\psi_\rho(x) \cap \psi_\rho(y) \neq \{0\}$ . Let  $w$  be non zero vector in  $\phi(x) \cap \phi(y)$  with  $w = u + v$  where  $u \in L$  and  $v \in Z^x$ . By Property 5.6 we have  $v \neq \vec{0}$ . By definition  $v \in \psi_\rho(y)$ . Along with  $v \in Z^x$ , we get  $\psi_\rho(x) \cap \psi_\rho(y) \neq \{0\}$ . ◀

Let  $Z = \text{span}_{j \in T_i} \{U_j^0 + U_j^1\}$ . We now prove that subspace assignment on the only vertex in the right partition of  $G_\rho$  which is  $\text{span}_{x \in V_1} \{\Pi_{Z^x}(R)\}$  is indeed  $\Pi_Z(R)$ .

► **Claim 5.8.**  $\Pi_Z(R) = \text{span}_{x \in V_1} \{\Pi_{Z^x}(R)\}$ .

**Proof.** We show  $\text{span}_{x \in V_1} \{\Pi_{Z^x}(R)\} \subseteq \Pi_Z(R)$ . Note that  $\text{span}_{x \in V_1} \{\Pi_{Z^x}(R)\} = \text{span}_{x \in V_1, w \in R} \{\Pi_{Z^x}(w)\}$ . For an arbitrary  $x \in V_1$  and  $w \in R$ , let  $v = \Pi_{Z^x}(w)$ . By definition of  $Z^x$  and the fact that

$\{U_i^b\}_{i \in [n], b \in \{0,1\}}$  are disjoint,  $\Pi_{Z^x}(w) = +_{i \in [n], \rho(i)=*} \Pi_{U_i^{x_i}}(w)$ . As  $Z = \text{span}\{U_j^0 + U_j^1\}_{j \in T_i}$ , every

$\Pi_{U_i^{x_i}}(w) \in \Pi_Z(R)$ . Hence the span is also in  $\Pi_Z(R)$ .

Now we show that  $\Pi_Z(R) \subseteq \text{span}_{x \in V_1}\{\Pi_{Z^x}(R)\}$ . Let  $T_i = \{i_1, \dots, i_k\}$ . For  $1 \leq j \leq k$  define  $x^j$  to be  $x + e^j$  where  $x \in \{0,1\}^n$  agrees with  $\rho$  and for any index  $i \in [n]$  with  $\rho(i) = *$ ,  $x_i = 0$  and  $e^j \in \{0,1\}^n$  is 0 at every index other than  $i_j$ . Note that for any  $j_1 \neq j_2, j_1, j_2 \in T_i$ ,  $Z^{x^{j_1}} \cap Z^{x^{j_2}} = \{0\}$  by Property 3 of Definition 5.2) Also note that  $\text{span}_{j \in T_i}\{Z^{x^j}\} = \text{span}_{j \in T_i}\{U_j^{x^j}\} = Z$ . Hence,  $\Pi_Z(R) = \text{span}_{j \in T_i}\{\Pi_{Z^{x^j}}(R)\}$ . But  $\text{span}_{j \in T_i}\{\Pi_{Z^{x^j}}(R)\} \subseteq \text{span}_{x \in V_1}\{\Pi_{Z^x}(R)\}$ . ◀

For any  $\rho$ , which fixes all variables outside  $T_i$ ,  $Z$  is the same. And since there is only one vertex on the right partition, for different  $\rho, \rho'$ ,  $\Pi_Z(R_\rho) = \Pi_Z(R_{\rho'})$  implies  $\psi_\rho = \psi_{\rho'}$ . Hence to count the number of different  $\psi_\rho$ 's for different  $f_\rho$ 's it is enough to count the number of different  $\Pi_Z(R)$ . To do so, we claim the following property on  $\Pi_Z(R)$ .

► **Property 5.9.** *Let  $S = \{e_u - e_v \mid e_u - e_v \in Z\}$ . Then there exists a subset  $S'$  of  $S$  such that all the vectors in  $S'$  are linearly independent and  $\Pi_Z(R) = \text{span}\{S'\}$ .*

**Proof.** By the property of the bitpdim assignment,  $\forall i \in [n]$  and  $\forall b \in \{0,1\}$ ,  $V_i^b = \text{span}\{F_i^b\}$  where  $F_i^b$  is a collection of difference of standard basis vectors. Recall that  $R = \text{span}\{V_j^{\rho(n+j)}\}_{j \in [n]}$ .

Let  $F = \{(e_u - e_v) \mid e_u - e_v \in F_j^{\rho(n+j)}, j \in [n]\}$ . Since projections are linear maps and the fact that  $F_j^{\rho(n+j)}$  spans  $V_j^{\rho(n+j)}$  we get that,  $\Pi_Z(R) = \text{span}_{w \in F}\{\Pi_Z(w)\}$ . Since  $Z$  is also a span of difference of standard basis vectors,  $\Pi_Z(e_u - e_v)$  is one of  $\vec{0}$ ,  $e_u - e_w$  or  $e_w - e_v$  where  $e_w$  is some standard basis vector in  $Z$ . Let  $S'' = \cup_{e_u - e_v \in F} \Pi_Z(e_u - e_v)$ . Hence  $S'' \subseteq S$ . Clearly,  $\text{span}_{e_u - e_v \in S''}\{e_u - e_v\} = \Pi_Z(R)$ . Choose  $S'$  as a linear independent subset of  $S''$ . ◀

Property 5.9 along with the fact that  $\Pi_Z(R)$  is a subspace of  $Z$ , gives us that the number of different  $\Pi_Z(R)$  is upper bounded by number of different subsets  $S'$  of  $S$  such that  $|S'| = d_i$  where  $d_i = \dim(Z)$ . As  $S'$  is a set of difference of standard basis vectors from  $Z$ ,  $|S'| \leq \binom{d_i}{2}$ . Thus the number of different such  $S'$  are at most  $\sum_{k=0}^{d_i} \binom{d_i}{k} = 2^{O(d_i \log d_i)}$ .

Hence the number of restrictions  $\rho$  (that leaves  $T_i$  unfixed) and leading to different  $f_\rho$  is at most  $2^{O(d_i \log d_i)}$ . But the number of such restrictions  $\rho$  is at least  $c_i(f)$ . Hence  $2^{O(d_i \log d_i)} \geq c_i(f)$  giving  $d_i = \Omega\left(\frac{\log c_i(f)}{\log(\log c_i(f))}\right)$ . Using  $d = \sum_i d_i$  completes the proof. ◀

Theorem 5.5 gives a super linear lower bound for Element Distinctness function. From a manuscript by Beame et.al, ([1], See also [6], Chapter 1), we have  $c_i(ED_n) \geq 2^{n/2}/n$ . Hence applying this count to Theorem 5.5, we get that  $d \geq \Omega\left(\frac{n}{\log n} \cdot \frac{n}{\log n}\right) = \Omega\left(\frac{n^2}{(\log n)^2}\right)$ .

Now we apply this to our context. To get a lower bound using framework described above it is enough to count the number of sub-functions of  $\text{Sl}_d$ .

► **Lemma 5.10.** *For any  $i \in [d]$ , there are  $2^{\Omega(d^2)}$  different restrictions  $\rho$  of  $\text{Sl}_d$  which fixes all entries other than  $i$ th row of the  $d \times d$  matrix in the left partition.*

**Proof.** Fix any  $i \in [d]$ . Let  $S$  be a subspace of  $\mathbb{F}_2^d$ . Define  $\rho_S$  to be  $\text{Sl}_d(\mathbf{A}, B)$  where  $B$  is a matrix whose rowspace is  $S$ . And  $\mathbf{A}$  is the matrix whose all but  $i$ th row is 0's and  $i$ th row consists of variables  $(x_{i_1}, \dots, x_{i_n})$ . Thus for any  $v \in \{0,1\}^d$ , rowspace of  $\mathbf{A}(x)$  is  $\text{span}\{v\}$ .

We claim that for any  $S, S' \subseteq \mathbb{F}_2^d$  where  $S \neq S'$ ,  $(\text{Sl}_d)_{\rho_S} \not\equiv (\text{Sl}_d)_{\rho_{S'}}$ . By definition  $(\text{Sl}_d)_{\rho_S} \equiv \text{Sl}_d(\mathbf{A}, B)$  and  $(\text{Sl}_d)_{\rho_{S'}} \equiv \text{Sl}_d(\mathbf{A}, B')$  where  $B$  and  $B'$  are matrices whose rowspaces

are  $S$  and  $S'$  respectively. Since  $S \neq S'$  there is at least one vector  $v \in \mathbb{F}_2^d$  such that it belongs to only one of  $S, S'$ . Without loss of generality let that subspace be  $S$ . Then  $\text{Sl}_d(\mathbf{A}(v), B) = 1$  as  $v \in S$  where as  $\text{Sl}_d(\mathbf{A}(v), B') = 0$  as  $v \notin S'$ . Hence the number of different restrictions is at least number of different subspaces of  $\mathbb{F}_2^d$  which is  $2^{\Omega(d^2)}$ . Hence the proof.  $\blacktriangleleft$

This completes the proof of Theorem 1.3 from the introduction. This implies that for  $\text{Sl}_d$ , the branching program size lower bound is  $\Omega\left(\frac{d^2}{\log d} \times d\right) = \Omega\left(\frac{d^3}{\log d}\right) = \Omega\left(\frac{n^{1.5}}{\log n}\right)$  where  $n = 2d^2$  is the number of input bits of  $\text{Sl}_d$ .

## 6 Standard Variants of Projective Dimension

In this section, we study two stringent variants of projective dimension for which exponential lower bounds and exact characterizations can be derived. Although these measure do not correspond to restrictions on branching programs, they illuminate essential nature of the general measure. We define the measures and show their characterizations in terms of well-studied graph theoretic parameters.

► **Definition 6.1** (Standard Projective Dimension). A Boolean function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  with the corresponding bipartite graph  $G(U, V, E)$  is said to have standard projective dimension (denoted by  $\text{spd}(f)$ )  $d$  over field  $\mathbb{F}$ , if  $d$  is the smallest possible dimension for which there exists a vector space  $K$  of dimension  $d$  over  $\mathbb{F}$  with a map  $\phi$  assigning subspaces of  $K$  to  $U \cup V$  such that

- for all  $(u, v) \in U \times V$ ,  $\phi(u) \cap \phi(v) \neq \{0\}$  if and only if  $(u, v) \in E$ .
- $u \in U \cup V$ ,  $\phi(u)$  is spanned by a subset of standard basis vectors in  $K$ .

In addition to the above constraints, if the assignment satisfies the property that for all  $(u, v) \in U \times V$ ,  $\dim(\phi(u) \cap \phi(v)) \leq 1$ , we say that the *standard projective dimension is with intersection dimension 1*, denoted by  $\text{uspd}(f)$ .

For  $N \times N$  bipartite graph  $G$  with  $m$  edges, consider the assignment of standard basis vectors to each of the edges and for any  $u \in U \cup V$ ,  $\phi(u)$  is the span of the basis vectors assigned to the edges incident on  $u$ . Moreover, the intersection dimension in this case is 1. Hence for any  $G$ ,  $\text{spd}(G) \leq \text{uspd}(G) \leq m$ . We show that  $bc(G_f) = \text{spd}(G_f)$  and  $\text{uspd}(G_f) = bp(G_f)$ . We refer the reader to the full version [7] for the details of the proof.

Even though  $\text{pd}(G) \leq \text{spd}(G)$ , there are graphs for which the gap is exponential. For example, consider the bipartite realization  $G$  of  $\text{EQ}_n$  with  $N = 2^n$ . We know  $\text{pd}(G) = \theta(\log N)$  but  $\text{spd}(G) \geq N$  since each of the vertices associated with the matched edges cannot share any basis vector with vertices in other matched edges. Hence dimension must be at least  $N$ . We show that standard projective dimension of bipartite  $G$  is equal to biclique cover number.

## 7 Discussion & Conclusion

In this paper we studied variants of projective dimension of graphs with improved connection to branching programs. We showed lower bounds for these measures indicating the weakness and of each of the variants.

An immediate question that arises from our work is whether  $\Omega(d^2)$  lower bound on  $\text{upd}(\mathcal{P}_d)$  is tight. In this direction, since we have established a gap between  $\text{upd}(\mathcal{P}_d)$  and  $\text{pd}(\mathcal{P}_d)$ , it is natural to study how  $\text{pd}$  and  $\text{upd}$  behave under composition of functions, in order to amplify this gap.

The subspace counting based lower bounds for  $\text{bitpdim}$  that we proved are tight for functions like  $\text{ED}_n$ . However, observe that under standard complexity theoretic assumptions

the bitpdim assignment for  $\mathcal{P}_d$  is not tight. Hence it might be possible to use the specific linear algebraic properties of  $\mathcal{P}_d$  to improve the bitpdim lower bound we obtained for  $\mathcal{P}_d$ .

**Acknowledgements.** The authors would like to thank the anonymous reviewers for several suggestions which improved the readability of the paper and specifically for pointing out that the proof of Proposition 5.1 follows from Remark 1.3 in [6].

---

## References

- 1 Paul Beame, Nathan Grosshans, Pierre McKenzie, and Luc Segoufin. Nondeterminism and an abstract formulation of Nečiporuk’s lower bound method. *CoRR*, abs/1608.01932, 2016. URL: <http://arxiv.org/abs/1608.01932>.
- 2 Béla Bollobás. *Random Graphs, Second edition*. Cambridge Studies in Advanced Mathematics 73. Cambridge University Press, 2001.
- 3 Philippe Delsarte. Association schemes and  $t$ -designs in regular semilattices. *Journal of Combinatorial Theory, Series A*, 20(2):230–243, mar 1976. doi:10.1016/0097-3165(76)90017-0.
- 4 Péter Frankl and Ronald L. Graham. Intersection theorems for vector spaces. *European Journal of Combinatorics*, 6(2):183–187, jun 1985. doi:10.1016/s0195-6698(85)80009-3.
- 5 Péter Frankl and Richard M. Wilson. The Erdős-Ko-Rado theorem for vector spaces. *Journal of Combinatorial Theory Series A*, 43(2):228–236, nov 1986. doi:10.1016/0097-3165(86)90063-4.
- 6 Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 27 of *Series: Algorithms and Combinatorics*. Springer New York Inc., 2012.
- 7 Dinesh Krishnamoorthy, Sajin Koroth, and Jayalal Sarma. Characterization and lower bounds for branching program size using projective dimension. *CoRR*, abs/1604.07200, 2016. URL: <http://arxiv.org/abs/1604.07200>.
- 8 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, New York, NY, USA, 1997.
- 9 Satyanarayana V. Lokam. Complexity lower bounds using linear algebra. *Foundations and Trends in Theoretical Computer Science*, 4(1&2):1–155, January 2009. doi:10.1561/0400000011.
- 10 Benjian Lv and Kaishun Wang. The eigenvalues of  $q$ -Kneser graphs. *Discrete Mathematics*, 312(6):1144–1147, 2012. doi:10.1016/j.disc.2011.11.042.
- 11 E. I. Nečiporuk. On a boolean function. *Doklady of the Academy of Sciences of the USSR*, 164(4):765–766, 1966.
- 12 P. Pudlák and V. Rödl. A combinatorial approach to complexity. *Combinatorica*, 12:221–226, 1992. doi:10.1007/BF01204724.
- 13 P. Pudlák and V. Rödl. Some combinatorial-algebraic problems from complexity theory. *Discrete Mathematics*, 136(1-3):253–279, dec 1994. doi:10.1016/0012-365x(94)00115-y.
- 14 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):17:1–17:24, September 2008.
- 15 Lajos Rónyai, László Babai, and Murali K. Ganapathy. On the number of zero-patterns of a sequence of polynomials. *Journal of the AMS*, 14:2001, 2002.
- 16 Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer New York Inc., 1999.



# Finer Separations Between Shallow Arithmetic Circuits

Mrinal Kumar<sup>\*1</sup> and Ramprasad Saptharishi<sup>†2</sup>

1 Rutgers University, New Brunswick, NJ, USA

mrinal.kumar@rutgers.edu

2 Tel Aviv University, Israel

ramprasad@cmi.ac.in

---

## Abstract

In this paper, we show that there is a family of polynomials  $\{P_n\}$ , where  $P_n$  is a polynomial in  $n$  variables of degree at most  $d = O(\log^2 n)$ , such that

- $P_n$  can be computed by linear sized homogeneous depth-5 circuits.
- $P_n$  can be computed by  $\text{poly}(n)$  sized non-homogeneous depth-3 circuits.
- Any homogeneous depth-4 circuit computing  $P_n$  must have size at least  $n^{\Omega(\sqrt{d})}$ .

This shows that the parameters for the depth reduction results of [1, 11, 20] are tight for extremely restricted classes of arithmetic circuits, for instance homogeneous depth-5 circuits and non-homogeneous depth-3 circuits, and over an appropriate range of parameters, qualitatively improve a result of Kumar and Saraf [14], which showed that the parameters of depth reductions are optimal for algebraic branching programs.

**1998 ACM Subject Classification** I.1.1 Expressions and Their Representation

**Keywords and phrases** arithmetic circuits, lower bounds, separations, depth reduction

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.38

## 1 Introduction

An arithmetic circuit over a field  $\mathbb{F}$  and variables  $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$  is a directed acyclic graph with nodes labelled by  $+$  and  $\times$  operations over  $\mathbb{F}$  and leaves (nodes of in-degree 0) labelled by elements of  $\mathbb{F}$  and  $\mathbf{x}$ . The circuit computes an  $n$  variate polynomial in  $\mathbb{F}[\mathbf{x}]$  in the natural way. Arithmetic circuits are natural and intuitive models of computation in the algebraic setting as they allow us to represent multivariate polynomials succinctly. For an introduction to the area of arithmetic circuit complexity, we refer the interested reader to the excellent survey of Shpilka and Yehudayoff [19].

### Bounded depth arithmetic circuits

Most of the recent research in the area of arithmetic circuit complexity is centered around the question of proving strong lower bounds for structured bounded depth arithmetic circuits, in particular homogeneous depth-4 arithmetic circuits [5, 9, 14]. The focus on such circuits is due to a result of Agrawal and Vinay [1] and subsequent strengthening by Koiran [11] and Tavenas [20], which show that strong enough lower bounds for such structured bounded

---

\* Research supported in part by Simons Graduate Fellowship.

† The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 257575



© Mrinal Kumar and Ramprasad Saptharishi;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 38; pp. 38:1–38:12



Leibniz International Proceedings in Informatics

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



depth circuits suffice for general arithmetic circuit lower bounds. For an outline of most of the recent results related to lower bounds for homogeneous depth-4 arithmetic circuits, we refer the reader to a survey of Saptharishi [17].

These set of structural results, collectively referred to as *depth reductions*, show that any homogeneous polynomial in  $n$  variables of degree  $d = \text{poly}(n)$  which can be computed by an arithmetic circuit of size  $\text{poly}(n)$  can also be computed by a homogeneous depth-4 arithmetic circuit of size  $n^{O(\sqrt{d})}$ . A natural question here is to try and understand if the parameters in the above result are asymptotically tight. This direction has previously been explored, and Kumar and Saraf [14] showed a lower bound of  $n^{\Omega(\sqrt{d})}$  for a polynomial that has a  $\text{poly}(n)$ -sized arithmetic circuit. This implies that, in general, the size bound of  $n^{O(\sqrt{d})}$  can not be improved to  $n^{o(\sqrt{d})}$  for  $\text{poly}(n)$ -sized arithmetic circuits. However, as far as we know, it was not known if such improved depth reductions are conceivable for slightly restricted classes of arithmetic circuits, for instance, arithmetic formulas or constant depth arithmetic circuits. In this paper, we study this problem and show that at least for the case when  $d = O(\log^2 n)$ , one cannot hope to prove such improved depth reduction results, for even extremely restricted classes of arithmetic circuits such as linear size homogeneous depth-5 arithmetic circuits, or polynomial sized non-homogeneous depth-3 arithmetic circuits.

We now state our results, and elaborate on how they compare to the known results.

## 1.1 Our results

We prove the following theorems.

► **Theorem 1.1.** *Let  $\mathbb{F}$  be any field. There is a family of polynomials  $\{P_n\}$  over  $\mathbb{F}$ , where  $P_n$  is of degree  $d = O(\log^2 n)$  on  $n$  variables such that  $P_n$  can be computed by a homogeneous depth-5 circuit of size  $O(n)$  whereas any homogeneous depth-4 circuit computing  $P_n$  requires size  $n^{\Omega(\sqrt{d})}$ .*

► **Theorem 1.2.** *Let  $\mathbb{F}$  be any field of characteristic zero. There is a family of polynomials  $\{P_n\}$  over  $\mathbb{F}$ , where  $P_n$  is of degree  $d = O(\log^2 n)$  on  $n$  variables such that  $P_n$  can be computed by a (non-homogeneous) depth-3 circuit of size  $\text{poly}(n)$  whereas any homogeneous depth-4 circuit computing  $P_n$  requires size  $n^{\Omega(\sqrt{d})}$ .*

## 1.2 Comparison to earlier results

An  $n^{\Omega(\sqrt{d})}$  lower bound for homogeneous depth-4 circuits was proved for an explicit polynomial of degree  $d$  in  $n$  variables in VNP by Kayal, Limaye, Saha and Srinivasan [9] and for the iterated matrix product (IMM) by Kumar and Saraf [14]. Improvements on this can happen on three fronts – (1) by improving the bound from  $n^{\Omega(\sqrt{d})}$  to  $n^{\omega(\sqrt{d})}$ , or (2) by making the lower bound work for a class more general than homogeneous depth-4 circuits, or (3) by proving the lower bound for a polynomial “simpler” than IMM. This work is of the last category where the polynomial is computed by linear sized homogeneous depth-5 circuits or polynomial sized depth-3 circuits.

We elaborate more on this now.

### Depth reduction to depth-4 as a springboard for stronger lower bounds

Let  $\mathcal{C}$  be a class of arithmetic circuits. If we had a depth reduction result that showed that all homogeneous polynomials of degree  $d$  in  $n$  variables that can be computed by an arithmetic circuit  $C \in \mathcal{C}$  of size  $s(n)$  can also be computed by a homogeneous depth-4 arithmetic circuit

of size  $s^{o(\sqrt{d})}$ , then it follows from the results in [9, 14] that there is an explicit polynomial in VP (or VNP) that cannot be computed by polynomial size arithmetic circuits in  $\mathcal{C}$ . In this sense, the *efficient* reductions to homogeneous depth-4 circuits is a *springboard* to prove lower bounds for many potentially stronger classes of circuits.

The lower bound for IMM in [14] rules out this strategy when  $\mathcal{C}$  is the class of algebraic branching programs, since it shows polynomial families (namely IMM) that have linear size ABPs but require homogeneous depth-4 circuits of size  $n^{\Omega(\sqrt{d})}$ . However the strategy could still, in principle, work for other interesting classes of arithmetic circuits such as arithmetic formulas, constant depth arithmetic circuits or, possibly the simplest of them all, the class of homogeneous depth-5 arithmetic circuits. Another simple class of circuits for which this strategy could be tried is the class of non-homogeneous depth-3 circuits, where superpolynomial lower bounds are not known when the size of the underlying field is large. Theorem 1.1 and Theorem 1.2 show that the above mentioned classes of arithmetic circuits cannot be reduced to homogeneous depth-4 arithmetic circuits of size  $n^{o(\sqrt{d})}$ , albeit for an appropriate range of parameters. So, even though quantitatively we do not prove improved lower bounds, qualitatively, we show near optimal separations between complexity classes which are much closer to each other than was earlier known. Unfortunately, we are only able to show such separations when the degree  $d = O(\log^2 n)$ .

### Non-homogeneous depth-3 circuits

Theorem 1.2 shows a separation between non-homogeneous depth-3 circuits and homogeneous depth-4 circuits, in a low degree regime. Intuitively, to prove such a separation, we need a candidate family of hard polynomials which have polynomial sized non-homogeneous depth-3 circuits and are believed to require homogeneous depth-4 circuits of size  $n^{\Omega(\sqrt{d})}$ . At first glance, it seems unclear what this polynomial should be. The elementary symmetric polynomial of degree  $d$  is *not* a good candidate<sup>1</sup> as it can indeed be computed by a homogeneous depth four circuit of size  $2^{O(\sqrt{d})}$  [7]. However, a *generic affine projection* of the elementary symmetric polynomial, as studied by Shpilka [18], is a natural candidate and is almost complete for this model.

In this paper, however, we do not directly work with this polynomial but it can be easily inferred that the lower bound applies to a generic affine projection of the elementary symmetric polynomial as well.

### Depth hierarchy theorems for arithmetic circuits

Depth hierarchy theorems, which show an exponential, (and near optimal) separation between depth  $h$  and depth  $h + 1$  circuits [6, 16] constitute some of the most celebrated results in the theory of lower bounds for bounded depth boolean circuits. It is natural to ask if such separations can be shown for arithmetic circuits. Unfortunately, superpolynomial lower bounds are not known in general when the depth of the arithmetic circuits is more than four<sup>2</sup>. So, at this point, we can only hope to show such separations between homogeneous depth-5 and homogeneous depth-4 arithmetic circuits. Due to the depth reduction results, the best such separation one can hope to prove for an  $n$  variate degree  $d$  polynomial would be  $n^{\Omega(\sqrt{d})}$ . We prove a matching lower bound, as long as the degree  $d$  is at most  $O(\log^2 n)$ . In the arithmetic circuit literature, the question of depth hierarchy theorems has previously been studied by Raz and Yehudayoff [15], where they show superpolynomial separation

<sup>1</sup> Indeed, for low enough degrees, they are known to have fairly high shifted partials complexity [3].

<sup>2</sup> For homogeneous depth-5 circuits, such lower bounds are known only over small finite fields.

separation between multilinear circuits of product depth  $d$  and product depth  $d + 1$ , for  $d = O(1)$ . In the non-multilinear world, to the best of our knowledge this is the first such attempt. Even in the context of constant depth multilinear circuits, the separation in [15] is between depth-4 and depth-6 circuits, and not between depth-4 and depth-5 circuits.

### The complexity measure

The proof of Kayal et al. [9] and Kumar and Saraf [14] rely on the notion of projected shifted partials of a polynomial as a measure of its complexity. This measure can be thought of as a variant of shifted partials which tries to take advantage of the fact that the hard polynomial is multilinear. The measure in this paper takes advantage of *set-multilinearity* instead of just multilinearity, and such a variant was essentially used in [10], where they showed an  $n^{O(\log n)}$  lower bound for iterated matrix multiplication and the determinant. Our proofs rely on a slightly different interpretation of the measure, which makes the proofs much more transparent. Intuitively, this measure tries to take advantage of the fact that the hard polynomial (Nisan-Wigderson design polynomials or the IMM) is not just multilinear, but in fact set-multilinear. In the regime where  $d \ll n$ , set multilinearity is a much more rigid restriction on a polynomial when compared to multilinearity, and in some sense our gain comes from this observation. Our hard polynomial for Theorem 1.1 is also a simple generic balanced depth-5 circuit.

One might wonder if the results in this paper could have been shown by using the dimension of the projected shifted partial derivatives as the complexity measure. In particular, can we show that the projected shifted partials complexity of a generic depth-5 circuit is sufficiently close to the largest possible value? This would suffice for Theorem 1.1. Although we do not have enough evidence to conjecture one way or the other, intuitively this problem seems tricky since so far the known analyses of the projected shifted partials of a polynomial seems to rely on pairwise distance between the monomials of the hard polynomial, either in the worst case (Nisan-Wigderson polynomial [9, 14]), or in the average case (IMM [14]). Clearly, the monomials in a generic depth-5 circuit do not have good distance in the worst case, and to the best of our understanding, the guarantees about distance in the average case seem a bit weaker than what would suffice to simulate the proof in [14] for a generic depth-5 circuit. However, this problem of proving lower bounds on the dimension of projected shifted partials of homogeneous depth-5 circuits is of independent interest, since even if the answer is negative and homogeneous depth-5 circuits do not have large enough projected shifted partials complexity, then we could use this as a measure to prove lower bounds for such circuits. So far, such lower bounds are only known over small finite fields [12].

## 2 Preliminaries

### 2.1 Notations

- Throughout the paper, we use bold-face letters such as  $\mathbf{x}$  to denote a sets of variables. Most of the times, the size of this set would be clear from context. We use  $\mathbf{x}^e$  to refer to the monomial  $x_1^{e_1} \cdots x_n^{e_n}$ .

- We use the short-hand  $\partial_{\mathbf{x}^e}(P)$  to denote

$$\frac{\partial^{e_1}}{\partial x_1^{e_1}} \left( \frac{\partial^{e_2}}{\partial x_2^{e_2}} (\cdots (P) \cdots) \right).$$

- For a set of polynomials  $\mathcal{P}$  use  $\partial^{=k}\mathcal{P}$  to denote the set of all  $k$ -th order partial derivatives of polynomials in  $\mathcal{P}$ , and  $\partial^{\leq k}\mathcal{P}$  similarly.

Also,  $\mathbf{x}^{\leq \ell} \mathcal{P}$  refer to the set of polynomials of the form  $\mathbf{x}^e \cdot P$  where  $\text{Deg}(\mathbf{x}^e) = \ell$  and  $P \in \mathcal{P}$ . Similarly  $\mathbf{x}^{\leq \ell} \mathcal{P}$ .

- For an integer  $m > 0$ , we use  $[m]$  to denote the set  $\{1, \dots, m\}$ .
- For a set of vectors (or polynomials)  $V$ , their span over  $\mathbb{F}$  will be denoted by  $\text{Span}(V)$  and their dimension by  $\text{Dim}(V)$ .
- For a subset  $\mathbf{y}$  of variables and a polynomial  $P \in \mathbb{F}[\mathbf{x}, \mathbf{y}]$ , by  $\text{Mult}_{\mathbf{y}}[P]$ , we denote the polynomial  $P' \in \mathbb{F}[\mathbf{x}, \mathbf{y}]$  which is obtained by projecting  $P$  only to its monomials which are multilinear in  $\mathbf{y}$ .

Similarly, for a set  $S$  of polynomials,  $\text{Mult}_{\mathbf{y}}[S]$  denotes the set of polynomials obtained by projecting every polynomial in  $S$  to the monomials which are multilinear in  $\mathbf{y}$ .

## 2.2 The hard polynomial

The hard function for the lower bounds will be a generic balanced  $\Pi\Sigma\Pi\Sigma$  circuit with appropriate parameters. We define the polynomial  $P_{m,d}$  as

$$P_{m,d} = \prod_{i=1}^{\sqrt{d}} \sum_{j=1}^m \prod_{i'=1}^{\sqrt{d}} \sum_{j'=1}^m x_{ijj'j'}.$$

The polynomial  $P_{m,d}$  depends on  $m^2 d$  variables. It would be useful to have  $L_{ijj'} = \sum_{j'} x_{ijj'j'}$  so that  $P_{m,d} = \prod_i \sum_j \prod_{i'} L_{ijj'}$ .

Observe that the polynomial  $P_{m,d}$  is a set multilinear polynomial for the partition of variables into  $\{\mathbf{x}_{i*i'*} : i, i' \in [\sqrt{d}]\}$ , where  $\mathbf{x}_{i*i'*} = \{x_{ijj'j'} : j, j' \in [m]\}$ . There are  $d$  such sets and each is of size  $m^2$ .

The range of parameters we will be working with in this paper when  $d = \delta \log^2 n$  for a small enough constant  $\delta$ . For such small  $d$ , it follows from observations in [4] that the polynomial  $P_{m,d}$  is computable by a polynomial sized non-homogeneous depth-3 circuit. More formally, the proof relies on the following lemma which is implicit in [4].

► **Lemma 2.1** ([4]). *Let  $C$  be a homogeneous  $\Sigma\Pi^{[a]}\Sigma\Pi^{[b]}\Sigma$  circuit of size  $s$  over  $\mathbb{C}$ , the field of complex numbers, which computes an  $n$ -variate polynomial  $P$ . Then there is an equivalent  $\Sigma\Pi\Sigma$  circuit  $C'$  of size  $s' = \text{poly}(2^a, 2^b, n, s)$  which computes  $P$ .*

Using this observation, we have the following lemma which shows that there is a small depth-3 circuit for  $P_{m,d}$ .

► **Lemma 2.2.** *Let  $P$  be an  $n$  variate polynomial of degree  $d = O(\log^2 n)$  which is computed by a homogeneous  $\Sigma\Pi^{[\sqrt{d}]\Sigma\Pi^{[\sqrt{d}]\Sigma}$  circuit  $C$  of size  $s$ . Then,  $P$  is computable by a  $\Sigma\Pi\Sigma$  circuit of size  $\text{poly}(n)$ .* ◀

Thus, to prove Theorem 1.1 and Theorem 1.2, it suffices to show an  $n^{\Omega(\sqrt{d})}$  lower bound on the size of homogeneous  $\Sigma\Pi\Sigma\Pi$  arithmetic circuits computing  $P_{m,d}$ .

## 2.3 Some useful approximations

► **Lemma 2.3** ([5]). *Let  $n, a, b$  satisfy  $a + b = o(n)$ . Then,*

$$\frac{(n+a)!}{(n-b)!} = n^{a+b} \cdot \exp(O((a+b)^2/n)).$$

*In particular, if  $a + b = o(\sqrt{n})$ , then the right hand side is  $(1 + o(1)) \cdot n^{a+b}$ .*

► **Lemma 2.4.** *For all  $x, y > 0$ ,*

$$e^{xy} \geq (1+x)^y \geq e^{\frac{xy}{x+1}}.$$

### 3 Proof of Theorem 1.1

The first step in previous lower bounds for homogeneous depth-4 circuits is using a random restriction to set each variable independently to zero with a certain probability. We shall first analyze the random restriction process on a homogeneous depth-4 circuit and also on the polynomial  $P_{m,d}$ .

#### 3.1 The effect of a random restriction

Our restrictions  $\mathcal{R}_p$  will be defined by setting every variable to zero with a probability  $1 - p$  and keeping it alive with a probability  $p$ .

► **Lemma 3.1.** *Let  $\epsilon > 0$  be any fixed constant and let  $p = \frac{1}{n^\epsilon}$ . Let  $C$  be a  $\Sigma\Pi\Sigma\Pi$  circuit of size  $n^{\frac{\epsilon^2}{2}\sqrt{d}}$ . Then with a probability at least  $1 - o(1)$  over  $\pi \leftarrow \mathcal{R}_p$ , every product gate at the lowest level of  $C$  (closest to the leaves) that depends on more than  $\epsilon\sqrt{d}$  distinct variables is set to zero in  $\pi(C)$ .*

**Proof.** Consider any product gate of support at least  $\epsilon\sqrt{d}$  present at the bottom level of  $C$ . The probability that this gate is not set to zero in  $\pi(C)$  is at most  $\frac{1}{n^{\epsilon^2\sqrt{d}}}$ . So, by a union bound over all the product gates in  $C$ , the probability that some gate of support at least  $\epsilon\sqrt{d}$  survives in  $\pi(C)$  is at most  $n^{\frac{\epsilon^2}{2}\sqrt{d}} \cdot \frac{1}{n^{\epsilon^2\sqrt{d}}}$  which is  $o(1)$ . ◀

We now analyse the effect of random restrictions on our candidate hard function.

► **Lemma 3.2.** *Let  $\epsilon$  be a small enough constant and let  $p = \frac{1}{n^\epsilon}$ , and let  $P_{m,d}$  be the polynomial as defined in subsection 2.2. Then, with probability at least  $1 - o(1)$  over  $\pi \leftarrow \mathcal{R}_p$ , the polynomial  $\pi(P_{m,d})$  is of the form*

$$\pi(P_{m,d}) = \prod_{i=1}^{\sqrt{d}} \sum_{j=1}^m \prod_{i'=1}^{\sqrt{d}} L'_{iji'}$$

where each  $L'_{iji'}$  is a non-zero linear form.

**Proof.** From our choice of parameters, observe that  $n = m^2d$ , and since  $d = O(\log^2 n)$ ,  $m > n^{1/4}$ . Now, for any fixed linear form  $L_{iji'}$ , the probability that  $\pi(L_{iji'})$  equals zero is equal to  $(1 - p)^m = (1 - 1/n^\epsilon)^m$  which is less than  $(1 - 1/n^\epsilon)^{n^{2\epsilon}} = \frac{1}{\omega(n)}$ . Therefore, the probability that there exists a linear form  $L_{iji'}$  such that  $\pi(L_{iji'}) \equiv 0$  is  $o(1)$ , and the lemma follows. ◀

At this point, we will deterministically set all but one alive variable in each  $L'_{iji'}$  in the above lemma to zero, and obtain the following corollary upto a relabelling of variables.

► **Corollary 3.3.** *Let  $\epsilon$  be a fixed constant and  $p = \frac{1}{n^\epsilon}$ , and let  $P_{m,d}$  be the polynomial as defined in subsection 2.2. Then, with probability at least  $1 - o(1)$  over  $\pi \leftarrow \mathcal{R}_p$ , there is a 0,1 projection of  $\pi(P_{m,d})$  which is of the form*

$$P'_{m,d} = \prod_{i=1}^{\sqrt{d}} \sum_{j=1}^m \prod_{i'=1}^{\sqrt{d}} x_{iji'}$$

where each  $x_{iji'}$  is a distinct variable.

Observe that Theorem 3.1 continues to hold under this additional deterministic restriction, as the bottom support of a depth-4 circuit does not increase under  $0,1$  projections. Clearly  $P'_{m,d}$  is computable by a homogeneous depth-4 circuit of bottom fan-in  $\sqrt{d}$ .

In order to complete the proof, it suffices to show that any homogeneous depth-4 circuit of *bottom support* bounded by  $\sqrt{d}/10$  that computes  $P'_{m,d}$  must have size  $n^{\Omega(\sqrt{d})}$ . In fact, Kumar and Saraf [13] have shown that any homogeneous depth-4 circuit of bottom fan-in at most  $\sqrt{d}/10$  computing  $P'_{m,d}$  must require size  $n^{\Omega(\sqrt{d})}$  using the measure of dimension of shifted partial derivatives. Thus we need to find a way to lift this lower bound to the class of homogeneous depth-4 circuit of *bottom support* bounded by  $\sqrt{d}/10$ . To do this, we modify the measure of dimension of shifted partials in order to address small bottom support instead of small bottom fan-in.

### 3.2 The complexity measure

The measure is again the dimension of an appropriate linear space of polynomials.

► **Definition 3.4** (The complexity measure). Let  $\mathbf{x} = \mathbf{x}_1 \sqcup \dots \sqcup \mathbf{x}_d$  be a partition of the variables into  $d$  sets. For any polynomial  $P \in \mathbb{F}[\mathbf{x}]$ , define  $P' \in \mathbb{F}[\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_d, y_1, y_2, \dots, y_d]$  be the polynomial derived from  $P$  by replacing every occurrence of the variable  $x_{ij} \in \mathbf{x}_i$  by  $y_i \cdot x_{ij}$ . Then, the complexity measure

$$\Gamma_{k,\ell}(P) := \text{Dim}_{\mathbb{F}} \{ (\mathbf{x}^{-\ell} \cdot \text{Mult}_{\mathbf{y}}[\partial^{=k}(P')]) \}.$$

We remark that all the derivatives and shifts in the definition of  $\Gamma_{k,\ell}$  are taken with respect to the variables in  $\mathbf{x}$ . However, the multilinearization is done with respect to the  $\mathbf{y}$  variables. As mentioned earlier, this measure was used in [10] where it was called *dimension of shifted projected partial derivatives*.

As is clear from the definition, the measure is subadditive, i.e for every pair of polynomials  $P$  and  $Q$  and for every pair of field constants  $\alpha$  and  $\beta$ , the inequality  $\Gamma_{k,\ell}(\alpha P + \beta Q) \leq \Gamma_{k,\ell}(P) + \Gamma_{k,\ell}(Q)$  holds for every choice of  $k$  and  $\ell$ .

Throughout this paper, we will be using very simple connections between the measure  $\Gamma_{k,\ell}$  and the well known notion of shifted partial derivatives of polynomials (first defined in [8]), defined as

► **Definition 3.5** (Shifted partial derivatives). Define  $P \in \mathbb{F}[\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_d]$  be a polynomial. Then, the *dimension of shifted partial derivatives* is defined as

$$\text{Dim}_{\mathbb{F}} \{ (\mathbf{x}^{-\ell} \cdot \partial^{=k}(P)) \}.$$

Observe that if a polynomial  $P$  is set-multilinear with respect to the partition of the variables in  $\mathbf{x}$  into  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_d$ , then multilinearization with respect to the  $\mathbf{y}$  variables does not kill any of the monomials in the partial derivatives. In particular, for a set multilinear polynomial  $P$ , and for every choice of  $k, \ell$ , the quantity  $\Gamma_{k,\ell}(P)$  is exactly equal to the dimension of shifted partial derivatives of the polynomial  $P$  where we take derivatives of order  $k$  and shifts are of degree  $\ell$ . This observation will be useful for us in the proof and is summarised below.

► **Observation 3.6.** Let  $P$  be a set multilinear polynomial of degree  $d$ . Then for every choice of parameters  $k$  and  $\ell$ ,

$$\Gamma_{k,\ell}(P) = \text{Dim} (\mathbf{x}^{-\ell} \cdot \partial^{=k}(P)).$$

Since  $P_{m,d}$  is set multilinear with respect to the partition

$$\mathbf{x} = \bigsqcup_{i,i' \leq \sqrt{d}} \mathbf{x}_{i**i'}$$

we use this partition for in the definition of  $\Gamma_{k,\ell}$ . To complete the proof, we use this measure to show that  $P'_{m,d}$  cannot be computed by small homogeneous depth-4 circuit of bottom support bounded by  $\sqrt{d}/10$ .

### 3.3 Upper bound for a small bottom-support depth-4 circuit

► **Lemma 3.7.** *Let  $C$  be a homogeneous  $\Sigma\Pi\Sigma\Pi$  circuit with bottom support at most  $s$  which computes a degree  $d$  polynomial in  $\mathbb{F}[\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_d]$ . Then, for every  $k$  and  $\ell$ ,*

$$\Gamma_{k,\ell}(C) \leq \text{Size}(C) \cdot 2^{2d} \cdot \binom{d}{k} \cdot \binom{n + \ell + ks}{n}.$$

**Proof.** Since the measure  $\Gamma_{k,\ell}$  is subadditive, we will prove an upper bound on  $\Gamma_{k,\ell}$  for one product term in  $C$ . So, let  $T = Q_1 \cdot Q_2 \cdots Q_t$ , where each  $Q_i$  has support at most  $s$ . Without loss of generality, we can assume that  $t \leq d$  since the circuit  $C$  is homogeneous to start with.

Recall that in the first step, we replace every variable  $x_{ij}$  by  $y_i \cdot x_{ij}$ . This transforms  $T = Q_1 \cdots Q_t$  into  $T = Q'_1 \cdot Q'_2 \cdots Q'_t$ . Every monomial  $\mathbf{x}^\alpha$  in the  $\mathbf{x}$  variables will be transformed to a monomial  $\mathbf{y}^{\alpha'} \cdot \mathbf{x}^\alpha$  by this transformation. The key points are that  $\mathbf{y}^{\alpha'}$  is only over  $d$  variables, and if  $\mathbf{x}^\alpha$  is non-multilinear then so is  $\mathbf{y}^{\alpha'}$ .

Let us now consider the derivative of  $T$  with respect to a monomial  $\mathbf{x}^\alpha$  of order  $k$ .

$$\partial_{\mathbf{x}^\alpha}(T') \in \text{Span} \left\{ \partial_{\mathbf{x}^\alpha}(Q'_A) \cdot Q'_{\bar{A}} : A \subseteq [t], |A| \leq k \right\},$$

where  $Q'_A$  is a shorthand for  $\prod_{i \in A} Q'_i$ .

$$\text{Mult}_{\mathbf{y}}[\partial_{\mathbf{x}^\alpha}(T')] \in \text{Span} \left\{ \text{Mult}_{\mathbf{y}} \left[ \partial_{\mathbf{x}^\alpha}(Q'_A) \cdot Q'_{\bar{A}} \right] : A \subseteq [t], |A| \leq k \right\}.$$

Since we are interested in the multilinear component, it suffices to only focus on multilinear (in  $\mathbf{y}$ ) monomials in both  $\partial_{\mathbf{x}^\alpha}(Q'_A)$  and  $Q'_{\bar{A}}$ . Since  $Q'_A$  is a product of at most  $k$  polynomials, each of support-size bounded by  $s$ , the only monomials  $\mathbf{x}^\beta$  that can contribute a multilinear  $\mathbf{y}$ -part can have degree at most  $ks$ . Therefore,

$$\begin{aligned} \text{Mult}_{\mathbf{y}}[\partial_{\mathbf{x}^\alpha}(Q'_A)] &\in \text{Span} \left\{ \mathbf{y}^\beta \cdot \mathbf{x}^\gamma : \text{Deg}(\mathbf{x}^\gamma) \leq ks, \mathbf{y}^\beta \text{ multilinear} \right\} \\ \text{Mult}_{\mathbf{y}}Q'_{\bar{A}} &= \sum_{\beta'} \mathbf{y}^{\beta'} \cdot Q'_{\bar{A},\beta'} \\ \implies \text{Mult}_{\mathbf{y}} \left[ \partial_{\mathbf{x}^\alpha}(Q'_A) \cdot Q'_{\bar{A}} \right] &\in \text{Span} \left\{ \mathbf{y}^\beta \mathbf{y}^{\beta'} \cdot \mathbf{x}^\gamma \cdot Q'_{\bar{A},\beta'} : \text{Deg}(\mathbf{x}^\gamma) \leq ks, \mathbf{y}^\beta \mathbf{y}^{\beta'} \text{ multilinear} \right\}. \end{aligned}$$

Taking the union over all shifts and all derivatives, we get

$$\begin{aligned} \mathbf{x}^{\alpha'} \cdot \text{Mult}_{\mathbf{y}}[\partial^k(T')] &\subseteq \text{Span} \left\{ \mathbf{y}^\beta \mathbf{y}^{\beta'} \cdot \mathbf{x}^\gamma \cdot Q'_{\bar{A},\beta'} : A \subseteq [t], |A| \leq k, \right. \\ &\quad \left. \text{degree}(\mathbf{x}^\gamma) \leq \ell + ks, \mathbf{y}^\beta \mathbf{y}^{\beta'} \text{ is multilinear} \right\}. \end{aligned}$$

For any  $k, \ell$ , it follows that

$$\Gamma_{k,\ell}(T') \leq 2^{2d} \cdot \binom{d}{k} \cdot \binom{n + \ell + ks}{n}.$$

Using subadditivity, we obtain the lemma. ◀



### 3.4 Lower bound for the measure on $P'_{m,d}$

The final technical ingredient of our proof will be a lower bound on the dimension of shifted partials of the polynomial  $P'_{m,d}$ . The bound follows from the calculations in [13], but we provide the calculation here for completeness.

► **Lemma 3.8.** *Recall the polynomial*

$$P'_{m,d} = \prod_{i=1}^{\sqrt{d}} \sum_{j=1}^m \prod_{i'=1}^{\sqrt{d}} x_{iji'}$$

where each  $x_{iji'}$  is a distinct variable. For  $k = \sqrt{d}$  and any  $\ell$ , we have

$$\text{Dim}(\mathbf{x}^{\ell} \cdot \partial^k(P)) \geq \frac{1}{4} \cdot \binom{n+\ell}{\ell}^{\frac{1}{2} \cdot (d-\sqrt{d})} \cdot \binom{n+\ell-1}{n}.$$

**Proof.** To show that the shifted partials complexity of  $P$  is large, we will follow the outline in [13]. We consider the following subset  $\mathcal{S}$  of monomials of degree equal to  $k = \sqrt{d}$ :

$$\mathcal{S} = \{x_{1a_11} \cdot x_{2a_21} \cdots x_{ka_k1} : a_1, a_2, \dots, a_k \in [m]\}.$$

Firstly, note that for any monomial  $\mathbf{x}^\alpha = x_{1a_11} \cdots x_{ka_k1} \in \mathcal{S}$ , the derivative  $\partial_{\mathbf{x}^\alpha}(P)$  is just the monomial

$$(x_{1a_12} \cdots x_{1a_1k}) \cdots (x_{ka_k2} \cdots x_{ka_kk}).$$

Thus, it suffices to get a lower bound of distinct monomials obtained as shifts of such derivatives. To assist this calculation, we pick a subset  $\mathcal{S}'$  of the set  $\mathcal{S}$  such that the distance between any two monomials in  $\mathcal{S}'$  is ‘large’, and the size of  $\mathcal{S}'$  is also ‘large’. This can be done by picking the monomials which correspond to a good code of length  $k$  over the alphabet  $\Sigma = \{1, 2, \dots, m\}$ . To this end, we pick a Reed-Solomon code of relative distance  $1/2$  and rate  $1/2$ . This can be done as long as  $m$  is a prime power and  $\sqrt{d} \leq m$ . Let  $\mathcal{S}'$  be a such set of size  $m^{k/2}$  where any pair of monomials in  $\mathcal{S}'$  differ on at least  $\sqrt{d}/2$  locations.

When we take derivatives of  $P$  with respect to monomials in the set  $\mathcal{S}'$ , two monomials obtained from distinct elements of  $\mathcal{S}'$  have distance at least  $\Delta = \sqrt{d}(\sqrt{d}-1)/2 = (d-\sqrt{d})/2$ . So, each of the shifted partial derivatives obtained by shifting the derivatives of  $P$  by monomials of degree  $\ell$  is just a monomial, and a lower bound on the number of distinct monomials obtained in this way gives us a lower bound on  $\text{Dim}(\mathbf{x}^{\ell} \cdot \partial^k(P))$ . In fact, we shall choose an even smaller set  $\mathcal{S}''$  to ensure the following bounds work out.

By the inclusion-exclusion approach of Chillara and Mukhopadhyay [2], for any set  $\mathcal{S}'' \subset \mathcal{S}'$  we get the following:

$$\text{Dim}(\mathbf{x}^{\ell} \cdot \partial^k(P)) \geq |\mathcal{S}''| \cdot \binom{n+\ell-1}{n} - \frac{|\mathcal{S}''|^2}{2} \cdot \binom{n+\ell-\Delta-1}{n}.$$

If we pick our parameters, such that the first term above is at least twice the second term, then we would be done. For this, we need

$$|\mathcal{S}''| \leq \frac{\binom{n+\ell-1}{n}}{\binom{n+\ell-\Delta-1}{n}}.$$

For our choice of parameters,  $\ell, n \gg d^2$ , the ratio  $\frac{\binom{n+\ell-1}{n}}{\binom{n+\ell-\Delta-1}{n}}$  can be approximated by  $\left(\frac{n+\ell}{\ell}\right)^\Delta$  within a factor  $1 \pm o(1)$  by Theorem 2.3. So, it suffices if our choice of parameters satisfies (omitting floors)

$$|\mathcal{S}''| = \frac{1}{2} \cdot \left(\frac{n+\ell}{\ell}\right)^\Delta.$$

Plugging in  $\Delta$  and the size of  $\mathcal{S}''$  in the inclusion-exclusion bound, we get

$$\text{Dim}(\mathbf{x}^{\ell} \cdot \partial^k(P)) \geq \frac{1}{4} \cdot \left(\frac{n+\ell}{\ell}\right)^{(d-\sqrt{d})/2} \cdot \binom{n+\ell-1}{n}. \quad \blacktriangleleft$$

### 3.5 Putting it together

► **Theorem 3.9** (Theorem 1.1 restated). *Let  $C$  be a homogeneous depth-4 arithmetic circuit which computes the polynomial  $P_{m,d}$  for  $d = 0.0001 \log^2 n$ . Then, the size of  $C$  is at least  $\exp(\Omega(\sqrt{d} \log n))$ .*

**Proof.** Assume on the contrary that the polynomial  $P_{m,d}$  can be computed by  $C$ , a homogeneous depth-4 circuit of size at most  $\exp(0.001\sqrt{d} \log n)$ . If we apply a random restriction that sets every variable to zero independently with probability  $1/n^{0.1}$ , by Theorem 3.1 (with  $\epsilon = 0.1$ ), the circuit reduces to  $C'$ , a homogeneous depth-4 circuit with bottom support bounded by  $\sqrt{d}/10$  with probability  $1 - o(1)$ .

On the other hand by Theorem 3.3, the polynomial  $P_{m,d}$  under such a random restriction still retains  $P'_{m,d}$  as a projection with high probability. Fix a restriction that satisfies both these properties and we now have a homogeneous depth-4 circuit  $C''$  with bottom support bounded by  $\sqrt{d}/10$  and size at most  $\exp(0.001\sqrt{d} \log n)$  that computes  $P'_{m,d}$ .

Let  $k = \sqrt{d}$  and  $\ell = \frac{n\sqrt{d}}{\log n}$ . By Theorem 3.7, we have

$$\Gamma_{k,\ell}(C'') \leq \text{Size}(C'') \cdot 2^{2d} \cdot \binom{n+\ell+(0.1)d}{n}.$$

On the other hand, by Theorem 3.8 and Theorem 3.6,

$$\Gamma_{k,\ell}(P'_{m,d}) \geq \frac{1}{4} \cdot \left(\frac{n+\ell}{\ell}\right)^{(d-\sqrt{d})/2} \cdot \binom{n+\ell-1}{n}.$$

Together, this implies that

$$\text{Size}(C'') \geq \frac{1}{4} \cdot \frac{\binom{n+\ell-1}{n} \cdot \left(\frac{n+\ell}{\ell}\right)^{(d-\sqrt{d})/2}}{2^{2d} \cdot \binom{n+\ell+(0.1)d}{n}}.$$

For our regime of parameters,  $\sqrt{d} = 0.01 \log n$  and hence  $2^{2d} = n^{0.02\sqrt{d}} = \exp(0.02\sqrt{d} \log n)$ . Simplifying the ratio of binomial coefficients using (Theorem 2.3), and using  $\frac{d-\sqrt{d}}{2} > \frac{d}{3}$ , we get

$$\begin{aligned} \text{Size}(C'') &\geq \frac{1}{\exp(0.02\sqrt{d} \log n)} \cdot \left(1 + \frac{n}{\ell}\right)^{d/3} \\ &\geq \frac{1}{\exp(0.02\sqrt{d} \log n)} \cdot \exp\left(\frac{(nd/3\ell)}{(n/\ell) + 1}\right) \quad (\text{By Theorem 2.4}) \\ &> \exp\left(0.1\sqrt{d} \log n\right), \end{aligned}$$

which contradicts the assumption on the size of  $C$ . Hence  $\text{Size}(C) \geq \exp(0.001\sqrt{d} \log n)$ . ◀

## 4 Open questions

We end with some open questions.

- One question of great interest to us would be to show the lower bounds in this paper when the degree is larger. The other proofs of lower bounds for homogeneous depth-4 circuits [9, 14] tolerate degrees as high as  $n^{1/2}$ . We conjecture that the results in this paper are true even when the degree  $d$  and the number of variables  $n$  are polynomially related.
- Is the dimension of projected shifted partials of a generic homogeneous depth-5 circuit close to the largest possible value? This could offer one approach to resolving the first open problem.
- If the answer to the second problem above is negative, then we might be able to use projected shifted partials as a complexity measure to prove new lower bounds for homogeneous depth-5 arithmetic circuits. Hence, even proving non-trivial *upper bounds* on the projected shifted partials complexity of homogeneous depth-5 circuits would be very interesting.

**Acknowledgements.** Part of this work was done while the authors were visiting Microsoft Research, Bangalore in Summer 2014. We are grateful to Neeraj Kayal for many insightful discussions.

---

## References

- 1 Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*, pages 67–75, 2008. doi:10.1109/FOCS.2008.32.
- 2 Suryaajith Chillara and Partha Mukhopadhyay. Depth-4 Lower Bounds, Determinantal Complexity : A Unified Approach. *Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, 2014. Preliminary version at [arXiv:1308.1640](https://arxiv.org/abs/1308.1640). doi:10.4230/LIPIcs.STACS.2014.239.
- 3 Hervé Fournier, Nutan Limaye, Meena Mahajan, and Srikanth Srinivasan. *The Shifted Partial Derivative Complexity of Elementary Symmetric Polynomials*, pages 324–335. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-48054-0\_27.
- 4 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic Circuits: A Chasm at Depth Three. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 578–587, 2013. doi:10.1109/FOCS.2013.68.
- 5 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Approaching the chasm at depth four. *Journal of the ACM*, 61(6):33:1–33:16, 2014. Preliminary version in the *28th Annual IEEE Conference on Computational Complexity (CCC 2013)*. doi:10.1145/2629541.
- 6 Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC 1986)*, pages 6–20, 1986. doi:10.1145/12130.12132.
- 7 Pavel Hrubes and Amir Yehudayoff. Homogeneous formulas and symmetric polynomials. *Computational Complexity*, 20(3):559–578, 2011.
- 8 Neeraj Kayal. An exponential lower bound for the sum of powers of bounded degree polynomials. In *Electronic Colloquium on Computational Complexity (ECCC)TR12-081*, 2012. URL: <http://eccc.hpi-web.de/report/2012/081/>.

- 9 Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. An Exponential Lower Bound for Homogeneous Depth Four Arithmetic Circuits. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014)*, 2014. doi:10.1109/FOCS.2014.15.
- 10 Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. Super-polynomial lower bounds for depth-4 homogeneous arithmetic formulas. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, 2014.
- 11 Pascal Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theoretical Computer Science*, 448:56–65, 2012. arXiv:1006.4700, doi:10.1016/j.tcs.2012.03.041.
- 12 Mrinal Kumar and Ramprasad Satharishi. An exponential lower bound for homogeneous depth-5 circuits over finite fields. *Electronic Colloquium on Computational Complexity (ECCC)*, 2015. eccc:TR15-109. URL: <http://eccc.hpi-web.de/report/2015/109/>.
- 13 Mrinal Kumar and Shubhangi Saraf. The limits of depth reduction for arithmetic formulas: it’s all about the top fan-in. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 136–145, 2014. doi:10.1145/2591796.2591827.
- 14 Mrinal Kumar and Shubhangi Saraf. On the power of homogeneous depth 4 arithmetic circuits. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014)*, 2014. doi:10.1109/FOCS.2014.46.
- 15 Ran Raz and Amir Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity*, 18(2):171–207, 2009. Preliminary version in the *23rd Annual IEEE Conference on Computational Complexity (CCC 2008)*. doi:10.1007/s00037-009-0270-8.
- 16 Benjamin Rossman, Rocco A. Servedio, and Li-Yang Tan. An average-case depth hierarchy theorem for boolean circuits. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2015)*, 2015. Preliminary version at arXiv:1504.03398. doi:10.1109/FOCS.2015.67.
- 17 Ramprasad Satharishi. A survey of lower bounds in arithmetic circuit complexity. Github survey, 2015. URL: <https://github.com/dasarpmar/lowerbounds-survey/releases/>.
- 18 Amir Shpilka. Affine projections of symmetric polynomials. *Journal of Computer and System Sciences*, 65(4):639–659, 2002. Preliminary version in the *16th Annual IEEE Conference on Computational Complexity (CCC 2001)*, eccc:TR01-035. doi:10.1016/S0022-0000(02)00021-1.
- 19 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5:207–388, March 2010. doi:10.1561/04000000039.
- 20 Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. *Inf. Comput.*, 240:2–11, 2015. Preliminary version in the *38th International Symposium on the Mathematical Foundations of Computer Science (MFCS 2013)*. doi:10.1016/j.ic.2014.09.004.

# Sum of Products of Read-Once Formulas

Ramya C.<sup>1</sup> and B. V. Raghavendra Rao<sup>2</sup>

1 Dept. of Computer Science and Engineering, IIT Madras, Chennai, India  
ramya@cse.iitm.ac.in

2 Dept. of Computer Science and Engineering, IIT Madras, Chennai, India  
bvrr@cse.iitm.ac.in

---

## Abstract

We study limitations of polynomials computed by depth two circuits built over read-once formulas (ROFs). In particular,

1. We prove an exponential lower bound for the sum of ROFs computing the  $2n$ -variate polynomial in VP defined by Raz and Yehudayoff [CC,2009].
2. We obtain an exponential lower bound on the size of arithmetic circuits computing sum of products of restricted ROFs of unbounded depth computing the permanent of an  $n$  by  $n$  matrix. The restriction is on the number of variables with  $+$  gates as a parent in a proper sub formula of the ROF to be bounded by  $\sqrt{n}$ . Additionally, we restrict the product fan in to be bounded by a sub linear function. This proves an exponential lower bound for a subclass of possibly non-multilinear formulas of unbounded depth computing the permanent polynomial.
3. We also show an exponential lower bound for the above model against a polynomial in VP.
4. Finally we observe that the techniques developed yield an exponential lower bound on the size of sums of products of syntactically multilinear arithmetic circuits computing a product of variable disjoint linear forms where the bottom sum gate and product gates at the second level have fan in bounded by a sub linear function.

Our proof techniques are built on the measure developed by Kumar et. al.[ICALP 2013] and are based on a non-trivial analysis of ROFs under random partitions. Further, our results exhibit strengths and provide more insight into the lower bound techniques introduced by Raz [STOC 2004].

**1998 ACM Subject Classification** F.1.1 Models of Computation, F.2.1 Numerical Algorithms and Problems

**Keywords and phrases** Arithmetic Circuits, Permanent, Computational Complexity, Algebraic Complexity Theory

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.39

## 1 Introduction

More than three decades ago, Valiant [26] developed the theory of Algebraic Complexity classes based on arithmetic circuits as the model of algebraic computation. Valiant considered the permanent polynomial  $perm_n$  defined over an  $n \times n$  matrix  $X = (x_{i,j})_{1 \leq i,j \leq n}$  of variables:

$$perm_n(X) = \sum_{\pi \in S_n} \prod_{i=1}^n x_{i,\pi(i)}$$

where  $S_n$  is the set of all permutations on  $[n]$ .



© Ramya C. and B. V. Raghavendra Rao;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 39; pp. 39:1–39:15



Leibniz International Proceedings in Informatics

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

Valiant [26] showed that the polynomial family  $(perm_n)_{n \geq 0}$  is complete for the complexity class VNP. Further, Valiant [26] conjectured that  $(perm_n)_{n \geq 0}$  does not have polynomial size arithmetic circuits (i.e.  $VP \neq VNP$ ). Since then, obtaining super-polynomial size lower bounds for arithmetic circuits computing  $perm_n$  has been a pivotal problem in Algebraic Complexity Theory. However, for general classes of arithmetic circuits, the best known size bound is an  $\Omega(n \log d)$  lower bound due to Baur and Strassen for an  $n$ -variate degree  $d$  polynomial [2]. In fact, this is the only super linear lower bound we know for general arithmetic circuits. While the challenge of proving lower bounds for general classes of circuits still seem to be afar, naturally the focus has been on proving lower bounds for restricted classes of circuits computing  $perm_n$ .

Recent research has focused on proving lower bounds for low depth circuits. Nisan and Wigderson [17] used partial derivatives to obtain exponential lower bounds against special cases of Depth-3  $\Sigma\Pi\Sigma$  circuits and set multilinear formulas. Later, Grigoriev and Karpinski [6] proved an exponential size lower bound for depth three circuits over finite fields. In 2001, Shpilka and Wigderson [23] proved a quadratic lower bound for  $\Sigma\Pi\Sigma$  circuits over infinite fields computing  $\det_n$  (or  $perm_n$ ) which has been improved recently to an almost cubic lower bound in [11]. Explaining the lack of progress in proving lower bounds even for  $\Sigma\Pi\Sigma$  circuits, Agrawal and Vinay [1] showed that proving exponential lower bounds against depth four arithmetic circuits is enough to resolve Valiant's conjecture. This was improved subsequently in [24, 12]. From then on, depth-4 circuits have been in the limelight. Recently, Gupta et al. [7] obtained  $2^{\Omega(\sqrt{n})}$  top fan-in lower bound for  $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$  circuits computing  $\det_n$  or  $perm_n$ . The techniques introduced in [7, 8] have been generalized and applied to prove lower bounds against several classes of constant depth arithmetic circuits, regular arithmetic formulas and homogeneous arithmetic formulas. (See e.g., [9, 14, 10].)

Apart from constant depth circuits, there has been significant interest in proving lower bounds for unbounded depth circuits with additional structural restrictions such as multilinearity, restricted read etc. A seminal work of Raz [19] showed that multilinear formulas (i.e., every gate in the formula computes a multilinear polynomial) computing  $\det_n$  or  $perm_n$  must have size  $n^{\Omega(\log n)}$ . In [19] Raz used rank of the partial derivative matrix as a complexity measure. Using the same complexity measure as [19], Raz and Yehudayoff [21] proved exponential lower bounds against constant depth multilinear formulas. Subsequently, several generalizations of Raz's measure were introduced. Kumar et al. [13] extended the techniques developed in [19] to prove lower bounds against non-multilinear circuits and formulas of constant depth using the rank of the polynomial coefficient matrix as a measure. (See Definition 8). In [5], Forbes and Shpilka used evaluation dimension of polynomials as a complexity measure to prove exponential lower bounds against Read-Once oblivious algebraic branching programs. Further, in [10] Kayal and Saha used the evaluation dimension to obtain exponential lower bound against depth three multi-k-ic circuits. Over large fields, the evaluation dimension with respect to a partition of the set of variables in a polynomial and rank of the partial derivative matrix with respect to that partition are the same (see Chapter 4 in [4]). However, the evaluation perspective sometimes comes handy in proving lower bounds against non-multilinear circuits.

**Motivation:** While one direction of research proceeds in proving lower bounds for shallow arithmetic circuits (motivated by the depth reduction results in [1, 24]), the other direction has been on proving lower bounds for unbounded depth circuits with additional structural restrictions.

Despite a large number of lower bound results in the directions mentioned above, the techniques for proving lower bounds presently available to us are very limited, owing to

difficulty in coming up with complexity measures that are sub-additive and sub-multiplicative. In this context, it is important to understand the strength and limitations of existing complexity measures for arithmetic circuits to see their applicability to general classes of arithmetic formulae/circuits. We explore classes of arithmetic formulas where the techniques developed in [19, 13] can be extended and applied. In particular, we consider models that serve as a bridge between shallow arithmetic formulas (e.g., depth two and three formulas) and restricted class of unbounded depth formulas (e.g. multilinear formulas).

**Models and Results:** Focus of the paper will be on shallow formulas built over restricted formulas of unbounded depth, i.e., a hybrid between bounded depth formulas and restricted formulas of possibly unbounded depth. To begin with we consider the simplest possible restricted formulas of unbounded depth:

► **Definition 1** (Read-Once Formula). A formula is said to be a *read-once formula* (ROF) if every variable labels at most one leaf in the formula. A polynomial computed by a read-once formula is called a *read-once polynomial* (ROP).

Observe that not all multilinear polynomials are read once. For instance, using the characterization of ROPs in [27] we can show that  $\det_n$  and  $\text{perm}_n$  are not read-once polynomial. Given that ROFs cannot compute all multilinear polynomials, it is natural to look for generalizations of ROFs that can compute all multilinear polynomials. As a first step, we consider the class  $\Sigma \cdot \text{ROF}$ : a polynomial  $g \in \mathbb{F}[x_1, \dots, x_n]$  is in  $\Sigma \cdot \text{ROF}$  if there exists ROFs  $f_1, f_2, \dots, f_s$  such that  $g = \sum_{i=1}^s f_i$ . Observe that  $\Sigma \cdot \text{ROF}$  is a subclass of multilinear unbounded depth formulas. Moreover, since each multilinear monomial is an ROP, any multilinear polynomial in  $\mathbb{F}[x_1, \dots, x_n]$  is in  $\Sigma \cdot \text{ROF}$ , thus making the model universal. It can be seen that the elementary symmetric polynomial in  $n$  variables of degree  $d$  denoted by  $\text{Sym}_{n,d}$  can be computed by linear size  $\Sigma \cdot \text{ROF}$  [25]. While the model  $\Sigma \cdot \text{ROF}$  is powerful enough to compute elementary symmetric polynomials, we study its limitations. We show:

► **Theorem 2.** *There is an explicit  $O(n)$  variate polynomial  $g \in \text{VP}$  such that for any ROFs  $f_1, \dots, f_s$ , if  $\sum_{i=1}^s f_i = g$ , then  $s = \exp(\Omega(n/\log n))$ .*

Shpilka and Volkovich [22] obtained a deterministic quasi polynomial time identity testing algorithm for the sum of a constant number of ROPs. An essential ingredient in their result was a linear lower bound for a special class of ROPs computing  $x_1 \cdots x_n$ . We note that Theorem 2 is an exponential lower bound against the same model as in [22] against a polynomial in VP defined by Raz-Yehudayoff [20].

► **Remark.** It should be noted that the result in Raz [19] immediately implies a lower bound of  $n^{\Omega(\log n)}$  for the sum of ROFs computing  $\det_n$  or  $\text{perm}_n$ . We exhibit a polynomial in VP that requires a sum of exponential many ROFs to compute it.

Having looked at a subclass of multilinear unbounded depth formulas it is natural to look for non-multilinear unbounded depth formulas. We now introduce our main computational model:  $\Sigma\Pi$  formulas built over ROFs ( $\Sigma\Pi \cdot \text{ROF}$  for short).

► **Definition 3** (Sum of Products of Read-Once Formula). A polynomial  $g \in \mathbb{F}[x_1, \dots, x_n]$  is in  $\Sigma\Pi \cdot \text{ROF}$  if there exists ROFs  $Q_{ij}, i \in [s], j \in [t]$  such that  $g = \sum_{i=1}^s \prod_{j=1}^t Q_{ij}$ .

Since linear forms are computable by ROFs,  $\Sigma\Pi \cdot \text{ROF}$  is a natural generalization of  $\Sigma\Pi\Pi$  formulas. As every variable is trivially computed by ROF, any polynomial in  $\mathbb{F}[x_1, \dots, x_n]$  can be computed by  $\Sigma\Pi \cdot \text{ROF}$ . Also,  $\Sigma\Pi \cdot \text{ROF}$  is a subclass of non-multilinear unbounded



depth formulas and it contains possibly non-homogeneous and non-multilinear polynomials built using the simplest possible multilinear formulas viz. ROFs. We observe that there is a simple ROF which computes a product of variable disjoint linear forms such that rank of the partial derivative matrix under a random partition is close to the maximum possible value with high probability (see Lemma 34). This necessitates further restrictions on ROFs that could lead to exponential lower bound against  $\Sigma\Pi \cdot \text{ROF}$  using the rank of the partial derivative matrix as the measure of complexity.

Let  $F$  be an ROF and for a gate  $v$  in  $F$ , let  $\text{sum-fan-in}(v)$  be the number of variables in the sub-formula rooted at  $v$  whose parents are labelled as  $+$ . Then  $s(F)$  is the maximum value of  $\text{sum-fan-in}(v)$ , where the maximum is taken over all  $+$  gates  $v$  in  $F$  of product height at least 1. For an ROP  $f$ , define  $s(f)$  as the smallest  $s(F)$  among all ROFs  $F$  computing  $f$ . Observe that the construction in [25] shows that  $Sym_{n,d} \in \sum_i \prod_j Q_{ij}$  where each  $Q_{ij}$  is an ROF and  $s(Q_{ij}) = 1$ . Our main result is the following :

► **Theorem 4.** *Let  $\mathcal{C}$  be the class of  $N$ -variate ROFs  $F$  with  $s(F) \leq N^{1/4}$ . For  $N = n^2$ , if  $\text{perm}_n = \sum_{i=1}^s \prod^{[N^{1/30}]} \mathcal{C}$  then  $s = \exp(\Omega(N^\epsilon))$  for some  $\epsilon > 0$ .*

As far as we know, in the commutative setting, this is the first exponential lower bound for a sub-class of non-multilinear formulas of unbounded depth. In the non-commutative setting, Nisan [16] showed that  $\det_n$  and  $\text{perm}_n$  require  $2^{\Omega(n)}$  size non-commutative arithmetic formula. It can be noted that our result above does not depend on the depth of the ROFs. Having proved an exponential lower bound against permanent which is in the class  $\text{VNP}$ , it is natural to ask if there are polynomials in  $\text{VP}$  that are hard to be computed by the model. We show the following :

► **Theorem 5.** *Let  $\mathcal{C}$  be the class of  $N$ -variate ROFs  $F$  with  $s(F) \leq N^{1/4}$ . Let  $N = n^2$ . Then there is an explicit family of polynomials  $p_{lin}$  such that if  $p_{lin} = \sum_{i=1}^s \prod^{[N^{1/30}]} \mathcal{C}$  then  $s = \exp(\Omega(N^\epsilon))$ , for some  $\epsilon > 0$ .*

Since multilinear  $\Sigma\Pi\Sigma$  circuits can be viewed as sum of depth two ROPs, we have the following corollary of Theorem 5,

► **Corollary 6.** *Let  $\mathcal{C}$  be the class of  $N$ -variate polynomials computed by multilinear depth three  $\sum^{[r]} \prod \sum^{[N^{1/4}]}$  formulas. Then there is an explicit family of polynomials  $p_{lin}$  such that if  $p_{lin} = \sum_{i=1}^s \prod^{[N^{1/30}]} \mathcal{C}$  then  $s \cdot r = \exp(\Omega(N^\epsilon))$ , for some  $\epsilon > 0$ .*

**Related Results:** In [15], Mahajan and Tawari obtain a tight linear lower bound for number of ROPs required to sum-represent elementary symmetric polynomials. That is, they show that the elementary symmetric polynomial  $Sym_n^{n-1}$  can be written as a sum of  $\lceil n/2 \rceil$  ROPs but cannot be written as a sum of  $k$  ROPs for any  $k < \lceil n/2 \rceil$ . Though the model in [15] is the same as the one in this paper, our lower bound shows that there is an explicit polynomial  $g$  that requires exponentially many ROPs to sum represent  $g$ . Kayal [8] showed that at least  $2^{n/d}$  many polynomials of degree  $d$  are required to represent the polynomial  $x_1 \dots x_n$  as sum of powers. Our model is significantly different from the one in [8] since our model includes high degree monomials, though the powers are restricted to be sub-linear, whereas Kayal's argument works against arbitrary powers.

**Our Techniques:** Our techniques are broadly based on the rank of polynomial coefficient matrix introduced by Kumar et. al. [13] as an extension of the partial derivative matrix introduced in [19]. It can be noted that the lower bounds obtained in [19] are super polynomial

and not exponential. Though Raz-Yehudayoff [21] proved exponential lower bounds, their argument works only against bounded depth multilinear circuits. Further, the arguments in [19, 21] do not work for the case of non-multilinear circuits, and fail even in the case of products of two multilinear formulas. This is because rank of the partial derivative matrix, a complexity measure used in [19, 21] (see Section 2 for a definition) is defined only for multilinear polynomials. Even though this issue can be overcome by a generalization introduced by Kumar et. al. [13], the limitation lies in the fact that the upper bound of  $2^{n-n^\epsilon}$  for an  $n^2$  or  $2n$  variate polynomial, obtained in [19] or [21] on the measure for the underlying arithmetic formula model is insufficient to handle products of two ROPs.

Our approach to prove Theorems 4 and 5 lie in obtaining exponentially stronger upper bounds (see Lemma 33) on the rank of the partial derivative matrix of an ROP  $F$  on  $N$  variables where  $s(F) \leq N^{1/4}$ . Our proof is a technically involved analysis of the structure of ROPs under random partitions of the variables. Even though the restriction on  $s(F)$  might look un-natural, in Lemma 34, we show that a simple product of variable disjoint linear forms in  $N$ -variables, with  $s(F) \geq N^{2/3}$  achieves exponential rank with probability  $1 - 2^{-\Omega(N^{1/3})}$ . Thus our results highlight the strength and limitations of the techniques developed in [21, 13] in the case of non-multilinear formulas.

The rest of the paper is organized as follows. Section 2 provides essential definitions used in the paper. Section 3 proves Theorem 2. Sections 4 proves the remaining results. Proofs omitted due to space constraints can be found in the full version of the paper [18].

## 2 Preliminaries

In this section we recall some basic definitions and introduce notations used in this article.

► **Definition 7 (Arithmetic Circuits).** Let  $\mathbb{F}$  be a field and  $X = \{x_1, \dots, x_N\}$  be a set of variables. An *arithmetic circuit*  $\mathcal{C}$  over  $\mathbb{F}$  is a directed acyclic graph with vertices of in-degree 0 or 2 and exactly one vertex of out-degree 0 called the output gate. The vertices of in-degree 0 are called *input gates* and are labeled by elements from  $X \cup \mathbb{F}$ . The vertices of in-degree 2 are labeled by either  $+$  or  $\times$ . Thus every gate of the circuit naturally computes a polynomial. The polynomial  $f$  computed by  $\mathcal{C}$  is the polynomial computed by the output gate of the circuit. The *size* of an arithmetic circuit is the number of gates in  $\mathcal{C}$ . The *depth* of  $\mathcal{C}$  is the length of the longest path from an input gate to the output gate in  $\mathcal{C}$ . An arithmetic circuit is called an *arithmetic formula* if the underlying undirected graph is a tree.

The *product height* of a gate  $v$  in  $\mathcal{C}$  is the maximum number of  $\times$  gates along any path from  $v$  to the root gate in  $\mathcal{C}$ . For  $g$  any gate in a circuit  $\mathcal{C}$ ,  $\text{var}(g)$  denote the set of variables that appear as leaf labels in the sub-circuit rooted at  $g$ . Abusing the notation, if  $g$  is a polynomial, then  $\text{var}(g)$  denotes the set of variables that  $g$  is dependent on. We now review the polynomial coefficient matrix introduced in [13]. Let  $\mathbb{F}$  be a field and  $X = \{x_1, \dots, x_N\}, Y = \{y_1, \dots, y_m\}$  and  $Z = \{z_1, \dots, z_m\}$  be disjoint sets of variables.

► **Definition 8 (Polynomial Coefficient Matrix).** Let  $f \in \mathbb{F}[Y, Z]$  be a polynomial. The *polynomial coefficient matrix* of  $f$  (denoted by  $M_f$ ) is a  $2^m \times 2^m$  matrix defined as : For monic multilinear monomials  $p$  and  $q$  in variables  $Y$  and  $Z$  respectively, the entry  $M_f[p, q] = A$  if and only if  $f$  can be uniquely expressed as  $f = pq \cdot A + B$  where  $A, B \in \mathbb{F}[Y, Z]$  such that (1)  $\text{var}(A) \subseteq \text{var}(p) \cup \text{var}(q)$  and (2) for every monomial  $m \in B$ , either  $pq \nmid m$  or  $\text{var}(m) \subsetneq \text{var}(p) \cup \text{var}(q)$ .

► **Observation 9.** For a multilinear polynomial  $f \in \mathbb{F}[Y, Z]$ , the polynomial coefficient matrix [13] and the partial derivative matrix [19] are the same.

The matrix  $M_f$  has entries in  $\mathbb{F}[Y, Z]$ . Therefore  $\text{rank}(M_f)$  is defined only under a substitution function. For  $\mathcal{S} : Y \cup Z \rightarrow \mathbb{F}$ , let  $M_f|_{\mathcal{S}}$  be the matrix obtained by substituting every variable  $w \in Y \cup Z$  to  $\mathcal{S}(w)$  at each entry of  $M_f$ .

$$\text{maxrank}(M_f) \triangleq \max_{\mathcal{S}: Y \cup Z \rightarrow \mathbb{F}} \{\text{rank}(M_f|_{\mathcal{S}})\}$$

It is known that  $\text{maxrank}(M_f)$  satisfies sub-additivity and sub-multiplicativity. The proofs of Lemma 10 and 11 follow directly from [13].

► **Lemma 10** (Sub-additivity, [13]). *Let  $f, g \in \mathbb{F}[Y, Z]$ . Then, we have that  $\text{maxrank}(M_{f+g}) \leq \text{maxrank}(M_f) + \text{maxrank}(M_g)$ .*

► **Lemma 11** (Sub-multiplicativity, [13]). *Let  $Y_1, Y_2 \subseteq Y$  and  $Z_1, Z_2 \subseteq Z$ . Then for any polynomials  $f \in \mathbb{F}[Y_1, Z_1]$ ,  $g \in \mathbb{F}[Y_2, Z_2]$ , we have  $\text{maxrank}(M_{fg}) \leq \text{maxrank}(M_f) \cdot \text{maxrank}(M_g)$ . Also, when  $Y_1 \cap Y_2 = \emptyset$  and  $Z_1 \cap Z_2 = \emptyset$  we have  $\text{maxrank}(M_{fg}) = \text{maxrank}(M_f) \cdot \text{maxrank}(M_g)$ .*

► **Observation 12.** *For any multilinear polynomial  $f \in \mathbb{F}[Y, Z]$ , the entries of  $M_f$  are constants from  $\mathbb{F}$ . Therefore  $\text{maxrank}(M_f) = \text{rank}(M_f)$ .*

► **Definition 13** (Partition function). A partition of  $X$  is a function  $\varphi : X \rightarrow Y \cup Z \cup \{0, 1\}$  such that  $\varphi$  is an injection when restricted to  $Y \cup Z$ , i.e.,  $\forall x \neq x' \in X$ , if  $\varphi(x) \in Y \cup Z$  and  $\varphi(x') \in Y \cup Z$  then  $\varphi(x) \neq \varphi(x')$ .

Let  $F$  be a formula with leaves labelled by elements in  $X \cup \mathbb{F}$  and  $\varphi : X \rightarrow Y \cup Z \cup \{0, 1\}$  be a partition function as in Definition 13. Denote by  $F^\varphi$  to be the formula obtained by replacing every variable  $x$  that appears as a leaf in  $F$  by  $\varphi(x)$ . Denote by  $f^\varphi$  the polynomial computed by  $F^\varphi$ . Then  $f^\varphi \triangleq f(\varphi(X)) \in \mathbb{F}[Y, Z]$ .

Consider a formula  $F$  all of whose leaves are labelled by constants. Then  $F$  computes a constant say  $\alpha$ . Observe that in this case for any partition function  $\varphi : X \rightarrow Y \cup Z \cup \{0, 1\}$ , we have  $\text{rank}(M_{\alpha^\varphi}) = 1$ . However, Lemmas 10 and 11 we may get  $\text{rank}(M_{\alpha^\varphi})$  as large as exponential in size of  $F$ . Hence we need a notion of formulas that use constants from  $\mathbb{F}$  in a minimal fashion :

► **Definition 14** (Constant-Minimal Formula). An arithmetic formula  $F$  is said to be *constant-minimal* if no gate  $u$  in  $F$  has both its children as constants from  $\mathbb{F}$ .

Observe that for any arithmetic formula  $F$ , if there exists a gate  $u$  in  $F$  such that  $u = a \text{ op } b, a, b \in \mathbb{F}$  then we can replace  $u$  in  $F$  by the constant  $a \text{ op } b$ , where  $\text{op} \in \{+, \times\}$ . Thus we assume without loss of generality that any arithmetic formula  $F$  is constant-minimal.

We state some observations on formulas that compute natural numbers. An arithmetic formula  $F$  is said to be monotone if no leaf in  $F$  is labelled by negative constants. Let  $G$  be a monotone arithmetic formula where the leaves are labelled numbers in  $\mathbb{N}$ . Then for any gate  $v$  in  $G$ , the value of  $v$  (denoted by  $\text{value}(v)$ ) is defined as : If  $u$  is a leaf then  $\text{value}(u) = a$  where  $a \in \mathbb{N}$  is the label of  $u$ . If  $u = u_1 \text{ op } u_2$  then  $\text{value}(u) = \text{value}(u_1) \text{ op } \text{value}(u_2)$ , where  $\text{op} \in \{+, \times\}$ . Finally,  $\text{value}(G)$  is the value of the output gate of  $G$ .

► **Lemma 15.** *Let  $G$  be a binary monotone arithmetic formula with  $t$  leaves. If every leaf in  $G$  takes a value at most  $N > 1$ , then  $\text{value}(G) \leq N^t$ .*

► **Definition 16** (*(rank-(1,2)-separator)*). Let  $G$  be a monotone arithmetic formula with leaves labelled by either 1 or 2. A node  $u$  in  $G$  at product height at least 1 is called a *rank-(1,2)-separator* if  $u$  is a leaf and  $\text{value}(u) = 2$  or  $u$  is a sum gate ( $u = u_1 + u_2$ ) with  $\text{value}(u) \geq 2$  and  $\text{value}(u_1), \text{value}(u_2) < 2$ .

► **Lemma 17.** *Let  $F$  be a binary monotone arithmetic formula with leaves labelled by either 1 or 2. Suppose  $\text{value}(F) > 2^r$  then there are at least  $\lceil \frac{r}{\log N} \rceil$  gates that are rank-(1,2)-separators, where  $N$  is the sum of labels of leaves in  $F$ .*

Finally, we state the following variants of the well known Chernoff-Hoeffding bounds.

► **Theorem 18** (Chernoff-Hoeffding bound, [3]). *Let  $X_1, X_2, \dots, X_n$  be independent random variables. Let  $X = X_1 + X_2 + \dots + X_n$  and  $\mu = \mathbb{E}[X]$ . Then for any  $\delta > 0$ ,*

1.  $\Pr[X \geq (1 + \delta)\mu] \leq e^{-\frac{\delta^2 \mu}{3}}$  when  $0 < \delta < 1$ ; and
2.  $\Pr[X \leq (1 - \delta)\mu] \leq e^{-\frac{\delta^2 \mu}{2}}$  when  $0 < \delta < 1$ ; and
3.  $\Pr[X \geq (1 + \delta)\mu] \leq e^{-\frac{\delta \mu}{3}}$  when  $\delta > 1$

### 3 Hardness of representation for Sum of ROPs

Let  $X = \{x_1, \dots, x_{2n}\}, Y = \{y_1, \dots, y_{2n}\}, Z = \{z_1, \dots, z_{2n}\}$ . Define  $\mathcal{D}'$  as a distribution on the functions  $\varphi : X \rightarrow Y \cup Z$  as follows : For  $1 \leq i \leq 2n$ ,

$$\varphi(x_i) \in \begin{cases} Y & \text{with prob. } \frac{1}{2} \\ Z & \text{with prob. } \frac{1}{2} \end{cases}$$

Observe that  $|\varphi(X) \cap Y| = |\varphi(X) \cap Z|$  is not necessarily true. Let  $F$  be a binary arithmetic formula computing a polynomial  $f$  on the variables  $X = \{x_1, \dots, x_{2n}\}$ . Note that any gate with at least one variable as a child can be classified as:

1. type- $A$  gates: sum gates both of whose children are variables; and
2. type- $B$  gates: product gates both of whose children are variables; and
3. type- $C$  gates: sum gates exactly one child of which is a variable; and
4. type- $D$  gates: product gates exactly one child of which is a variable.

Given any ROF  $F$ , let there be  $a$  type- $A$  gates,  $b$  type- $B$ ,  $c$  type- $C$  and  $d$  type- $D$  gates in  $F$ . Note that  $2a + 2b + c + d \leq 2n$ .

► **Observation 19.** *Let  $F$  be a binary arithmetic formula. Then there is a formula  $F'$  computing the same polynomial as  $F$  such that no root to leaf path in  $F'$  has two consecutive type- $C$  gates. Therefore, for any binary formula  $F$ , without the loss of generality we have  $c \leq a + b + d$ .*

We say a gate  $G$  computing a polynomial  $g$  achieves rank-1 under  $\varphi$  if  $\text{rank}(M_{g^\varphi}) = 1$  and we say the gate  $G$  achieves rank-2 under  $\varphi$  if  $\text{rank}(M_{g^\varphi}) = 2$ . Let  $\varphi \sim \mathcal{D}'$ . Let there be  $a'$  gates of type- $A$  that achieve rank-1 under  $\varphi$  and let  $a''$  gates of type- $A$  that achieve rank-2 under  $\varphi$ . Then,  $a = a' + a''$ . The following lemma bounds the rank of  $M_{f^\varphi}$ .

► **Lemma 20.** *Let  $F$  be an ROF computing an ROP  $f$  and  $\varphi : X \rightarrow Y \cup Z$ . Then,  $\text{rank}(M_{f^\varphi}) \leq 2^{a'' + \frac{a'}{2} + \frac{2b}{3} + \frac{c}{2}}$ , where  $a'', a', b$  and  $c$  are as defined above.*

► **Lemma 21.** *Let  $F$  be a ROF. Let there be  $a$  type- $A$  gates in  $F$  and  $a'$  be the number type- $A$  gates in  $F$  that achieve rank-1 under  $\varphi \sim \mathcal{D}$ . Then,  $\Pr_{\varphi \sim \mathcal{D}'} [\frac{2}{5}a \leq a' \leq \frac{3}{5}a] = 1 - 2^{-a/100}$ .*

**Proof.** Let  $v$  be a type- $A$  gate in  $F$ . Then  $f_v = x_i + x_j$  for some  $i, j \in [N]$ . Then  $\Pr[\text{rank}(M_{f_v^\varphi}) = 1] = \Pr[(\varphi(x_i), \varphi(x_j)) \in Z] \vee (\varphi(x_i), \varphi(x_j)) \in Y] = \frac{1}{2}$ . Therefore,  $\mu = \mathbb{E}[a'] = a/2$ . Applying Theorem 18 (2) and (3) with  $\delta = 1/5$ , we get the required bounds. ◀

► **Lemma 22.** *Let  $f$  be an ROP on  $2n$  variables and  $\varphi \sim \mathcal{D}'$ . Then with probability at least  $1 - 2^{-\Omega(\frac{n}{\log n})}$ ,  $\text{rank}(M_{f^\varphi}) \leq 2^{n - \frac{n}{15 \log n}}$ .*

**Proof.** Let  $F$  be an ROF computing  $f$ , and  $a, b, c, d, a'$  and  $a''$  be as in the discussion preceding Lemma 20. We have two cases:

**Case 1:**  $a + c \geq \frac{2n}{\log n}$ . Then either  $a \geq \frac{n}{\log n}$  or  $c \geq \frac{n}{\log n}$ .

- (i) Suppose  $a \geq \frac{n}{\log n}$ , then by Lemma 20, we have  $\text{rank}(M_{f_\varphi}) \leq 2^{a''+a'/2+2b/3+c/2} \leq 2^{a''+a'/2+b+c/2}$ . Since  $2a'' + 2a' + 2b + c + d \leq 2n$ ,  $a'' + a'/2 + b + c/2 \leq n - a'/2$ . By Lemma 21,  $a' \geq \frac{2a}{5} \geq \frac{2n}{5 \log n}$  with probability  $1 - 2^{-\Omega(\frac{n}{\log n})}$ . Therefore,  $\text{rank}(M_{f_\varphi}) \leq 2^{a''+a'/2+b+c/2} \leq 2^{n-a'/2} \leq 2^{n-\frac{n}{5 \log n}}$ .
- (ii) Suppose  $c \geq \frac{n}{\log n}$ . By Observation 19,  $a + b + d \geq c \geq \frac{n}{\log n}$ , then either  $a \geq \frac{n}{3 \log n}$  or  $b \geq \frac{n}{3 \log n}$  or  $d \geq \frac{n}{3 \log n}$ .
- If  $a \geq \frac{n}{3 \log n}$ , similar to (i) we have  $\text{rank}(M_{f_\varphi}) \leq 2^{n-\frac{n}{15 \log n}}$  with probability  $1 - 2^{-\Omega(\frac{n}{\log n})}$ .
  - If  $b \geq \frac{n}{3 \log n}$  by Lemma 20,  $\text{rank}(M_{f_\varphi}) \leq 2^{a+2b/3+c/2}$ . Since  $2a + 2b + c + d \leq 2n$ , we have  $a + \frac{c}{2} \leq n - b$ . Therefore  $\text{rank}(M_{f_\varphi}) \leq 2^{n-\frac{b}{3}} \leq 2^{n-\frac{n}{9 \log n}} \leq 2^{n-\frac{n}{15 \log n}}$ .
  - If  $d \geq \frac{n}{3 \log n}$ , since  $2a + 2b + c + d \leq 2n$ ,  $a + b + \frac{c}{2} \leq n - \frac{d}{2}$ . Therefore by Lemma 20  $\text{rank}(M_{f_\varphi}) \leq 2^{a''+a'/2+2b/3+c/2} \leq 2^{a+b+c/2} \leq 2^{n-\frac{d}{2}} \leq 2^{n-\frac{n}{6 \log n}} \leq 2^{n-\frac{n}{15 \log n}}$ .

**Case 2:**  $a + c < \frac{2n}{\log n}$ . Observe that  $b \leq n$ . By Lemma 20,  $\text{rank}(M_{f_\varphi}) \leq 2^{a+2b/3+c} \leq 2^{2n/3+2n/\log n} \leq 2^{n-n/15 \log n}$  for large enough  $n$ . ◀

The following polynomial was introduced by Raz and Yehudayoff [20].

► **Definition 23.** Let  $n \in \mathbb{N}$  be an integer. Let  $X = \{x_1, \dots, x_{2n}\}$  and  $\mathcal{W} = \{w_{i,k,j}\}_{i,k,j \in [2n]}$ . For any two integers  $i, j \in \mathbb{N}$ , we define an interval  $[i, j] = \{k \in \mathbb{N}, i \leq k \leq j\}$ . Let  $|[i, j]|$  be the length of the interval  $[i, j]$ . Let  $X_{i,j} = \{x_p \mid p \in [i, j]\}$  and  $W_{i,j} = \{w_{i',k,j'} \mid i', k, j' \in [i, j]\}$ . Let  $\mathbb{G} = \mathbb{F}(\mathcal{W})$ , the rational function field. For every  $[i, j]$  such that  $|[i, j]|$  is even we define a polynomial  $g_{i,j} \in \mathbb{G}[X]$  as  $g_{i,j} = 1$  when  $|[i, j]| = 0$  and if  $|[i, j]| > 0$  then,  $g_{i,j} \triangleq (1+x_i x_j)g_{i+1,j-1} + \sum_k w_{i,k,j} g_{i,k} g_{k+1,j}$ , where  $x_k, w_{i,k,j}$  are distinct variables,  $1 \leq k \leq j$  and the summation is over  $k \in [i+1, j-2]$  such that  $|[i, k]|$  is even. Let  $g \triangleq g_{1,2n}$ .

The following lemma builds on Lemma 4.3 in [20].

► **Lemma 24.** Let  $X = \{x_1, \dots, x_{2n}\}$ ,  $Y = \{y_1, \dots, y_{2n}\}$ ,  $Z = \{z_1, \dots, z_{2n}\}$  and  $\mathcal{W} = \{w_{i,k,j}\}_{i,k,j \in [2n]}$  be sets of variables. Suppose  $\varphi \sim \mathcal{D}'$  such that  $|\varphi(X) \cap Y| - |\varphi(X) \cap Z| = \ell$ . Then for the polynomial  $g$  as in Definition 23 we have,  $\text{rank}(M_{g^\varphi}) \geq 2^{n-\ell/2}$ .

► **Lemma 25.** For  $\mathcal{Q} \in \{Y, Z\}$ ,  $\Pr_{\varphi \sim \mathcal{D}'}[n - n^{2/3} \leq |\varphi(X) \cap \mathcal{Q}| \leq n + n^{2/3}] \geq 1 - 2^{-\Omega(n^{1/3})}$ .

**Proof.** Proof is a simple application of Chernoff's bound (Theorem 18) with  $\delta = 1/n^{1/3}$ . ◀

► **Corollary 26.**  $\Pr_{\varphi \sim \mathcal{D}'}[\text{rank}(M_{g^\varphi}) \geq 2^{n-n^{2/3}}] \geq 1 - 2^{-\Omega(n^{1/3})}$ .

**Proof.** Apply Lemma 24 with  $\ell = 2n/n^{1/3} = 2n^{2/3}$  and apply Lemma 25. ◀

## Proof of Theorem 2

**Proof.** Suppose  $s < \exp(o(n/\log n))$ . Then by Lemma 22 and union bound, probability that there is an  $i$  such that  $\text{rank}(M_{f_\varphi}) \geq \exp(n - n/15 \log n)$  is  $s \exp(-\Omega(n/\log n)) = \exp(-\Omega(n/\log n))$  and hence by Lemma 10,  $\text{rank}(M_{g^\varphi}) \leq s \exp(n - n/15 \log n) \leq \exp(n - n/20 \log n)$  with probability  $1 - \exp(-\Omega(n/\log n))$  for large enough  $n$ . However, by Corollary 26,  $\text{rank}(M_{g^\varphi}) \geq \exp(n - n^{2/3}) > \exp(n - n/20 \log n)$  with probability at least  $1 - \exp(-\Omega(n^{1/3}))$ , a contradiction. Therefore,  $s = \exp(\Omega(n/\log n))$ . ◀

## 4 Sum of Products of ROPs

### 4.1 ROPs under random partition

Throughout the section, let  $m \triangleq N^{1/3}$ ,  $N \triangleq n^2$  and  $\kappa \triangleq 20 \log n$ . Let  $X = \{x_{11}, \dots, x_{nm}\}$  be a set of  $n^2$  variables and  $\mathcal{D}$  denote the distribution on the functions  $\varphi : X \rightarrow Y \cup Z \cup \{0, 1\}$  defined as follows

$$\varphi(x_{ij}) \in \begin{cases} Y & \text{with prob. } \frac{m}{N} \\ Z & \text{with prob. } \frac{m}{N} \\ 1 & \text{with prob. } \frac{\kappa n}{N} \\ 0 & \text{with prob. } 1 - \left(\frac{2m + \kappa n}{N}\right) \end{cases}$$

The following Lemmas show that bottom  $\times$  gates do not contribute much to the rank.

► **Lemma 27.** *Let  $F$  be a ROF and  $\varphi \sim \mathcal{D}$ . Let  $\mathcal{X}$  be a random variable that denotes the number of non-zero multiplication gates at depth 1. Then  $\Pr_{\varphi \sim \mathcal{D}} [\mathcal{X} > (N^{1/4})] \leq \exp(-\Omega(N^{1/4}))$ .*

► **Lemma 28.** *Let  $F$  be an ROF computing an ROP  $f$  and  $\varphi \sim \mathcal{D}$ . Then there exists an ROF  $F'$  such that every gate in  $F'$  at depth-1 is an addition gate, and  $\text{rank}(M_{F\varphi}) \leq \text{rank}(M_{F'\varphi}) \cdot \exp(\mathcal{O}(N^{1/4}))$  with probability atleast  $1 - \exp(-\Omega(N^{1/4}))$ .*

Recall that an arithmetic formula  $F$  over  $\mathbb{Z}$  is said to be monotone if it does not have any node labelled by a negative constant. We have:

► **Lemma 29.** *Let  $F$  be an ROF, and  $\varphi \sim \mathcal{D}$ . Then there exists a monotone formula  $G$  such that  $\text{rank}(M_{F\varphi}) \leq \text{value}(G)$ .*

► **Observation 30.** *Let  $F$  be an ROF and  $\varphi \sim \mathcal{D}$ . By Lemma 29, we have,  $\Pr[\text{rank}(M_{F\varphi}) > 2^r] \leq \Pr[\text{value}(G) > 2^r]$ .*

Let  $F$  be an ROF and  $\varphi \sim \mathcal{D}$ . Then by Lemma 17 we have the following corollary,

► **Corollary 31.**

$$\Pr[\text{rank}(M_{F\varphi}) > 2^r] \leq \Pr[\exists u_1, \dots, u_{\frac{r}{\log N}} \in F^\varphi \text{ s.t. } \forall i \ u_i \text{ is a rank-}(1, 2)\text{-separator}].$$

Now all we need to do is to estimate the probability that a given set of nodes  $u_1, \dots, u_t$  where  $t > \frac{r}{\log N}$  are a set of rank- $(1, 2)$ -separators.

► **Lemma 32.**  *$F$  be an ROF and let  $u_1, \dots, u_t$  be a set of  $+$  gates in  $F$  that have product height at least 1 and are not descendants of each other. Suppose  $s(F) \leq N^{1/4}$ . Then  $\Pr_{\varphi}[\bigwedge_{i=1}^t u_i \text{ is a rank-}(1, 2)\text{-separator}] \leq c^t N^{-5t/6}$ , for some constant  $c > 0$ .*

**Proof.** Note that for  $1 \leq i \leq t$   $\text{rank}(M_{u_i^\varphi}) = 2$  only if  $|\text{var}(u_i^\varphi) \cap Y| \geq 1$  and  $|\text{var}(u_i^\varphi) \cap Z| \geq 1$ . Therefore  $\Pr[u_i \text{ is a } (1, 2) \text{ separator}] \leq \Pr[|\text{var}(u_i^\varphi) \cap Y| \geq 1 \text{ and } |\text{var}(u_i^\varphi) \cap Z| \geq 1] \leq \Pr[|\text{var}(u_i^\varphi) \cap (Y \cup Z)| \geq 2]$ . Let  $\ell_{i_1}, \dots, \ell_{i_{r_i}}$  be the addition gates at depth-1 in the subformula rooted at  $u_i$ . For  $0 \leq i \leq t$ , we define  $S_i \triangleq \text{var}(\ell_{i_1}) \cup \dots \cup \text{var}(\ell_{i_{r_i}})$ . Then for  $0 \leq i \leq t$ ,  $\Pr[u_i \text{ is a } (1, 2) \text{ separator}] \leq \Pr[|S_i \cap (Y \cup Z)| \geq 2]$ . Since  $|\text{var}(u_i)| \leq s(F)$ , we have  $|S_i| \leq s(F) \leq N^{1/4}$ . Since  $(1 - 2m/N)^{|S_i|-2} \leq 1$ ,  $|S_i| \leq N^{1/4}$  and  $m = N^{1/3}$ , we have

$$\begin{aligned} \Pr[|S_i \cap (Y \cup Z)| \geq 2] &= \binom{|S_i|}{2} \left(\frac{2m}{N}\right)^2 (1 - 2m/N)^{|S_i|-2} \leq \binom{|S_i|}{2} \left(\frac{2m}{N}\right)^2 \\ &\leq 2^2 s(F)^2 N^{-4/3} = \mathcal{O}(N^{-5/6}). \end{aligned}$$



Similarly,  $\Pr[|S_i \cap (Y \cup Z)| = 3] \leq \mathcal{O}(N^{-5/4})$ . By union bound  $\Pr[|S_i \cap (Y \cup Z)| \geq 3] \leq |Y \cup Z| \Pr[|S_i \cap (Y \cup Z)| = 3] \leq N^{-11/12} \leq \mathcal{O}(N^{-5/6})$ . Then for some constant  $c > 0$

$$\Pr_{\varphi} \left[ \bigwedge_{i=1}^t u_i \text{ is a } (1, 2) \text{ separator} \right] \leq \prod_{i=1}^t \Pr[|S_i \cap (Y \cup Z)| \geq 2] \leq \prod_{i=1}^t \mathcal{O}(N^{-5/6}) = c^t N^{-5t/6} \blacktriangleleft$$

► **Lemma 33.** *Let  $f$  be an ROP on  $N$  variables computed by an ROF  $F$ , with  $s(F) \leq N^{1/4}$ . Then,  $\Pr_{\varphi \sim \mathcal{D}}[\text{rank}(M_{f\varphi}) \geq 2^{N^{4/15}}] \leq 2^{-\Omega(N^{1/4})}$ .*

**Proof.** By Lemma 28, note that  $\times$  gates in  $F$  with at least two variables as their input contribute a multiplicative factor of  $2^{N^{1/4}}$  to  $\text{rank}(M_{f\varphi})$  with probability at least  $1 - 2^{-\Omega(N^{1/4})}$ . Thus, without loss of generality we can assume that  $F$  has no  $\times$  gate with at more than two variables as its input. By Corollary 31 we have

$$\begin{aligned} \Pr[\text{rank}(M_{f\varphi}) \geq 2^{N^{4/15}}] &\leq \Pr[\exists \text{ rank-}(1, 2)\text{-separators } u_1, \dots, u_{\frac{N^{4/15}}{\log N}}] \\ &\leq \Pr[\exists \text{ rank-}(1, 2)\text{-separators } u_1, \dots, u_{N^{1/4}}] \\ &\leq \binom{N}{N^{1/4}} c^{N^{1/4}} N^{-\frac{5}{6}N^{1/4}} \\ &\leq c^{N^{1/4}} e^{N^{1/4}} N^{(3/4)N^{1/4} - (5/6)N^{1/4}} \leq N^{-\Omega(N^{1/4})}. \end{aligned}$$

The penultimate inequality follows by Lemma 32 and union bound. For the last inequality, we use the fact that  $\binom{n}{k} \leq (ne/k)^k$ , where  $e$  is the base of natural logarithm.  $\blacktriangleleft$

## 4.2 Polynomials with High Rank

In this section, we prove rank lower bounds for two polynomials under a random partition  $\varphi \sim \mathcal{D}$ . The first one is in VP and the other one is in VNP.

► **Lemma 34.** *Let  $p_{lin} = \ell_1 \cdots \ell_{m'}$  where  $\ell_j = \left( \sum_{i=(j-1)(N/2m)+1}^{jN/2m} x_i \right) + 1$ , where  $m' = 2m$ . Then,  $\text{rank}(M_{p_{lin}\varphi}) = \exp(\Omega(m))$  with probability  $1 - \exp(-\Omega(m))$ .*

**Proof.** Let  $p_{lin} = \ell_1 \cdots \ell_{m'}$  where  $\ell_j = \left( \sum_{i=(j-1)(N/2m)+1}^{jN/2m} x_i \right) + 1$  and  $m' = 2m$ .

Define indicator random variables  $\rho_1, \rho_2, \dots, \rho_{m'}$ , where  $\rho_i = 1$  if  $\text{rank}(M_{\ell_i\varphi}) = 2$  and 0 otherwise. Observe that for any  $1 \leq i \leq m'$ ,  $\text{rank}(M_{\ell_i\varphi}) = 2$  iff  $\ell_i^\varphi \cap Y \neq \emptyset$  and  $\ell_i^\varphi \cap Z \neq \emptyset$ . Therefore,  $\Pr[\text{rank}(M_{\ell_i\varphi}) = 2] = \Pr[\ell_i^\varphi \cap Y \neq \emptyset \text{ and } \ell_i^\varphi \cap Z \neq \emptyset]$ . For any  $1 \leq j \leq m'$ ,  $\Pr[\ell_j^\varphi \cap Y \neq \emptyset \text{ and } \ell_j^\varphi \cap Z \neq \emptyset] \geq \frac{N}{2m} \left( \frac{N}{2m} - 1 \right) \left( \frac{m}{N} \right)^2 \left( 1 - \frac{m}{N} \right)^{\frac{N}{2m} - 2} \geq 1/16$  for large enough  $N$ . Let  $\rho = \sum_{i=1}^{m'} \rho_i$ . Then by linearity of expectation,  $\mu \triangleq \mathbb{E}[\rho] = \sum_{i=1}^{m'} \mathbb{E}[\rho_i] \geq \frac{m}{8}$ . Since  $\mu \geq m/8$ , we have  $\Pr[\rho < (1 - \delta)m/8] \leq \Pr[\rho < (1 - \delta)\mu] = \exp(-\Omega(m))$  by Theorem 18 with  $\delta = 1/4$ , since  $\text{rank}(M_{p_{lin}\varphi}) = \exp(\rho)$ .  $\blacktriangleleft$

Throughout the section let  $\varphi$  denote a function of the form  $\varphi : X \rightarrow Y \cup Z \cup \{0, 1\}$ . Let  $X_\varphi$  denote the matrix  $(\varphi(x_{ij}))_{1 \leq i, j \leq n}$ . If and when  $\varphi$  involved in a probability argument, we assume that  $\varphi$  is distributed according to  $\mathcal{D}$ .

► **Definition 35.** Let  $1 \leq i, j \leq n$ .  $(i, j)$  is said to be a *Y-special* (respectively *Z-special*) if  $\varphi(x_{ij}) \in Y$  (respectively  $\varphi(x_{ij}) \in Z$ ),  $\forall i' \in [n], i' \neq i \varphi(x_{i'j}) \in \{0, 1\}$  and  $\forall j' \in [n], j' \neq j \varphi(x_{ij'}) \in \{0, 1\}$ .

► **Lemma 36.** *Let  $\mathcal{Q} \in \{Y, Z\}$ ,  $\varphi$  as above and  $\chi = |\varphi(X) \cap \mathcal{Q}|$  where  $\varphi(X) = \{\varphi(x_{ij})\}_{i, j \in [n]}$ . Then,  $\Pr_{\varphi \sim \mathcal{D}}[3m/4 < \chi < 5m/4] = 1 - \exp(-\Omega(m))$ .*



Let  $C_1, \dots, C_n$  denote the columns of  $X_\varphi$  and  $R_1, \dots, R_n$  denote the rows of  $X_\varphi$ .

► **Definition 37.** Let  $\mathcal{Q} \in \{Y, Z\}$ . A column  $C_j$ ,  $1 \leq j \leq n$  is said to be  $\mathcal{Q}$ -good if  $\exists i \in [n]$ ,  $\varphi(x_{ij}) \in \mathcal{Q}$ ; and  $\forall i' \in [n], i' \neq i$   $\varphi(x_{i'j}) \in \{0, 1\}$ .  $\mathcal{Q}$ -good rows are defined analogously.

► **Observation 38.** Let  $C_i$  be a  $Y$ -good column in  $X_\varphi$ . Let  $i, i' \in [n]$ ,  $\mathcal{R}$  be the event that  $\varphi(x_{ij}) \in Y$  and  $\mathcal{T}$  be the event that  $\varphi(x_{i'j}) \in Y$ . The events  $\mathcal{R}$  and  $\mathcal{T}$  are mutually exclusive.

By Observation 38 and union bound we have:

► **Lemma 39.** For  $1 \leq i \leq n$ , let  $C_i$  be a column in  $X_\varphi$ . Then for any  $\mathcal{Q} \in \{Y, Z\}$ ,  $\Pr_{\varphi \sim \mathcal{D}}[C_i \text{ is } \mathcal{Q}\text{-good}] = n \cdot \frac{m}{N} \left(1 - \frac{2m}{N}\right)^{n-1}$ .

For  $\mathcal{Q} \in \{Y, Z\}$  let  $\eta_{\mathcal{Q}} \triangleq |\{C_i \mid C_i \text{ is } \mathcal{Q}\text{-good}\}|$  and  $\zeta_{\mathcal{Q}} \triangleq |\{R_j \mid R_j \text{ is } \mathcal{Q}\text{-good}\}|$ .

► **Lemma 40.** With notations as above,  $\forall \mathcal{Q} \in \{Y, Z\}$ ,  $\Pr_{\varphi \sim \mathcal{D}}[\eta_{\mathcal{Q}} \geq \frac{2m}{3}] = 1 - \exp(-\Omega(m))$ ; and  $\Pr_{\varphi \sim \mathcal{D}}[\zeta_{\mathcal{Q}} \geq \frac{2m}{3}] = 1 - \exp(-\Omega(m))$ .

► **Lemma 41.** For  $\mathcal{Q} \in \{Y, Z\}$ , let  $\gamma_{\mathcal{Q}}$  denote the number of  $\mathcal{Q}$ -special positions in  $X_\varphi$ . Then  $\forall \mathcal{Q} \in \{Y, Z\}$ ,  $\Pr_{\varphi \sim \mathcal{D}}[\gamma_{\mathcal{Q}} \geq \frac{m}{12}] = 1 - \exp(-\Omega(m))$ .

**Proof.** We argue for  $\mathcal{Q} = Y$ , the proof is analogous when  $\mathcal{Q} = Z$ . Let  $\varphi$  be distributed according to  $\mathcal{D}$ . Consider the following events on  $X_\varphi$ . E1 :  $2m/3 \leq |X_\varphi \cap Y| \leq 5m/4$ ; E2 : The number of  $Y$ -good columns and  $Y$ -good rows is at least  $r \triangleq 2m/3$ . By Lemmas 36 and 40,  $X_\varphi$  satisfies the events E1 and E2 with probability  $1 - \exp(-\Omega(m))$ . Henceforth we assume that  $X_\varphi$  satisfies the events E1 and E2.

Without loss of generality, let  $R_1, \dots, R_r$  be the first  $r$   $Y$ -good rows in  $X_\varphi$ . For every  $Y$ -good row  $R_i$ ,  $1 \leq i \leq r$  there exists a corresponding witness column  $C_j, j \in [n]$  such that  $\varphi(x_{ij}) \in Y$ . Without loss of generality, assume  $C_1, \dots, C_r$  be columns that are witnesses for  $R_1, \dots, R_r$  being  $Y$ -good. Further, let  $X_\varphi(C_j)$  denote the set of values along the column  $C_j$ . Each of the column  $C_j$  has at least one variable from  $Y$  and hence the columns  $C_1, \dots, C_t$  contain at least  $t$  distinct variables from  $Y$ . By event E2, there are at least  $\frac{2m}{3}$   $Y$ -good columns that are distinct from  $C_1, \dots, C_t$ , each containing exactly one distinct variable from  $Y$ . Since the total number of variables from  $Y$  in  $X_\varphi$  is at most  $5m/4$  (by E1) we have,  $t \leq 5m/4 - 2m/3 \leq 7m/12$ . That is, at most  $7m/12$  of the columns among  $C_1, \dots, C_r$  are not  $Y$ -good. Therefore, at least  $r - t$  of the columns among  $C_1, \dots, C_r$  are  $Y$  good and hence the number of  $Y$ -special positions in  $X_\varphi$  is atleast  $r - t \geq (2/3 - 7/12)m = m/12$ . We conclude,  $\Pr_{\varphi \sim \mathcal{D}}[\gamma_Y \geq m/12] = 1 - \exp(-\Omega(m))$ . ◀

A row  $R$  in the matrix  $A \in (Y \cup Z \cup \{0, 1\})^{n \times n}$  said to be 1-good if there is at least one 1 in  $R$  in a column other than  $Y$ -special and  $Z$ -special positions. The following is immediate :

► **Observation 42.** Let  $\varphi$  be distributed according to  $\mathcal{D}$ . Then for any row (column)  $R$ :  $\Pr_{\varphi \sim \mathcal{D}}[R \text{ is 1-good}] \geq (1 - 1/n^3)$ .

Finally, we are ready to show that perm has high rank under a random  $\varphi \sim \mathcal{D}$ .

► **Theorem 43.**  $\Pr[\text{rank}(M_{\text{perm}_n^\varphi}) \geq 2^{m/12}] \geq (1 - O(1/n^2))/2$ .

We need a few notations and Lemmas before proving Theorem 43. Consider a  $\varphi : X \rightarrow Y \cup Z \cup \{0, 1\}$  and let the number of  $Y$ -special positions and the number of  $Z$ -special positions in  $X_\varphi$  are both be at least  $\gamma$ . Let  $(i_1, j_1), (i_2, j_2), \dots, (i_\gamma, j_\gamma)$  be a set of distinct  $Y$ - special

$$A = \left( \begin{array}{c|c} \begin{array}{cccc} \overbrace{B_1 \ * \ \cdots \ *}^{2\gamma \text{ columns}} & \overbrace{\ * \ * \ \cdots \ *}^{n-2\gamma \text{ columns}} \\ * & B_2 \ * \ \cdots \ * \\ * & * \ B_3 \ \cdots \ * \\ \vdots & \vdots \ \vdots \ \vdots \ \vdots \\ * & * \ * \ \cdots \ B_\gamma \end{array} & \begin{array}{cccc} * & * & \cdots & * \\ * & * & \cdots & * \\ * & * & \cdots & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & \cdots & * \end{array} \\ \hline \begin{array}{cccc} * & * & \underbrace{\ * \ \cdots \ *}_{A'} & * \\ * & * & * & \cdots \ * \\ * & * & * & \cdots \ * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & * & \cdots \ * \end{array} & \begin{array}{cccc} * & * & \cdots & * \\ * & * & \cdots & * \\ * & * & \cdots & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & \cdots & * \end{array} \\ \hline \end{array} \right) \begin{array}{l} \left. \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array} \right\} 2\gamma \text{ rows} \\ \left. \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array} \right\} (n-2\gamma) \text{ rows} \\ \underbrace{\hspace{10em}}_{A''} \end{array}$$

■ **Figure 1** The matrix  $A$  after permuting the rows and columns.  $*$  denotes unspecified entries.

positions that do not share any row or column and  $(k_1, \ell_1), (k_2, \ell_2), \dots, (k_\gamma, \ell_\gamma)$  be a set of distinct  $Z$ -special positions in  $X_\varphi$  that do not share any row or column.

Without loss of generality, suppose  $i_1 < i_2 < \dots < i_\gamma$  and  $k_1 < k_2 < \dots < k_\gamma$ . Let  $\mathcal{M}$  be the perfect matching  $((i_1, j_1), (k_1, \ell_1)), \dots, ((i_\gamma, j_\gamma), (k_\gamma, \ell_\gamma))$ . For an edge  $\{(i_p, j_p), (k_p, \ell_p)\} \in \mathcal{M}$ ,  $1 \leq p \leq \gamma$  consider the  $2 \times 2$  matrix :

$$B_p = \begin{pmatrix} X_\varphi[i_p, j_p] & X_\varphi[i_p, \ell_p] \\ X_\varphi[k_p, j_p] & X_\varphi[k_p, \ell_p] \end{pmatrix}.$$

There exists a partition  $\varphi : X \rightarrow Y \cup Z \cup \{0, 1\}$  such that  $\text{rank}(M_{B_p^\varphi}) = 2$ . Let  $A$  be the matrix obtained by permuting the rows and columns in  $X_\varphi$  such that  $A$  can be written as in the Figure 1.

Since  $(i_p, j_p)$  is a  $Y$ -special position,  $(k_p, \ell_p)$  is a  $Z$ -special position we have  $X_\varphi[i_p, j_p] \in Y$ ,  $X_\varphi[k_p, \ell_p] \in Z$ . Also  $X_\varphi[i_p, \ell_p] \in \{0, 1\}$  and  $X_\varphi[k_p, j_p] \in \{0, 1\}$ . Further,  $\text{rank}(M_{\text{perm}(B_p)}) = 2$  if and only if  $X_\varphi[k_p, j_p] = X_\varphi[i_p, \ell_p] = 1$ . Consider the following events:  $F_1$ :  $\gamma \geq m/12$ ; and  $F_2$ : Rows  $i_1, \dots, i_\gamma, k_1, \dots, k_\gamma$  are 1-good. The following lemma estimates the probability of  $\text{perm}(A'') \neq 0$ .

► **Lemma 44.** *Let  $A''$  be matrix as in Figure 1. Then  $\Pr_\varphi[\text{perm}(A'') \neq 0 \mid F_1, F_2] \geq 1 - \frac{1}{n^2}$ .*

Let  $F_3$  denote the event “ $\text{perm}(A'') \neq 0$ ”. Define sets of matrices:

$$\mathcal{A} \triangleq \left\{ X_\varphi \mid \begin{array}{l} X_\varphi \in F_1 \cap F_2 \cap F_3 \text{ and } \exists i \leq \gamma \\ \text{rank}(M_{\text{perm}(B_i)}) = 1 \end{array} \right\}; \quad \mathcal{B} \triangleq \left\{ X_\varphi \mid \begin{array}{l} X_\varphi \in F_1 \cap F_2 \cap F_3 \text{ and } \forall i \leq \gamma \\ \text{rank}(M_{\text{perm}(B_i)}) = 2. \end{array} \right\}$$

► **Observation 45.**  $\forall A \in \mathcal{A}, \text{rank}(M_{\text{perm}(A)}) < 2\gamma$  and  $\forall B \in \mathcal{B}, \text{rank}(M_{\text{perm}(B)}) \geq 2\gamma$ .

► **Lemma 46.** *Let  $\mathcal{A}$  and  $\mathcal{B}$  as defined above. Then*

- (a)  $\Pr_{\varphi \sim \mathcal{D}}[\text{rank}(M_{\text{perm}(X_\varphi)}) \geq 2\gamma] \geq \mathcal{D}(\mathcal{B})$ ; and
- (b)  $\mathcal{D}(\mathcal{B}) \geq \mathcal{D}(\mathcal{A})$ , where  $\mathcal{D}(S) = \Pr_{\varphi \sim \mathcal{D}}[X_\varphi \in S]$  for  $S \in \{\mathcal{A}, \mathcal{B}\}$ .

**Proof.** (a) follows from Observation 45. For (b), we establish a one-one mapping  $\pi : \mathcal{A} \rightarrow \mathcal{B}$  defined as follows. Let  $\varphi$  be such that  $X_\varphi \in \mathcal{A}$ . Consider  $1 \leq p \leq \gamma$  such that  $\text{rank}(M_{\text{perm}(B_p)}) = 1$ . Then either  $X_\varphi[k_p, j_p] = 0$  or  $X_\varphi[i_p, \ell_p] = 0$  or both. If  $X_\varphi[k_p, j_p] = 0$ ,

then set  $X_{\varphi'}[k_p, j_p] = 1$ , and  $X_{\varphi'}[k_p, \iota_p] = 0$  where  $\iota_p \in [n] \setminus \{j_1, \dots, j_\gamma, \ell_1, \dots, \ell_\gamma\}$  is the first index from left such that  $X_\varphi[k_p, \iota_p] = 1$ . Similarly, if  $X_\varphi[i_p, \ell_p] = 0$ , then set  $X_{\varphi'}[i_p, \ell_p] = 1$ , and  $X_{\varphi'}[i_p, \lambda_p] = 0$  where  $\lambda_p \in [n] \setminus \{j_1, \dots, j_\gamma, \ell_1, \dots, \ell_\gamma\}$  is the first index from left such that  $X_\varphi[k_p, \lambda_p] = 1$ . Let  $\varphi'$  be the partition obtained from  $\varphi$  by applying the above mentioned swap operation for every  $1 \leq p \leq \gamma$  with  $\text{rank}(M_{\text{perm}(B_p)}) = 1$ , keeping other values of  $\varphi$  untouched. Clearly  $X_{\varphi'} \in \mathcal{B}$ . Set  $\pi(X_\varphi) \mapsto X_{\varphi'}$ . It can be seen that  $\pi$  is an one-one map. Further, for any fixed  $A \in \mathcal{A}$ ,  $\Pr_\varphi[X_\varphi = A] = \Pr_{\varphi'}[X_{\varphi'} = \pi(A)]$  since  $\varphi$  is independently and identically distributed for any position in the matrix. Thus we have  $\mathcal{D}(\mathcal{A}) \leq \mathcal{D}(\mathcal{B})$ . ◀

**Proof of Theorem 43.** It is enough to argue that  $\Pr_{\varphi \sim \mathcal{D}}[X_\varphi \in \mathcal{A} \cup \mathcal{B}] = 1 - O(\frac{1}{n^2})$ , as  $\mathcal{A} \cap \mathcal{B} = \emptyset$ . Now,  $\Pr_{\varphi \sim \mathcal{D}}[X_\varphi \in \mathcal{A} \cup \mathcal{B}] = \Pr_{\varphi \sim \mathcal{D}}[F_1 \cap F_2 \cap F_3]$ . By Lemma 41,  $\Pr_{\varphi \sim \mathcal{D}}[F_1] = 1 - 2^{-\Omega(m)}$ . From Observation 42 and the union bound we have  $\Pr_{\varphi \sim \mathcal{D}}[F_2] \geq 1 - \gamma/n^3$ . By Lemma 44,  $\Pr_{\varphi \sim \mathcal{D}}[F_3 | F_1, F_2] \geq 1 - 2/n^2$ . Thus we conclude  $\Pr_{\varphi \sim \mathcal{D}}[F_1 \cap F_2 \cap F_3] = 1 - O(\frac{1}{n^2})$ . As  $\mathcal{D}(\mathcal{B} \cup \mathcal{A}) = \mathcal{D}(\mathcal{A}) + \mathcal{D}(\mathcal{B})$ , by Lemma 46,  $\Pr_{\varphi \sim \mathcal{D}}[\text{rank}(M_{\text{perm}(X_\varphi)}) \geq 2^\gamma] \geq 1/2(1 - O(\frac{1}{n^2}))$ . ◀

### 4.3 Putting them all together

#### Proof of Corollary 6

**Proof.** Suppose  $p_{lin} = \sum_{i=1}^s \prod_{j=1}^t f_{i,j}$  where  $f_{i,j}$  are syntactically multilinear  $\Sigma\Pi\Sigma$  formula, with  $s < \exp(N^{1/4})$ . Let  $f_{i,j} = \sum_{k=1}^{s'} T_{i,j,k}$ , and  $T_{i,j,k}$  are products of variable disjoint linear forms, and hence ROPs. Further, since the bottom fan-in of each  $f_{i,j}$  is bounded by  $N^{1/4}$ , we have  $s_{T_{i,j,k}} \leq \exp(N^{1/4})$ . Then by Lemma 33 and union bound there is an  $i, j, k$  such that  $\text{rank}(M_{T_{i,j,k}^\varphi}) \geq \exp(N^{4/15})$  with probability at most  $sts' \exp(-\Omega(N^{1/4}))$ . By Lemma 10 and 11, we have  $\text{maxrank}(M_{p_{lin}^\varphi}) \leq 2^{N^{4/15}}$  with probability  $1 - o(1)$ . However by Lemma 34,  $\text{maxrank}(M_{p_{lin}^\varphi}) = \text{rank}(M_{p_{lin}^\varphi}) = \exp(\Omega(m))$  with probability at least  $1 - \exp(-\Omega(m))$ , a contradiction. Hence  $ss' = \exp(\Omega(N^{1/4}))$ . ◀

#### Proof of Theorem 5

**Proof.** Suppose  $s = \exp(o(N^{1/4}))$ . Then by Lemma 33, the probability that there is an  $f_{i,j}$  with  $\text{rank}(M_{f_{i,j}^\varphi}) \geq \exp(N^{4/15})$  is at most  $\exp(-\Omega(N^{1/4}))s = o(1)$ . By Lemma 10 and 11 and since  $\text{maxrank}(M_{f_{i,j}^\varphi}) = \text{rank}(M_{f_{i,j}^\varphi})$ , we have  $\text{maxrank}(M_{p_{lin}^\varphi}) \leq (s \cdot \exp(N^{4/15}))^{N^{1/30}} = \exp(o(N^{1/3}))$  with probability  $1 - o(1)$ . However by Lemma 34,  $\text{maxrank}(M_{p_{lin}^\varphi}) = \exp(\Omega(m))$  with probability  $1 - \exp(-\Omega(m))$ , a contradiction. Hence  $s = \exp(\Omega(N^{1/4}))$ . ◀

#### Proof of Theorem 4

**Proof.** Suppose  $s = \exp(o(N^{1/4}))$ . Then by Lemma 33, Probability that there is an  $f_{i,j}$  with  $\text{rank}(M_{f_{i,j}^\varphi}) \geq \exp(N^{4/15})$  is at most  $\exp(-\Omega(N^{1/4}))s = o(1)$ . Then, by Lemma 10 and 11, we have  $\text{maxrank}(M_{\text{mathitperm}_n^\varphi}) \leq s \cdot (\exp(N^{4/15}))^{N^{1/30}} = \exp(o(N^{1/3}))$  with probability  $1 - o(1)$ . However, by Theorem 43,  $\text{maxrank}(M_{\text{mathitperm}_n^\varphi}) = \text{rank}(M_{\text{mathitperm}_n^\varphi}) \exp(\Omega(m))$  with probability  $(1 - 1/n^2)/2$ , a contradiction. Hence  $s = \exp(\Omega(N^{1/4}))$ . ◀

**Acknowledgements.** We thank anonymous reviewers of an earlier version of the paper for suggestions which improved the presentation. Further, we thank one of the anonymous reviewers for pointing an observation that lead to Lemma 20.

---

**References**

---

- 1 Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *FOCS*, pages 67–75, 2008. doi:10.1109/FOCS.
- 2 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.
- 3 Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.
- 4 Michael Forbes. Polynomial identity testing of read-once oblivious algebraic branching programs. *PhD thesis, Massachusetts Institute of Technology*, 2014.
- 5 Michael A. Forbes and Amir Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 243–252, 2013.
- 6 Dima Grigoriev and Marek Karpinski. An exponential lower bound for depth 3 arithmetic circuits. In *STOC*, pages 577–582, 1998. doi:10.1145/276698.276872.
- 7 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Approaching the chasm at depth four. *J. ACM*, 61(6):33:1–33:16, 2014. doi:10.1145/2629541.
- 8 Neeraj Kayal. An exponential lower bound for the sum of powers of bounded degree polynomials. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:81, 2012.
- 9 Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. Super-polynomial lower bounds for depth-4 homogeneous arithmetic formulas. In *STOC*, pages 119–127, 2014. doi:10.1145/2591796.2591823.
- 10 Neeraj Kayal and Chandan Saha. Multi-k-ic depth three circuit lower bound. In *STACS*, pages 527–539, 2015. doi:10.4230/LIPIcs.STACS.2015.527.
- 11 Neeraj Kayal, Chandan Saha, and Sébastien Tavenas. An almost cubic lower bound for depth three arithmetic circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:6, 2016. Accepted at ICALP 2016. URL: <http://eccc.hpi-web.de/report/2016/006>.
- 12 Pascal Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theor. Comput. Sci.*, 448:56–65, 2012. doi:10.1016/j.tcs.2012.03.041.
- 13 Mrinal Kumar, Gaurav Maheshwari, and Jayalal Sarma. Arithmetic circuit lower bounds via maximum-rank of partial derivative matrices. *TOCT*, 8(3):8, 2016. doi:10.1145/2898437.
- 14 Mrinal Kumar and Shubhangi Saraf. On the power of homogeneous depth 4 arithmetic circuits. In *FOCS*, pages 364–373, 2014. doi:10.1109/FOCS.2014.46.
- 15 Meena Mahajan and Anuj Tawari. Sums of read-once formulas: How many summands suffice? In *Computer Science – Theory and Applications – 11th International Computer Science Symposium in Russia, CSR 2016, St. Petersburg, Russia, June 9-13, 2016, Proceedings*, pages 266–279, 2016. doi:10.1007/978-3-319-34171-2\_19.
- 16 Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 410–418, 1991. doi:10.1145/103418.103462.
- 17 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. doi:10.1007/BF01294256.
- 18 C. Ramya and B. V. Raghavendra Rao. Limitations of sum of products of read-once polynomials. *CoRR*, abs/1512.03607, 2015. URL: <https://arxiv.org/abs/1512.03607>.
- 19 Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2), 2009. doi:10.1145/1502793.1502797.

- 20 Ran Raz and Amir Yehudayoff. Balancing syntactically multilinear arithmetic circuits. *Computational Complexity*, 17(4):515–535, 2008.
- 21 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.
- 22 Amir Shpilka and Ilya Volkovich. Read-once polynomial identity testing. *Computational Complexity*, 24(3):477–532, 2015. doi:10.1007/s00037-015-0105-8.
- 23 Amir Shpilka and Avi Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Computational Complexity*, 10(1):1–27, 2001. doi:10.1007/PL00001609.
- 24 Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. *Inf. Comput.*, 240:2–11, 2015. doi:10.1016/j.ic.2014.09.004.
- 25 Iddo Tzameret. Studies in algebraic and propositional proof complexity. *Ph.D Thesis*, page 33, 2008. URL: <http://www.cs.rhul.ac.uk/home/tzameret/Iddo-PhD-thesis.pdf>.
- 26 Leslie G. Valiant. Completeness classes in algebra. In *STOC*, pages 249–261, 1979. doi:10.1145/800135.804419.
- 27 Ilya Volkovich. Characterizing arithmetic read-once formulae. *TOCT*, 8(1):2, 2016. doi:10.1145/2858783.



# Understanding Cutting Planes for QBFs\*

Olaf Beyersdorff<sup>1</sup>, Leroy Chew<sup>2</sup>, Meena Mahajan<sup>3</sup>, and Anil Shukla<sup>4</sup>

1 School of Computing, University of Leeds, United Kingdom

2 School of Computing, University of Leeds, United Kingdom

3 The Institute of Mathematical Sciences, HBNI, Chennai, India

4 The Institute of Mathematical Sciences, HBNI, Chennai, India

---

## Abstract

We define a cutting planes system  $CP+\forall\text{red}$  for quantified Boolean formulas (QBF) and analyse the proof-theoretic strength of this new calculus. While in the propositional case, Cutting Planes is of intermediate strength between resolution and Frege, our findings here show that the situation in QBF is slightly more complex: while  $CP+\forall\text{red}$  is again weaker than QBF Frege and stronger than the CDCL-based QBF resolution systems Q-Res and QU-Res, it turns out to be incomparable to even the weakest expansion-based QBF resolution system  $\forall\text{Exp}+\text{Res}$ .

Technically, our results establish the effectiveness of two lower bound techniques for  $CP+\forall\text{red}$ : via strategy extraction and via monotone feasible interpolation.

**1998 ACM Subject Classification** F.2.2 [Nonnumerical Algorithms and Problems] Complexity of Proof Procedures

**Keywords and phrases** proof complexity, QBF, cutting planes, resolution, simulations

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.40

## 1 Introduction

The main problem of *proof complexity* is to understand the minimal size of proofs for natural classes of formulas in important proof systems. Proof complexity deeply connects to a number of other areas, most notably computational complexity, circuit complexity, first-order logic, and practical solving. Recently the connection to practical solving has been a main driver for the field. Modern SAT solvers routinely solve huge industrial instances of the NP-hard SAT problem with even millions of variables. Because runs of the solver on unsatisfiable formulas can be interpreted as proofs for unsatisfiability in a system corresponding to the solver, proof complexity provides the main theoretical tool for an understanding of the power and limitations of these algorithms.

During the last decade there has been great interest and research activity to extend the success of SAT solvers to the more expressive *quantified Boolean formulas (QBF)*. Due to its PSPACE completeness (even for restricted versions [2]), QBF is far more expressive than SAT and thus applies to further fields such as formal verification or planning [34, 5, 21].

Triggered by this exciting development in QBF solving, *QBF proof complexity* has seen a stormy development in past years. A number of resolution-based systems have been designed with the aim to capture ideas in QBF solving. Broadly, these systems can be classified into

---

\* This work was supported by the EU Marie Curie IRSES grant CORCON, grant no. 48138 from the John Templeton Foundation, EPSRC grant EP/L024233/1, and a Doctoral Training Grant from EPSRC (2nd author).





two types corresponding to two principal approaches in QBF solving: proof systems modelling *conflict driven clause learning (CDCL)*: Q-resolution Q-Res [29, 7], universal resolution QU-Res [38], long-distance resolution [3], and their extensions [4]; and proof systems modelling *expansion solving*:  $\forall\text{Exp}+\text{Res}$  [28] and their extensions [7]. Proof complexity research of these systems resulted in a complete understanding of the relative complexity of QBF resolution systems [8, 4], and the transfer of classical techniques to QBF systems was thoroughly assessed [9, 10, 11]. In addition, stronger QBF Frege and Gentzen systems were defined and investigated [20, 6, 12].

Most SAT and QBF solvers use resolution as their underlying proof system. Resolution is a weak proof systems for which a wealth of lower bounds and in fact lower bound techniques are known (cf. [37, 16]). This raises the question – often controversially discussed within the proof complexity and solving communities – whether it would be advantageous to build solvers on top of more powerful proof systems. While Frege systems appear too strong and proof search is hindered by non-automatisability results [31, 14], a natural system of intermediate strength is Cutting Planes first defined in [19].

Using ideas from integer linear programming [25, 17], Cutting Planes works with linear inequalities, allowing addition of inequalities as well as multiplication and division by positive integers as rules. Translating propositional clauses into inequalities, Cutting Planes derives the contradiction  $0 \geq 1$ , thereby demonstrating that the original set of inequalities (and hence the corresponding clause set) has no solution. As mentioned, Cutting Planes is a proof system of intermediate strength: it simulates resolution, but allows short proofs for the famous pigeonhole formulas hard for resolution [27], while it is simulated by and strictly weaker than Frege [24, 33].

## Our contributions

For QBFs a similar Cutting Planes system based on integer linear programming has been missing. It is the aim of this paper to define a natural Cutting Planes system for QBF and give a comprehensive analysis of its proof complexity.

**1. Cutting Planes for QBF.** We introduce a complete and sound QBF proof system  $\text{CP}+\forall\text{red}$  that works with quantified linear inequalities, where each variable is either quantified existentially or universally in a quantifier prefix. The system  $\text{CP}+\forall\text{red}$  extends the classical Cutting Planes system with one single  $\forall$ -reduction rule allowing manipulation of universally quantified variables. The definition of the system thus naturally aligns with the QBF resolution systems Q-Res [29] and QU-Res [38] and the stronger QBF Frege systems [6] that likewise add universal reduction to their classical base systems.

Inspired by the recent work on semantic Cutting Planes [23] we also define a stronger system  $\text{semCP}+\forall\text{red}$  where in addition to universal reduction all semantically valid inferences between inequalities are allowed (Section 7).

**2. Lower bound techniques for  $\text{CP}+\forall\text{red}$ .** We establish two lower bound methods for  $\text{CP}+\forall\text{red}$ : strategy extraction (Section 4) and feasible interpolation (Section 5).

*Strategy extraction* as a lower bound technique was first devised for Q-Res [8] and subsequently extended to QBF Frege systems [6, 12]. The technique applies to calculi that allow to efficiently extract winning strategies for the universal player from a refutation (or alternatively Skolem functions for the existential variables from a proof of a true QBF). Here we show that  $\text{CP}+\forall\text{red}$  admits strategy extraction in  $\text{TC}^0$ , thus establishing an appealing link between  $\text{CP}+\forall\text{red}$  proofs (which can count) and the counting circuit class  $\text{TC}^0$  (Theorem 8).

For each function  $f \in \text{PSPACE/poly}$  we construct false QBFs  $Q_{\text{qbf}}f_n$  where each winning strategy forces the universal player to compute  $f$ . Thus assuming the existence of  $f \in \text{PSPACE/poly} \setminus \text{TC}^0$  we obtain lower bounds for  $Q_{\text{qbf}}f_n$  in  $\text{CP}+\forall\text{red}$  (Corollary 9) and even  $\text{semCP}+\forall\text{red}$  (Corollary 21).

*Feasible interpolation* is another classical technique transferring circuit lower bounds to proof size lower bounds; however, here we import lower bounds for monotone arithmetic circuits [33] and hence the connection between the circuits and the lines in the proof system is less direct than in strategy extraction. Feasible interpolation holds for classical resolution [30] and Cutting Planes [33], and indeed was shown to be effective for all QBF resolution systems [9]. Following the approach of [33] we establish this technique for  $\text{CP}+\forall\text{red}$  (Theorem 12) and in fact for the stronger  $\text{semCP}+\forall\text{red}$  (Theorem 22).

It is interesting to note that while feasible interpolation is the only technique known for classical Cutting Planes, we have two conceptually different lower bound methods – and hence more (conditionally) hard formulas in QBF. This is in line with recent findings in [12] showing that lower bounds for QBF Frege either stem from circuit lower bounds (for  $\text{NC}^1$ ) or from classical Frege lower bounds. Our results here illustrate the same paradigm for  $\text{CP}+\forall\text{red}$ : lower bounds arise either from  $\text{TC}^0$  lower bounds (via strategy extraction) or via classical lower bound methods for Cutting Planes (feasible interpolation).

**3. Relations to other QBF proof systems.** We compare our new system  $\text{CP}+\forall\text{red}$  with previous QBF resolution and Frege systems. In contrast to the classical setting, the emerging picture is somewhat more complex: while  $\text{CP}+\forall\text{red}$  is strong enough to simulate the core CDCL QBF resolution systems Q-Res and QU-Res and indeed is exponentially stronger than these systems (Theorem 17),  $\text{CP}+\forall\text{red}$  is incomparable (under a natural circuit complexity assumption) to even the base system  $\forall\text{Exp}+\text{Res}$  of the expansion resolution systems (Theorem 18). Conceptually, this means that in contrast to the SAT case, QBF solvers based on linear programming and corresponding to  $\text{CP}+\forall\text{red}$  will not encompass the full strength of current resolution-based QBF solving techniques.

On the other hand,  $\text{CP}+\forall\text{red}$  turns out to be simulated by  $\text{Frege}+\forall\text{red}$ , and  $\text{Frege}+\forall\text{red}$  is exponentially more powerful than  $\text{CP}+\forall\text{red}$  (Theorem 19). While this separation could be achieved by lifting the classical separation [33] to QBF by considering purely existentially quantified formulas, we highlight that our separation also holds for natural QBFs expressing the clique-co-clique principle, which is not known to have a succinct propositional representation.

## 2 Notation and preliminaries

**Quantified Boolean Formulas.** A literal is a Boolean variable or its negation. We say a literal  $x$  is complementary to the literal  $\neg x$  and vice versa. A *clause* is a disjunction of literals and a *term* is a conjunction of literals. The empty clause is denoted by  $\square$ , and is semantically equivalent to false, denoted  $\perp$ . A formula in *conjunctive normal form* (CNF) is a conjunction of clauses. For a literal  $l = x$  or  $l = \neg x$ , we write  $\text{var}(l)$  for  $x$  and extend this notation to  $\text{var}(C)$  for a clause  $C$ . Let  $\alpha$  be any partial assignment. For a clause  $C$ , we write  $C|_\alpha$  for the clause obtained after applying the partial assignment  $\alpha$  to  $C$ .

Quantified Boolean Formulas (QBFs) extend propositional logic with Boolean quantifiers with the standard semantics that  $\forall x.F$  is satisfied by the same truth assignments as  $F|_{x=0} \wedge F|_{x=1}$  and  $\exists x.F$  as  $F|_{x=0} \vee F|_{x=1}$ . We assume that QBFs are in *closed prenex form* with a CNF matrix, i.e., we consider the form  $Q_1x_1 \cdots Q_nx_n . \phi$  where each  $Q_i$  is either  $\exists$  or  $\forall$ ,

and  $\phi$  is a quantifier-free CNF formula, called the matrix, in the variables  $x_1, \dots, x_n$ . Any QBF can be efficiently (in polynomial time) converted to an equivalent QBF in this form (using PSPACE-completeness of such QBFs). We denote such formulas succinctly as  $\mathcal{Q} . \phi$ . The *index*  $\text{ind}(y)$  of a variable  $y$  is its position in the prefix  $\mathcal{Q}$ ; for each  $i \in [n]$ ,  $\text{ind}(x_i) = i$ . If  $\text{ind}(x) < \text{ind}(y)$ , we say that  $x$  occurs *before*  $y$ , or *to the left of*  $y$ . The *quantification level*  $\text{lv}(y)$  of a variable  $y$  in  $\mathcal{Q} . \phi$  is the number of alternations of quantifiers to the left of  $y$  in the quantifier prefix of  $\mathcal{Q} . \phi$ . For instance, in the QBF  $\exists x_1 \forall x_2 \forall x_3 \exists x_4 \phi$ ,  $\text{lv}(x_1) = 1$ ,  $\text{lv}(x_2) = \text{lv}(x_3) = 2$ , and  $\text{lv}(x_4) = 3$ .

Often it is useful to think of a QBF  $\mathcal{Q}_1 x_1 \cdots \mathcal{Q}_n x_n . \phi$  as a game between two players: *universal* ( $\forall$ ) and *existential* ( $\exists$ ). In the  $i$ -th step of the game, the player  $\mathcal{Q}_i$  assigns a value to the variable  $x_i$ . The existential player wins if  $\phi$  evaluates to 1 under the assignment constructed in the game. The universal player wins if  $\phi$  evaluates to 0. A *strategy for*  $x_i$  is a function from all variables of index  $< i$  to  $\{0, 1\}$ . A *strategy* for the universal player is a collection of strategies, one for each universally quantified variable. Similarly, a *strategy* for the existential player is a collection of strategies, one for each existentially quantified variable. A strategy for the universal player is a winning strategy if using this strategy to assign values to variables, the universal player wins any possible game, irrespective of the strategy used by the existential player. Winning strategies for the existential player are similarly defined. For any QBF, exactly one of the two players has a winning strategy. A QBF is false if and only if there exists a *winning strategy* for the universal player ([26],[1, Sec. 4.2.2],[32, Chap. 19]).

**Proof systems.** Following notation from [18], a *proof system* for a language  $\mathcal{L}$  is a polynomial-time onto function  $f : \{0, 1\}^* \rightarrow \mathcal{L}$ . Each string  $\phi \in \mathcal{L}$  is a *theorem*, and if  $f(\pi) = \phi$ , then  $\pi$  is a *proof* of  $\phi$  in  $f$ . Given a polynomial-time function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  the fact that  $f(\{0, 1\}^*) \subseteq \mathcal{L}$  is the *soundness property* for  $f$  and the fact that  $f(\{0, 1\}^*) \supseteq \mathcal{L}$  is the *completeness property* for  $f$ .

Proof systems for the language of propositional unsatisfiable formulas (UNSAT) are called *propositional proof systems* and proof systems for the language of false QBFs are called *QBF proof systems*. These are *refutational* proof systems. Equivalently, propositional proof systems and QBF proof systems can be defined respectively for the languages of true propositional formulas (TAUT) and of true QBFs. Since any QBF  $\mathcal{Q} . \phi$  can be converted in polynomial time to another QBF  $\mathcal{Q}' . \phi'$  such that exactly one of  $\mathcal{Q} . \phi$  and  $\mathcal{Q}' . \phi'$  is true, it suffices to consider only refutational QBF proof systems.

Given two proof systems  $f_1$  and  $f_2$  for the same language  $L$ , we say that  $f_1$  simulates  $f_2$ , if there exists a function  $g$  and a polynomial  $p$  such that  $f_1(g(w)) = f_2(w)$  and  $|g(w)| \leq p(|w|)$  for all  $w$ . Thus  $g$  translates a proof  $w$  of  $x \in L$  in the system  $f_2$  into a proof  $g(w)$  of  $x \in L$  in the system  $f_1$ , with at most polynomial blow-up in proof-size. If there is such a  $g$  that is also polynomial-time computable, then we say that  $f_1$   $p$ -simulates  $f_2$ .

**QBF resolution calculi.** *Resolution* (Res), introduced by Blake [13] and Robinson [36], is a refutational proof system for formulas in CNF form. The lines in the Res proofs are clauses. The only inference (resolution) rule is  $\frac{C \vee x \quad D \vee \neg x}{C \cup D}$  where  $C, D$  denote clauses and  $x$  is a variable. A Res refutation derives the empty clause  $\square$ .

*Q-resolution* (Q-Res) [29] is a resolution-like calculus operating on QBFs in prenex form with a CNF matrix. The lines in the Q-Res proofs are clauses. It uses the propositional resolution rule above with the side conditions that variable  $x$  is existential, and if  $z \in C$ , then  $\neg z \notin D$ . (Unlike in the propositional case, dropping this latter condition that  $C \cup D$  is not a tautology can lead to unsoundness.) In addition Q-Res has the universal reduction rule

$\frac{C \vee u}{C}$  and  $\frac{C \vee \neg u}{C}$  ( $\forall$ -Red), where variable  $u$  is universal and every existential variable  $x \in C$  has  $\text{lv}(x) < \text{lv}(u)$ . If resolution is also permitted with universal variable  $x$  (as long as tautologies are not created), then we get the calculus QU-Res [38].

*Expansion-based* calculi are another type of resolution systems significantly different from Q-Res. In this paper, we will briefly refer to one such calculus, the  $\forall\text{Exp}+\text{Res}$  from [28]. In  $\forall\text{Exp}+\text{Res}$ , one expands the formula on universal variables, creating multiple annotated copies of existential variables, and then uses classical resolution. For details, see [28].

**Frege systems.** Frege proof systems are the ‘textbook’ proof systems for propositional logic based on axioms and rules [18]. A Frege system comprises a finite set of axiom schemes and rules. A *Frege proof* is a sequence of formulas (using  $\wedge, \vee, \neg$ ) where each formula is either a substitution instance of an axiom, or can be inferred from previous formulas by a valid inference rule. Frege systems are required to be sound and implicationally complete.

A refutation of a false QBF  $\mathcal{Q}.\phi$  in the system  $\text{Frege}+\forall\text{red}$  [6] is sequence of lines  $L_1, \dots, L_\ell$  where each line is a formula,  $L_1 = \phi$ ,  $L_\ell = \perp$  and each  $L_i$  is inferred from previous lines  $L_j$ ,  $j < i$ , using the inference rules of Frege or using the universal reduction rule  $\frac{L_j}{L_j[u/B]}$  ( $\forall\text{Red}$ ), where  $u$  is a universal variable and is the rightmost (highest index) variable among the variables of  $L_j$ ,  $B$  is a formula containing only variables left of  $u$ , and  $L_j[u/B]$  is the formula obtained from  $L_j$  by replacing each occurrence of  $u$  in  $L_j$  by  $B$ .

**Circuit classes.** We recall the definitions of some standard circuit classes (cf. [39]). The class  $\text{TC}^0$  contains all languages recognisable by polynomial-size circuits using  $\neg, \vee, \wedge$  and threshold gates with constant depth and unbounded fan-in. Stronger classes are obtained by using  $\text{NC}^1$  circuits of polynomial size and logarithmic depth with bounded fan-in  $\neg, \vee, \wedge$  gates, and by P/poly circuits of polynomial size. We use non-uniform classes throughout.

**Decision lists [35].** A *decision list* is a list  $L$  of pairs  $(t_1, v_1), \dots, (t_r, v_r)$ , where each  $t_i$  is a term and  $v_i$  is a value in  $\{0, 1\}$ , and the last term  $t_r$  is the constant term **true** (i.e., the empty term). A decision list  $L$  defines a Boolean function as follows: for any assignment  $\alpha$ ,  $L(\alpha)$  is defined to be equal to  $v_j$  where  $j$  is the least index such that  $t_j|_\alpha = 1$ . (Such an item always exists, since the last term always evaluates to 1). In [6], this definition has been generalised to  $\mathcal{C}$ -decision lists (for some circuit class  $\mathcal{C}$ ), where instead of terms one can use circuits from  $\mathcal{C}$ . A  $\mathcal{C}$ -decision list yields the circuit  $f(x) = \bigvee_{i=1}^r (v_i \wedge C_i(x) \wedge \bigwedge_{j < i} \neg C_j(x))$ . Thus a polynomial-sized  $\text{TC}^0$ -decision list yields a  $\text{TC}^0$  circuit.

### 3 The CP+ $\forall\text{red}$ proof system

In this section we define a QBF analogue of the classical Cutting Planes proof system by augmenting it with a reduction rule for universal variables. We denote this system by CP+ $\forall\text{red}$ . Consider a false quantified set of inequalities  $\mathcal{F} \equiv \mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n. F$ , where  $F$  is a set of linear inequalities of the form  $\sum x_i a_i \geq A$  for integers  $a_i$  and  $A$ , and  $F$  includes the set of inequalities  $B = \{x_i \geq 0, -x_i \geq -1 \mid i \in [n]\}$ . The inequalities in  $B$  are called the Boolean axioms, because they force any integer-valued assignment  $\bar{a}$  to the variables, satisfying  $F$ , to take only 0, 1-values. We point out that classical Cutting Planes proof systems (only existential variables) can refute any inconsistent set of linear inequalities over integers. However, once universal quantification is allowed, dealing with an unbounded domain is more

messy. Since our primary goal in defining this proof system is to refute false QBFs, and since QBFs have only Boolean variables, we only consider sets of inequalities that contain  $B$ .

► **Definition 1** (CP+ $\forall$ red proofs for inequalities). Consider a set of quantified inequalities  $\mathcal{F} \equiv \mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n. F$ , where  $F$  also contains the Boolean axioms. A CP+ $\forall$ red refutation  $\pi$  of  $\mathcal{F}$  is a quantified sequence of linear inequalities  $\mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n. [I_1, I_2, \dots, I_l]$  where the quantifier prefix is the same as in  $\mathcal{F}$ ,  $I_l$  is an inequality of the form  $0 \geq C$  for some positive integer  $C$ , and for every  $j \in \{1, \dots, l\}$ , either  $I_j \in F$ , or  $I_j$  is derived from earlier inequalities in the sequence via one of the following inference rules:

1. **Addition:** From  $\sum_k c_k x_k \geq C$  and  $\sum_k d_k x_k \geq D$ , derive  $\sum_k (c_k + d_k) x_k \geq C + D$ .
2. **Multiplication:** From  $\sum_k c_k x_k \geq C$ , derive  $\sum_k d c_k x_k \geq dC$ , where  $d \in \mathbb{Z}^+$ .
3. **Division:** From  $\sum_k c_k x_k \geq C$ , derive  $\sum_k \frac{c_k}{d} x_k \geq \left\lceil \frac{C}{d} \right\rceil$ , where  $d \in \mathbb{Z}^+$  divides each  $c_k$ .
4.  **$\forall$ -red:** From  $\sum_{k \in [n] \setminus \{i\}} c_k x_k + h x_i \geq C$ , derive  $\begin{cases} \sum_{k \in [n] \setminus \{i\}} c_k x_k \geq C & \text{if } h > 0; \\ \sum_{k \in [n] \setminus \{i\}} c_k x_k \geq C - h & \text{if } h < 0. \end{cases}$

This rule can be used provided variable  $x_i$  is universal, and provided all existential variables with nonzero coefficients in the hypothesis are to the left of  $x_i$  in the quantification prefix. (That is, if  $x_j$  is existential, then  $j > i \Rightarrow c_j = 0$ .) Observe that when  $h > 0$ , we are replacing  $x_i$  by 0, and when  $h < 0$ , we are replacing  $x_i$  by 1. We say that the universal variable  $x_i$  has been reduced.

Each inequality  $I_j$  is a line in the proof  $\pi$ . Note that proof lines are always of the form  $\sum_k c_k x_k \geq C$  for integer-valued  $c_k, C$ . The length of  $\pi$  (denoted  $|\pi|$ ) is the number of lines in it, and the size of  $\pi$  (denoted  $\text{size}(\pi)$ ) is the bit-size of a representation of the proof (this depends on the number of lines and the binary length of the numbers in the proof).

In order to use CP+ $\forall$ red as a refutational system for QBFs in prenex form with CNF matrix, we must translate QBFs into quantified sets of inequalities.

► **Definition 2** (Encoding QBFs as inequalities). We first describe how to encode a CNF formula  $F$  over variables  $x_1, \dots, x_n$  as a set of linear inequalities. Define  $R(x) = x$ ,  $R(\bar{x}) = 1 - x$ . A clause  $C \equiv (l_1 \vee \dots \vee l_k)$  is translated into the inequality  $R(C) \equiv \sum_{i=1}^k R(l_i) \geq 1$ . A CNF formula  $\phi = C_1 \wedge \dots \wedge C_m$  is represented as the set of inequalities  $F_\phi = \{R(C_1), R(C_2), \dots, R(C_m)\} \cup B$ , where  $B$  is the set of Boolean axioms  $x \geq 0, -x \geq -1$  for each variable  $x$ . We call this the standard encoding. For a QBF  $\mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n. \phi$  with a CNF matrix  $\phi$ , the encoding is the quantified set of linear inequalities  $\mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n. F_\phi$ .

We say that a 0, 1-assignment  $\alpha$  satisfies the inequality  $I \equiv \sum_{i=1}^n a_i x_i \geq b$  (i.e.,  $I|_\alpha = 1$ ), if  $\sum_{i=1}^n a_i \alpha_i \geq b$ . For any clause  $C$ , an assignment satisfies  $C$  if and only if it satisfies  $R(C)$ . Since the standard encoding includes all Boolean axioms, we obtain the following:

► **Proposition 3.** *Let  $\mathcal{Q}. \phi$  be a QBF in closed prenex CNF, and let  $\mathcal{F} = \mathcal{Q}. F_\phi$  be its encoding as a quantified set of linear inequalities. Then  $\mathcal{Q}. \phi$  is false if and only if  $\mathcal{F}$  is false.*

As for QBFs, we can play the 2-player game on the encoding  $\mathcal{F}$  of a QBF. Players choose 0-1 values for their variables in the order defined in the prefix. The  $\forall$  player wins if the assignment so constructed violates some inequality in  $F$ . As before, when  $\mathcal{F}$  is false, the universal player has a winning strategy; otherwise the existential player has a winning strategy.

► **Definition 4** (CP+ $\forall$ red proofs for QBFs). Let  $\mathcal{Q}.\phi = \mathcal{Q}_1x_1 \cdots \mathcal{Q}_nx_n . \phi$  be a false QBF in prenex CNF, and let  $\mathcal{F}$  be its encoding as a quantified set of linear inequalities. A CP+ $\forall$ red (refutation) proof of  $\mathcal{Q}.\phi$  is a CP+ $\forall$ red proof of  $\mathcal{F}$  as defined in Definition 1.

It is worth noting that a CP+ $\forall$ red proof for inequalities, as in Definition 1, can start with encodings of QBFs, but can also start with quantified sets of inequalities that contain the Boolean axioms but do not correspond to any QBF, since the initial non-Boolean inequalities can have arbitrary integer coefficients.

Observe that in the  $\forall$ -red step of CP+ $\forall$ red, if  $u$  is the universal variable being reduced, then  $u$  need not be the rightmost variable with a non-zero coefficient. There may be universal variables to the right of  $u$  with non-zero coefficients. This is analogous to the conditions in QU-Res, where we require only that every existential variable  $x$  in  $C$  has  $\text{lv}(x) < \text{lv}(u)$ . However, in the Frege+ $\forall$ red proof system defined in [6], the variable being reduced from a formula is required to be the rightmost in the formula; that is,  $\text{ind}(x) < \text{ind}(u)$  for every variable other than  $x$  in  $C$ . We show below that imposing such a condition in CP+ $\forall$ red does not affect the strength of the proof system. That is, if we call a proof where the  $\forall$ -red steps are applied only to the rightmost universal variables with non-zero coefficients a **normal-form** proof, then any CP+ $\forall$ red proof can be efficiently converted to one in normal form. In later sections we often assume this normal form.

► **Lemma 5.** *Any CP+ $\forall$ red proof can be converted into normal form in polynomial time.*

**Proof Sketch.** To reduce a variable  $u$ , first reduce all universal variables to the right of  $u$ , then reduce  $u$ , then re-introduce the previously reduced variables using Boolean axioms. The constant on the right-hand-side may change along the way but finally reverts to its original value. ◀

Now we show that CP+ $\forall$ red is a complete and sound proof system for false QBFs.

► **Theorem 6.** *CP+ $\forall$ red is a complete and sound proof system for false QBFs. That is, if  $\varphi$  is a false QBF, then there exists a CP+ $\forall$ red refutation of  $\varphi$  (completeness), and if there exists a CP+ $\forall$ red refutation of  $\varphi$ , then  $\varphi$  is false (soundness).*

**Proof Sketch.** Completeness: We show that CP+ $\forall$ red p-simulates QU-Res; given a QU-Res proof  $\pi$ , for each  $C \in \pi$  we can derive  $R(C)$  in CP+ $\forall$ red. (The resolution rule is simulated by the CP part as in the classical case, and the  $\forall$ -Red rule of QU-Res is also present in CP+ $\forall$ red.) Since QU-Res is known to be complete, it follows that CP+ $\forall$ red is complete.

Soundness: Let  $\mathcal{F} = \mathcal{Q}.F$  be the standard encoding of  $\varphi$ , and let  $\pi = \mathcal{Q}.[I_1, I_2, \dots, I_l]$  be a normal form CP+ $\forall$ red refutation of  $\mathcal{F}$ . We show that the following is valid for each  $j \in [l]$ :  $\mathcal{Q}. [F \wedge I_1 \wedge \cdots \wedge I_{j-1}] \implies \mathcal{Q}. [F \wedge I_1 \wedge \cdots \wedge I_{j-1} \wedge I_j]$ . Thus if  $\mathcal{F} = \mathcal{Q}.F$  is true, then so is  $\mathcal{Q}. [F \wedge I_1 \wedge \cdots \wedge I_{l-1} \wedge I_l]$ . However,  $I_l$  is not satisfied by any assignment, so this statement is false. Hence  $\mathcal{F}$  is false, and by Proposition 3,  $\varphi$  is also false. ◀

Note that for false quantified inequalities, the soundness of CP+ $\forall$ red follows from the same proof, but completeness will require an additional argument.

Since we will refer to the p-simulation of QU-Res by CP+ $\forall$ red later, we state it as a separate lemma; the proof is in the completeness part of the proof of Theorem 6.

► **Lemma 7.** *CP+ $\forall$ red p-simulates QU-Res.*

## 4 Strategy extraction for CP+ $\forall$ red

*Strategy extraction* is an important paradigm in QBF, also very desirable in practice (cf. [26, 3, 22, 7]). Winning strategies for the universal player can be very complex. But a QBF proof system has the strategy extraction property for a particular class of circuits  $\mathcal{C}$  whenever we can efficiently extract, from every refutation  $\pi$  of a false QBF  $\varphi$ , a winning strategy for the universal player where the strategies for individual universal variables are computable in circuit class  $\mathcal{C}$ .

In this section we show how to extract, from a refutation in CP+ $\forall$ red, winning strategies computable by bounded depth circuits with threshold gates.

► **Theorem 8** (Strategy Extraction Theorem). *Given a false QBF  $\varphi = \mathcal{Q}. \phi$ , with  $n$  variables, and a CP+ $\forall$ red refutation  $\pi$  of  $\varphi$  of size  $m$ , it is possible to extract from  $\pi$  a winning strategy where for each universal variable  $u \in \varphi$ , the strategy  $\sigma_u$  can be computed by Boolean circuits of  $(m+n)^{O(1)}$  size, constant depth, with unbounded fanin AND, OR, NOT gates as well as threshold gates. In particular, if  $\varphi$  can be refuted in CP+ $\forall$ red in  $n^{O(1)}$  size, then the winning strategies can be computed in TC<sup>0</sup>.*

**Proof Sketch.** We adapt the technique from [6]. Let  $\mathcal{Q}. F$  be the standard encoding of  $\varphi$ , and let  $\pi = \mathcal{Q}. [I_1, \dots, I_l]$  be a normal-form CP+ $\forall$ red proof of  $\mathcal{Q}. F$  of length  $l$  and size  $m \geq l$ . For  $j \in \{0, 1, \dots, l\}$ , define  $\pi_j = \mathcal{Q}. [I_{j+1}, \dots, I_l]$  and  $F_j = F \cup \{I_1, \dots, I_j\}$ . By downward induction on  $j$ , from  $\pi_j$  we show how to compute, for each universal variable  $u$ , a Boolean function  $\sigma_u^j$  that maps each assignment to the variables quantified before  $u$  to a bit  $\{0, 1\}$ . These functions satisfy the property that in a 2-player game played on the formula  $\mathcal{Q}. F_j$ , if the universal player uses strategy  $\sigma_u^j$  for each universal variable  $u$ , then finally some inequality in  $F_j$  is falsified. We describe the functions  $\sigma_u^j$  by decision lists of size  $O(l)$ , where each condition is checkable by a constant-depth polynomial-in- $m$  sized threshold circuit.

Since all axioms are included in  $F$ , we can skip the axiom steps in the proof.

The strategy is as follows:  $\sigma_u^l = 0$  for all  $u$ . For  $j \leq l$ , if  $I_j$  is obtained by a classical rule, then  $\sigma_u^{j-1} = \sigma_u^j$  for every universal variable  $u$ . If  $I_j$  is derived using a  $\forall$ -red rule; that is  $I_j = I_k|_{u=b_j}$  for some  $k < j$ , then for all  $u' \neq u$ ,  $\sigma_{u'}^{j-1} = \sigma_{u'}^j$ . For  $u$ , if  $I_k|_{u=b_j}(\vec{a}) = 0$ , then  $\sigma_u^{j-1}(\vec{a}) = b_j$ , else  $\sigma_u^{j-1}(\vec{a}) = \sigma_u^j(\vec{a})$ . (The value  $I_k|_{u=b_j}(\vec{a})$  can be determined since variables to the right of  $u$  have zero coefficient in  $I_k$ .) It is easy to see that these functions so defined have the desired property. ◀

Theorem 8 yields the following conditional lower bound for CP+ $\forall$ red proof size.

► **Corollary 9.** *If PSPACE/poly  $\not\subseteq$  TC<sup>0</sup>, then there exists a family of false QBFs  $Q_{qbf} f_n$  that requires super-polynomial size proofs in CP+ $\forall$ red.*

**Proof.** Let  $f_n \in \text{PSPACE/poly} \setminus \text{TC}^0$ . Consider the following false sentence based on  $f_n$ :

$$\exists x_1 \dots x_n \forall z. [f(\vec{x}) \neq z].$$

Since  $f_n$  is in PSPACE/poly and QBF is PSPACE-complete, the value of  $f_n$  can be compactly expressed by a QBF. That is,  $f_n(\vec{x}) \equiv \mathcal{Q}_1 y_1 \dots \mathcal{Q}_r y_r. \psi_n(\vec{x}, \vec{y})$  where  $r$  is polynomial in  $n$  and  $\psi_n(\vec{x}, \vec{y})$  is in P/poly. Thus we have the false sentence

$$\exists x_1 \dots x_n \forall z. \left[ \overbrace{(\mathcal{Q}_1 y_1 \dots \mathcal{Q}_r y_r. \psi_n(\vec{x}, \vec{y}))}^{f_n(\vec{x})} \leftrightarrow \neg z \right].$$



We now choose circuits  $C_n$  computing  $\psi_n$  and use additional variables  $\vec{s}$  and  $\vec{t}$  to represent the gate values in the P/poly circuits  $C_n$  and  $\neg C_n$ , respectively. We obtain the QBF

$$\exists x_1 \dots x_n \forall z \mathcal{Q}_1 y_1 \dots \mathcal{Q}_r y_r \bar{\mathcal{Q}}_1 w_1 \dots \bar{\mathcal{Q}}_r w_r \exists \vec{s}, \vec{t}. [(C_n(\vec{x}, \vec{y}, \vec{s}) \vee z) \wedge (\neg C_n(\vec{x}, \vec{w}, \vec{t}) \vee \neg z)]$$

where  $\bar{\mathcal{Q}} = \exists$  if  $\mathcal{Q} = \forall$  and vice versa. We call this formula  $Q_{qbf}\text{-}f_n$  and remark that it is a false prenex QBF with CNF matrix. ( $C_n$  can be expressed as a CNF; then adding the literal  $z$  to each clause expresses  $C_n \vee z$ . Similarly for  $\neg C_n \vee \neg z$ .)

In the two-player game on  $Q_{qbf}\text{-}f_n$  or on its standard encoding, the only winning strategy for the universal variable  $z$  is the function  $f_n(\vec{x})$  itself. Therefore if there exists a polynomial size CP+ $\forall$ red proof for  $Q_{qbf}\text{-}f_n$ , then from Theorem 8,  $f_n \in \text{TC}^0$ , a contradiction.  $\blacktriangleleft$

## 5 Feasible (monotone) interpolation for CP+ $\forall$ red

In this section we show that CP+ $\forall$ red admits feasible monotone interpolation. We adapt the technique first used by Pudlák [33] to re-prove and generalise the result of Krajíček [30].

Consider a false QBF of the form

$$\varphi = \exists \vec{p} \mathcal{Q} \vec{q} \bar{\mathcal{Q}} \vec{r}. [A'(\vec{p}, \vec{q}) \wedge B'(\vec{p}, \vec{r})]$$

where  $\vec{p}$ ,  $\vec{q}$ , and  $\vec{r}$  are mutually disjoint sets of propositional variables,  $A'(\vec{p}, \vec{q})$  is a set of clauses using only the  $\vec{p}$  and  $\vec{q}$  variables, and  $B'(\vec{p}, \vec{r})$  is a set of clauses using only the  $\vec{p}$  and  $\vec{r}$  variables. Thus  $\vec{p}$  are the common variables between them. The  $\vec{q}$  and  $\vec{r}$  variables can be quantified arbitrarily, with any number of quantification levels. Since  $\varphi$  is false, on any assignment  $\vec{a}$  to the variables in  $\vec{p}$ , either  $\varphi_{\vec{a},0} = \mathcal{Q} \vec{q}. A'(\vec{a}, \vec{q})$  or  $\varphi_{\vec{a},1} = \bar{\mathcal{Q}} \vec{r}. B'(\vec{a}, \vec{r})$  (or both) must be false. An interpolant for  $\varphi$  is a Boolean function that, given  $\vec{a}$ , indicates which of  $\varphi_{\vec{a},0}$ ,  $\varphi_{\vec{a},1}$  is false. As defined in [9], a QBF proof system  $S$  admits feasible interpolation if from an  $S$ -proof  $\pi$  of such a QBF  $\varphi$ , we can extract a Boolean circuit  $C_\pi$  computing an interpolant for  $\varphi$ , such that, the size of  $C_\pi$  is polynomially related to the size of  $\pi$ . If, whenever the  $\vec{p}$  variables occur only positively in  $A'$  or only negatively in  $B'$ , the polynomial sized (with respect to the size of  $\pi$ ) interpolating circuit for  $\varphi$  is monotone, then we say that  $S$  admits monotone feasible interpolation.

Cutting Planes naturally gives rise to arithmetic rather than Boolean circuits, as in the classical case in [33]. Generalising this to the case of QBFs, we have the following definitions.

► **Definition 10** ([33]). A monotone real circuit is a circuit which computes with real numbers and uses arbitrary non-decreasing real unary and binary functions as gates.

We say that a monotone real circuit computes a Boolean function (uniquely determined by the circuit), if for all inputs of 0's and 1's the circuit outputs 0 or 1.

► **Definition 11.** A QBF proof system  $S$  admits *monotone real feasible interpolation* if for any false QBF  $\varphi$  of the form  $\exists \vec{p} \mathcal{Q} \vec{q} \bar{\mathcal{Q}} \vec{r}. [A'(\vec{p}, \vec{q}) \wedge B'(\vec{p}, \vec{r})]$  where the  $\vec{p}$  variables occur only positively in  $A'$  or only negatively in  $B'$ , and for any  $S$ -proof  $\pi$  of  $\varphi$ , we can extract from  $\pi$  a monotone real circuit  $C$  of size polynomial in the length of  $\pi$  and the number  $n$  of  $\vec{p}$  variables, such that  $C$  computes a Boolean function, and on every 0,1 assignment  $\vec{a}$  for  $\vec{p}$ ,

$$\begin{aligned} C(\vec{a}) = 0 &\implies \mathcal{Q} \vec{q}. A'(\vec{a}, \vec{q}) \text{ is false, and} \\ C(\vec{a}) = 1 &\implies \bar{\mathcal{Q}} \vec{r}. B'(\vec{a}, \vec{r}) \text{ is false.} \end{aligned}$$

Such a  $C$  is called a monotone real interpolating circuit for  $\varphi$ .

We prove that the CP+ $\forall$ red proof system for false QBFs has this property:

► **Theorem 12.** *CP+ $\forall$ red for false QBFs admits monotone real feasible interpolation.*

To prove this, we will actually prove a stronger theorem, about interpolants for all false quantified sets of inequalities (not just those arising from false QBFs).

► **Theorem 13.** *CP+ $\forall$ red for inequalities admits monotone real feasible interpolation. That is, let  $\mathcal{F}$  be any false quantified set of inequalities of the form  $\exists \vec{p} \mathcal{Q} \vec{q} \mathcal{Q} \vec{r}. [A(\vec{p}, \vec{q}) \wedge B(\vec{p}, \vec{r})]$  where  $A \cup B$  includes all Boolean axioms, and where the coefficients of  $\vec{p}$  are either all non-negative in  $A$  or are all non-positive in  $B$ . If  $\mathcal{F}$  has a CP+ $\forall$ red-proof  $\pi$ , of length  $l$ , then we can extract a monotone real circuit  $C$  of size polynomial in  $l$  and the number  $n$  of  $\vec{p}$  variables in  $\mathcal{F}$ , such that  $C$  computes a Boolean function, and on any 0,1 assignment  $\vec{a}$  to  $\vec{p}$ ,*

$$\begin{aligned} C(\vec{a}) = 0 &\implies \mathcal{Q} \vec{q}. A(\vec{a}, \vec{q}) \text{ is false, and} \\ C(\vec{a}) = 1 &\implies \mathcal{Q} \vec{r}. B(\vec{a}, \vec{r}) \text{ is false.} \end{aligned}$$

Such a  $C$  is called a monotone real interpolating circuit for  $\mathcal{F}$ .

**Proof Sketch.** Let  $\pi = \exists \vec{p} \mathcal{Q} \vec{q} \mathcal{Q} \vec{r}. [I_1, \dots, I_l]$  be a CP+ $\forall$ red refutation of  $\mathcal{F}$ . The idea, as in [33], is to associate with each inequality

$$I \equiv \sum_k e_k p_k + \sum_i f_i q_i + \sum_j g_j r_j \geq D$$

in  $\pi$ , two inequalities

$$I_0 \equiv \sum_i f_i q_i \geq D_0, \quad I_1 \equiv \sum_j g_j r_j \geq D_1$$

depending on the Boolean assignment  $\vec{a}$  to the  $\vec{p}$  variables, in such a way that

- $I_0$  and  $I_1$  together imply  $I|_{\vec{a}}$ . (It suffices to ensure  $D_0 + D_1 \geq D - \sum_k e_k a_k$ .)
- $I_0$  can be derived solely from the  $\mathcal{Q} \vec{q}. A(\vec{a}, \vec{q})$  part in CP+ $\forall$ red.
- $I_1$  can be derived solely from the  $\mathcal{Q} \vec{r}. B(\vec{a}, \vec{r})$  part in CP+ $\forall$ red.

Then the inequalities corresponding to the last step of the proof,  $I_l$ , are  $0 \geq D_0$  and  $0 \geq D_1$ , with  $D_0 + D_1 \geq 1$ . Hence  $D_0 > 0 \implies \mathcal{Q} \vec{q}. A(\vec{a}, \vec{q})$  is false, and  $D_1 > 0 \implies \mathcal{Q} \vec{r}. B(\vec{a}, \vec{r})$  is false. Note that we only need to compute one of the values  $D_0, D_1$  to identify a false part of  $\mathcal{F}$ . Furthermore, we will show that if all the coefficients  $e_k$  in  $B(\vec{p}, \vec{r})$  are non-positive, then  $D_1$  can be computed by a real monotone circuit of size  $O(nl)$ . If all the coefficients  $e_k$  in  $A(\vec{p}, \vec{q})$  are non-negative, then we will show that  $-D_0$  can be computed by a real monotone circuit of size  $O(nl)$ . (The inputs to the circuit are an assignment  $\vec{a}$  to the  $\vec{p}$  variables.) Applying the unary non-decreasing threshold function  $D_1 > 0?$  or  $-D_0 \geq 0?$  to its output will then give a monotone real interpolating circuit for  $\mathcal{F}$ . ◀

Using monotone interpolation (Theorem 12), we now prove an unconditional lower bound for the CP+ $\forall$ red proof system, which is based on the false clique-co-clique formulas from [9].

► **Definition 14.** Fix positive integers  $k, n$  with  $k \leq n$ . CLIQUECOCLIQUE $_{n,k}$  is the class of QBFs of the form  $\exists \vec{p} \mathcal{Q} \vec{q} \mathcal{Q} \vec{r}. [A_{n,k}(\vec{p}, \vec{q}) \wedge B_{n,k}(\vec{p}, \vec{r})]$  where

- $\vec{p}$  is the set of variables  $\{p_{uv} \mid 1 \leq u < v \leq n\}$ . An assignment to  $\vec{p}$  picks a set of edges, and thus an  $n$ -vertex graph that we denote  $G_{\vec{p}}$ .
- $\mathcal{Q} \vec{q}. A_{n,k}(\vec{p}, \vec{q})$  is a QBF expressing the property that  $G_{\vec{p}}$  has a clique of size  $k$ .
- $\mathcal{Q} \vec{r}. B_{n,k}(\vec{p}, \vec{r})$  is a QBF expressing the property that  $G_{\vec{p}}$  has no clique of size  $k$ .

Any QBF in  $\text{CLIQUECOCLIQUE}_{n,k}$  expresses the clique-co-clique principle (there is a graph both containing and not containing a  $k$ -clique) and is obviously false. In [9], a particular QBF  $\varphi_n \in \text{CLIQUECOCLIQUE}_{n,n/2}$  of size polynomial in  $n$  is described. It can be easily generalised to QBFs  $\varphi_{n,k} \in \text{CLIQUECOCLIQUE}_{n,k}$  of size polynomial in  $n$ .

Let  $\Phi_{n,k}$  be any QBF in  $\text{CLIQUECOCLIQUE}$ , and suppose that it has a  $\text{CP}+\forall\text{red}$  proof of length  $l$ . From Theorem 12, we obtain a monotone real circuit  $C$  of size  $O(l + n^2)$  computing a Boolean function, such that for every 0, 1 input vector  $\vec{a}$  of length  $\binom{n}{2}$  encoding a graph  $G$ ,  $C(\vec{a}) = 1 \iff G$  has a  $k$  clique.

In [33], Pudlák showed the following exponential lower bound on the size of real monotone circuits interpolating the famous “clique-color” encodings.

► **Theorem 15** ([33]). *Suppose that the inputs for a monotone real circuit  $C$  are 0, 1 vectors of length  $\binom{n}{2}$  encoding in the natural way graphs on an  $n$ -element set. Suppose that  $C$  outputs 1 on all cliques of size  $k$  and outputs 0 on all complete  $(k-1)$ -partite graphs, where  $k = \lfloor \frac{1}{8}(n/\log n)^{2/3} \rfloor$ . Then the size of the circuit is at least  $2^{\Omega((n/\log n)^{1/3})}$ .*

(In some earlier literature, clique-color has been referred to as clique-co-clique. However, this is misleading because the clique-color encoding is weaker than  $\Phi_{n,k}$  in the following sense. The clique-color encoding says that there exists a graph which has a  $k$ -clique and is complete  $(k-1)$ -partite (maximal  $(k-1)$ -colorable). A graph may neither have a  $k$ -clique nor be complete  $(k-1)$ -partite, so both parts of the clique-color formula may be false. Our clique-co-clique formulas, on the other hand, always have exactly one true part.)

Since complete  $(k-1)$ -partite graphs have no  $k$ -clique, the real monotone interpolating circuit  $C$  we obtain from a  $\text{CP}+\forall\text{red}$  proof of  $\Phi_{n,k}$  also satisfies the premise of Theorem 15. Hence,  $C$  must have size exponential in  $n$ . But  $C$ 's size is polynomially related to the length of the  $\text{CP}+\forall\text{red}$  proof of  $\Phi_{n,k}$ . We have thus obtained the following:

► **Corollary 16.** *For  $k = \lfloor \frac{1}{8}(n/\log n)^{2/3} \rfloor$ , any false QBF  $\Phi_{n,k} \in \text{CLIQUECOCLIQUE}_{n,k}$  requires proofs of length exponential in  $n$  in the  $\text{CP}+\forall\text{red}$  proof system. In particular, the QBF  $\varphi_{n,k}$  from Definition 14 requires proofs of length exponential in  $|\varphi_{n,k}|$  in  $\text{CP}+\forall\text{red}$ .*

## 6 Relative power of $\text{CP}+\forall\text{red}$ and other QBF proof systems

In this section we relate the power of  $\text{CP}+\forall\text{red}$  with other well known QBF proof systems.

► **Theorem 17.**  *$\text{CP}+\forall\text{red}$  is exponentially stronger than  $\text{Q-Res}$  and  $\text{QU-Res}$ .*

**Proof.** By Lemma 7,  $\text{CP}+\forall\text{red}$   $p$ -simulates  $\text{QU-Res}$  (and hence  $\text{Q-Res}$ ), and is thus at least as strong as them. From classical proof complexity we know that false CNF formulas based on the pigeonhole principle are easy for Cutting Planes proof system [19] but hard for resolution [27]. Therefore  $\text{CP}+\forall\text{red}$  is exponentially more powerful than any QBF proof system based on resolution ( $\text{Q-Res}$ ,  $\text{QU-Res}$ , etc.); these systems cannot simulate  $\text{CP}+\forall\text{red}$ . ◀

► **Remark.** Note that the separating QBFs have only existential quantification. However, there are also separating QBFs using universal quantifiers.

This means that  $\text{CP}+\forall\text{red}$  is stronger than the classical CDCL proof systems. However, as we show next, it is weaker than even the base system of expansion solving.

- **Theorem 18.**  *$\text{CP}+\forall\text{red}$  and  $\forall\text{Exp}+\text{Res}$  are incomparable unless  $\text{P/poly} = \text{TC}^0$ , i.e.,*
- $\forall\text{Exp}+\text{Res}$  cannot simulate  $\text{CP}+\forall\text{red}$ .
  - If  $\text{P/poly} \not\subseteq \text{TC}^0$  then  $\text{CP}+\forall\text{red}$  cannot simulate  $\forall\text{Exp}+\text{Res}$ .

**Proof.** In [28], Janota and Marques-Silva show that there exists a family of false QBFs which are hard for  $\forall\text{Exp}+\text{Res}$  but easy to refute in  $\text{Q-Res}$ . As  $\text{CP}+\forall\text{red}$   $p$ -simulates  $\text{Q-Res}$  (Lemma 7), we conclude that  $\forall\text{Exp}+\text{Res}$  cannot simulate  $\text{CP}+\forall\text{red}$ .

For the second claim, let  $f_n \in \text{P/poly} \setminus \text{TC}^0$  be computed by circuit family  $C_n$  of size  $l(n) \in n^{O(1)}$ . We use  $C_n$  to express the obviously false sentence  $\exists x_1 \cdots x_n \forall z. f(\vec{x}) \neq z$ . Associate a variable  $t_i$  with each gate  $g_i$  in  $C_n$ , and consider the QBF

$$Q\text{-}f_n \equiv \exists x_1 \cdots x_n \forall z \exists t_1 \cdots t_l. (t_l \neq z) \wedge \bigwedge_{i=1}^l (t_i \text{ is consistent with the inputs to gate } i).$$

The inner formula can be written as an  $O(l)$ -sized CNF, so  $Q\text{-}f_n$  has size  $n^{O(1)}$ . Note that  $Q\text{-}f_n$  has a single universal variable  $z$ , and the (only) winning strategy for the universal player is  $z = f(\vec{x})$ . If  $Q\text{-}f_n$  has a proof of size polynomial in  $n$ , then by Theorem 8, this strategy, and hence  $f_n$ , are in  $\text{TC}^0$ , a contradiction. On the other hand, from [8, Proposition 28], we know that the formula  $Q\text{-}f_n$  can be refuted in  $\forall\text{Exp}+\text{Res}$  in  $O(n+l)$  steps. (Here, expand on both polarities of the single universal variable  $z$ , creating two copies  $t_i^0$  and  $t_i^1$  of each variable  $t_i$ . Inductively derive that for each  $b \in \{0, 1\}$ ,  $t_i^b$  is consistent with the inputs to gate  $i$  with the same polarity  $b$ , and with the circuit inputs  $x_j$  which do not have any polarity. Hence derive  $t_l^0 = t_l^1$ . Since the clauses expressing  $t_l \neq z$  on expansion give the unit clauses  $\neg t_l^1$  and  $t_l^0$ , we obtain a contradiction.)  $\blacktriangleleft$

► **Theorem 19.** *Frege+ $\forall\text{red}$  is exponentially stronger than  $\text{CP}+\forall\text{red}$ : Frege+ $\forall\text{red}$   $p$ -simulates  $\text{CP}+\forall\text{red}$ , whereas  $\text{CP}+\forall\text{red}$  does not simulate Frege+ $\forall\text{red}$ .*

**Proof Sketch.** In the classical (propositional) setting, Cook, Coullard and Turán [19] first showed that Extended Frege  $p$ -simulates Cutting Planes. Then Goerdt [24] showed that even Frege  $p$ -simulates Cutting Planes. Using techniques from [15], [19], and [24], we show that the same simulation goes through with minor modifications for QBFs.

Since Frege is exponentially more powerful than Cutting Planes over propositional formulas (as witnessed by the clique-colour formulas [33], see also Section 5), the converse simulation fails, and  $\text{CP}+\forall\text{red}$  and Frege+ $\forall\text{red}$  are exponentially separated.  $\blacktriangleleft$

There are also separating examples with non-trivial universal quantifiers. In Section 5, we described a class of QBF formulas expressing the clique-co-clique principle. By Corollary 16, none of them have short proofs in  $\text{CP}+\forall\text{red}$ . We show that a particular member of this class (i.e., a particular way of encoding clique-co-clique) has short proofs in Frege+ $\forall\text{red}$ .

► **Theorem 20.** *There is a  $\Phi_{n,k} \in \text{CLIQUECOCLIQUE}_{n,k}$  of size polynomial in  $n$ , with a Frege+ $\forall\text{red}$  proof of size polynomial in  $n$ .*

## 7 Semantic cutting planes for QBFs

The classical Cutting Planes proof system can be extended to the semantic Cutting Planes proof system by allowing the following semantic inference rule: from inequalities  $I'$ ,  $I''$ , we can infer  $I$  in one step if every Boolean assignment satisfying both  $I'$  and  $I''$  also satisfies  $I$ . In [23], it is shown that semantic Cutting Planes is exponentially more powerful than Cutting Planes. We now augment the system semantic Cutting Planes with the  $\forall$ -reduction rule as defined for  $\text{CP}+\forall\text{red}$ , to obtain a QBF version denoted  $\text{semCP}+\forall\text{red}$ . In fact, in this system we need only two rules, semantic inference and  $\forall$ -reduction, since the addition, multiplication and division rules of Cutting Planes are also semantic inferences, and the Boolean axioms can be semantically inferred from any inequality.

It is clear that  $\text{semCP}+\forall\text{red}$  is sound and complete. However it is not possible to verify the semantic rule efficiently (unless  $P=NP$ ).

As in  $\text{CP}+\forall\text{red}$ , we call a  $\text{semCP}+\forall\text{red}$  proof  $\pi$  a normal-form proof if  $\forall\text{-red}$  is applied only to the rightmost universal variable. Since one can use Boolean axioms in  $\text{semCP}+\forall\text{red}$ ; Lemma 5 is valid in  $\text{semCP}+\forall\text{red}$  as well. That is one can convert any  $\text{semCP}+\forall\text{red}$  proof  $\pi$  into a normal form in polynomial time.

Clearly,  $\text{SemCP}+\forall\text{red}$  is at least as powerful as  $\text{CP}+\forall\text{red}$ . From classical proof complexity we know that semantic Cutting Planes is exponentially more powerful than Cutting Planes [23]. That is, in [23, Theorem 2], it has been shown that for every  $n$ , there exists a CNF formula  $F_n$  which has a short semantic Cutting Planes refutation but needs  $2^{n^{\Omega(1)}}$  lines to refute in Cutting Planes. Thus  $\text{semCP}+\forall\text{red}$  is also exponentially more powerful than  $\text{CP}+\forall\text{red}$ , as witnessed by these purely existentially quantified formulas.

In Theorem 8, we established strategy extraction from  $\text{CP}+\forall\text{red}$  proofs. These results hold for  $\text{semCP}+\forall\text{red}$  proofs as well; if  $I_j$  is obtained by semantic inference, we do not change the strategy functions and let  $\sigma_u^{j-1} = \sigma_u^j$  for every universal variable  $u$ . Thus all the conditional lower bounds on  $\text{CP}+\forall\text{red}$  (Corollary 9, Theorem 18) continue to hold:

► **Corollary 21.**

1. If  $\text{PSPACE} \not\subseteq \text{TC}^0$ , then for any  $f_n \in \text{PSPACE} \setminus \text{TC}^0$ , the false QBFs  $Q_{\text{qbf-}f_n}$  require super-polynomial size proofs in  $\text{semCP}+\forall\text{red}$ .
2. If  $\text{P/poly} \not\subseteq \text{TC}^0$ , then  $\text{semCP}+\forall\text{red}$  cannot simulate  $\forall\text{Exp}+\text{Res}$ . For any  $f_n \in \text{P/poly} \setminus \text{TC}^0$ , the false QBFs  $Q\text{-}f_n$  require super-polynomial size proofs in  $\text{semCP}+\forall\text{red}$ .

For obtaining unconditional lower bounds, we need an analogue of real monotone interpolation (Theorems 12, 13). For this, we adapt the corresponding proof technique used in the classical case from [23]. Using their technique for semantic inference, and handling axioms and  $\forall$ -reduction rules as in the proof of Theorem 13, everything goes through as desired.

► **Theorem 22.** *SemCP+ $\forall$ red admits monotone real feasible interpolation for false QBFs.*

Using Theorem 22, we obtain an unconditional exponential lower bound for  $\text{semCP}+\forall\text{red}$ , analogous to Corollary 16.

► **Corollary 23.** *For  $k = \lfloor \frac{1}{8}(n/\log n)^{2/3} \rfloor$ , any false QBF  $\Phi_{n,k} \in \text{CLIQUECoCLIQUE}_{n,k}$  requires proofs of length exponential in  $n$  in the  $\text{semCP}+\forall\text{red}$  proof system. In particular, the QBFs  $\varphi_{n,k}$  from Definition 14 require proofs of length exponential in  $|\varphi_{n,k}|$  in  $\text{semCP}+\forall\text{red}$ .*

---

## References

- 1 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.
- 2 Albert Atserias and Sergi Oliva. Bounded-width QBF is PSPACE-complete. *J. Comput. Syst. Sci.*, 80(7):1415–1429, 2014. doi:10.1016/j.jcss.2014.04.014.
- 3 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1):45–65, 2012. doi:10.1007/s10703-012-0152-6.
- 4 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *SAT'14*, pages 154–169, 2014.
- 5 Marco Benedetti and Hratch Mangassarian. QBF-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.
- 6 Olaf Beyersdorff, Ilario Bonacina, and Leroy Chew. Lower bounds: From circuits to QBF proof systems. In *Proc. ACM Conference on Innovations in Theoretical Computer Science (ITCS'16)*, pages 249–260. ACM, 2016.

- 7 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. On unification of QBF resolution-based calculi. In *MFCS, II*, pages 81–93, 2014.
- 8 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. Proof complexity of resolution-based QBF calculi. In *Proc. Symposium on Theoretical Aspects of Computer Science (STACS'15)*, pages 76–89. LIPIcs, 2015.
- 9 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Feasible interpolation for QBF resolution calculi. In *Proc. International Colloquium on Automata, Languages, and Programming (ICALP'15)*, pages 180–192. Springer, 2015.
- 10 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Are short proofs narrow? QBF resolution is not simple. In *Proc. Symposium on Theoretical Aspects of Computer Science (STACS'16)*, 2016.
- 11 Olaf Beyersdorff, Leroy Chew, and Karteek Sreenivasaiah. A game characterisation of tree-like Q-resolution size. In *LATA*, pages 486–498. Springer, 2015.
- 12 Olaf Beyersdorff and Ján Pich. Understanding Gentzen and Frege systems for QBF. In *Proc. ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2016.
- 13 A. Blake. *Canonical expressions in boolean algebra*. PhD thesis, University of Chicago, 1937.
- 14 Maria Luisa Bonet, Carlos Domingo, Ricard Gavaldà, Alexis Maciel, and Toniann Pitassi. Non-automatizability of bounded-depth Frege proofs. *Computational Complexity*, 13(1–2):47–68, 2004.
- 15 Samuel R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *J. Symb. Log.*, 52(4):916–927, 1987.
- 16 Samuel R. Buss. Towards NP-P via proof complexity and search. *Ann. Pure Appl. Logic*, 163(7):906–917, 2012.
- 17 Václav Chvátal. Edmonds polytopes and weakly hamiltonian graphs. *Math. Program.*, 5(1):29–40, 1973.
- 18 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979.
- 19 William J. Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987.
- 20 Uwe Egly. On sequent systems and resolution for QBFs. In *Theory and Applications of Satisfiability Testing (SAT'12)*, pages 100–113, 2012.
- 21 Uwe Egly, Martin Kronegger, Florian Lonsing, and Andreas Pfandler. Conformant planning as a case study of incremental QBF solving. In *Artificial Intelligence and Symbolic Computation (AISC'14)*, pages 120–131, 2014.
- 22 Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *Logic for Programming, Artificial Intelligence, and Reasoning – 19th International Conference (LPAR)*, pages 291–308, 2013.
- 23 Yuval Filmus, Pavel Hrubes, and Massimo Lauria. Semantic versus syntactic cutting planes. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS, Orléans, France*, pages 35:1–35:13, 2016.
- 24 Andreas Goerdt. Cutting plane versus Frege proof systems. In *Computer Science Logic, 4th Workshop, CSL'90, Heidelberg, Germany, October 1-5, Proceedings*, pages 174–194, 1990.
- 25 Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.
- 26 Alexandra Goultiaeva, Allen Van Gelder, and Fahiem Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In *IJCAI*, pages 546–553, 2011.



- 27 Amin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.
- 28 Mikolás Janota and Joao Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015.
- 29 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.
- 30 Jan Krajíček. Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic. *The Journal of Symbolic Logic*, 62(2):457–486, 1997.
- 31 Jan Krajíček and Pavel Pudlák. Some consequences of cryptographical conjectures for  $S_2^1$  and  $EF$ . *Information and Computation*, 140(1):82–94, 1998.
- 32 Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- 33 Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.
- 34 Jussi Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *AAAI*, pages 1045–1050. AAAI Press, 2007.
- 35 Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.
- 36 John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.
- 37 Nathan Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(4):417–481, 2007.
- 38 Allen Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In *CP*, pages 647–663, 2012.
- 39 Heribert Vollmer. *Introduction to Circuit Complexity – A Uniform Approach*. Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 1999.





# Summaries for Context-Free Games\*

Lukáš Holík<sup>1</sup>, Roland Meyer<sup>2</sup>, and Sebastian Muskalla<sup>3</sup>

- 1 Brno University of Technology, Brno, Czech Republic  
holik@fit.vutbr.cz
- 2 TU Braunschweig, Germany; and  
Aalto University, Finland  
meyer@cs.uni-kl.de
- 3 TU Braunschweig, Germany  
muskalla@cs.uni-kl.de

---

## Abstract

We study two-player games played on the infinite graph of sentential forms induced by a context-free grammar (that comes with an ownership partitioning of the non-terminals). The winning condition is inclusion of the derived terminal word in the language of a finite automaton. Our contribution is a new algorithm to decide the winning player and to compute her strategy. It is based on a novel representation of all plays starting in a non-terminal. The representation uses the domain of Boolean formulas over the transition monoid of the target automaton. The elements of the monoid are essentially procedure summaries, and our approach can be seen as the first summary-based algorithm for the synthesis of recursive programs. We show that our algorithm has optimal (doubly exponential) time complexity, that it is compatible with recent antichain optimizations, and that it admits a lazy evaluation strategy. Our preliminary experiments indeed show encouraging results, indicating a speed up of three orders of magnitude over a competitor.

**1998 ACM Subject Classification** F.1.1 Models of Computation

**Keywords and phrases** summaries, context-free games, Kleene iteration, transition monoid, strategy synthesis

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.41

## 1 Introduction

The motivation of our work is to generalize the language-theoretic approach to verification of recursive programs [24, 29] to synthesis. Central to verification are queries  $\mathcal{L}(G) \subseteq \mathcal{L}(A)$ , where  $G$  is a context-free grammar representing the control-flow of a recursive program and  $A$  is a finite automaton representing the specification. When moving to synthesis, we replace the inclusion query by a strategy synthesis for an *inclusion game*. This means  $G$  comes with an ownership partitioning of the non-terminals. It induces a game arena defined by the sentential forms and the left-derivation relation (replace the leftmost non-terminal, corresponds to executing the recursive program). The winning condition is inclusion in a regular language given by a finite automaton  $A$ . To be precise, player *prover* tries to meet the inclusion by deriving terminal words from the language or enforcing infinite derivations. The goal of *refuter* is to disprove the inclusion by deriving a word outside  $\mathcal{L}(A)$ .

For the verification of recursive programs, the two major paradigms are *summarization* [37, 33] and *saturation* [9, 18]. Procedure summaries compute the effect of a procedure in

---

\* This work was partially supported by the Czech Science Foundation project 16-24707Y, the IT4IXS: IT4Innovations Excellence in Science project LQ1602, and the BUT project FIT-S-14-2486.



the form of an input-output relation. Saturation techniques compute the  $\text{pre}^*$ -image over the configurations of a pushdown system (including the stack). Both were extensively studied, optimized, and implemented [35, 41, 6, 7]. What speaks for summaries is that they seem to be used more often, as witnessed by the vast majority of verification tools participating in the software verification competition [6, 7]. The reason, besides simpler implementability, may be that the stack maintained by the  $\text{pre}^*$ -construction increases the search space.

Saturation has been lifted to games and synthesis in [13, 22], from which closest to our setting is the work of Cachat [13], where the game arena is defined by a pushdown system and the winning condition is given by a regular set of goal configurations, and the work of Muscholl, Schwentick, and Segoufin [30], where a problem similar to ours is solved by a reduction to [13]. In this paper<sup>1</sup>, we fill in the empty spot in the picture and propose a solver and synthesis method for context-free inclusion games based on summaries.

Problem \ Method	Saturation	Summarization
Verification	[9, 18]	[37, 33]
Synthesis	[13, 30, 22]	

**Overview of Our Method.** Our main contribution is a novel representation of inclusion games that combines well with efficient methods from algorithmic verification (see below). The basic data structure are the elements of the *transition monoid* of the automaton  $A$ , called boxes. Boxes are relations over the states of  $A$  that capture the state changes on  $A$  induced by terminal words [12]. As such, they correspond to procedure summaries. The set of all plays starting in a non-terminal yields a (typically infinite) tree. We show how to represent this tree by a (finite) *negation-free Boolean formula over the transition monoid*, where conjunction and disjunction represent the behavior of the players on the inner nodes.

To compute the representation, we employ a fixed-point iteration on a system of equations that reflects closely the rules of the grammar (and hence the shape of the tree). Indeed, we simultaneously compute the formulas for all non-terminals. In the fixed-point computation, a strategy of prover to enforce an infinite play naturally yields a formula equivalent to *false*. For the domain to be finite, we work modulo logical equivalence. The order is implication. Key to the fixed-point computations is the following compositionality: The formula describing the plays from a sentential form  $\alpha\beta$  can be obtained by appropriately composing the formulas for  $\alpha$  and  $\beta$ . Indeed, since we consider left-derivations, each play starting in  $\alpha\beta$  will have a prefix that coincides with a maximal play starting in  $\alpha$ , followed by a suffix that essentially is a play from  $\beta$ . Composition is monotonic wrt. implication.

Having a finite representation for the set of plays starting in each non-terminal has several applications. With compositionality, we can construct the formulas for all sentential forms. This allows us to decide whether a sentential form is in the winning region of a player: We compute the formula and check whether it is rejecting in the sense that refuter can enforce the derivation of a word rejected by the automaton. The latter amounts to evaluating the formula under the assignment that sets to *true* the rejecting boxes. When a sentential form is found to belong to the winning region of a player, we show how to compute a winning strategy, explained here for refuter. We transform the formula to conjunctive normal form (CNF). On CNFs, we define so-called *choice functions* that select a box from each clause. We define a strategy such that all conforming plays end in a terminal word represented by a

<sup>1</sup> The full version is available as technical report [25].

chosen box. Instantiating the strategy for a choice function that only picks rejecting boxes (always possible if the initial formula is rejecting) yields a winning strategy for refuter.

**Complexity and Efficiency.** We show that our algorithm is in 2EXPTIME, which is tight by [30]. Cachat’s algorithm is singly exponential and our input instances can be reduced to his with an exponential blow-up, which together also gives a doubly exponential procedure. The complexity of the reduction comes from that it must determinize the automaton  $A$  [30].

Our domain is compatible with algorithmic techniques that have proven efficient in a number of applications (see Section 8). We show how to adapt two heuristics to our fixed-point computation over formulas over boxes, namely antichains from [19, 3, 4] and lazy evaluation inspired by [17]. We also discuss the compatibility of our technique with recent algorithms for the analysis of well-structured systems. It is not immediate how to use the same heuristics for Cachat’s domain of automata. Moreover, the determinization within the reduction to Cachat’s method does not offer much opportunities for optimization, which means there is one level of exponential complexity that is hardly amenable to heuristics.

In preliminary experiments, we have compared an implementation of Cachat’s saturation-based algorithm with our new summary-based algorithm. The benchmarks were generated according to the Tabakov-Vardi random automata model [40] that we adapted to grammars. The running times of our algorithm were consistently better by three orders of magnitude (without the aforementioned optimizations). This supports our conjecture that keeping the stack has a negative impact on search procedures, and summaries should be preferable.

## 2 Inclusion Games on Context-Free Grammars

A context-free grammar (CFG) is a tuple  $G = (N, T, P)$ , where  $N$  is a finite set of non-terminals,  $T$  is a finite set of terminals with  $N \cap T = \emptyset$ , and  $P \subseteq N \times \vartheta$  is a finite set of production rules. Here,  $\vartheta = (N \cup T)^*$  denotes the set of sentential forms. We write  $X \rightarrow \eta$  if  $(X, \eta) \in P$ . We assume that every non-terminal is the left-hand side of some rule. The *left-derivation relation*  $\Rightarrow_L$  replaces the leftmost non-terminal  $X$  in  $\alpha$  by the right-hand side of a rule. Formally,  $\alpha \Rightarrow_L \beta$  if  $\alpha = wX\gamma$  with  $w \in T^*$ ,  $\beta = w\eta\gamma$ , and there is a rule  $X \rightarrow \eta \in P$ . We use  $w$  to refer to terminal words (so that a following non-terminal is understood to be leftmost). We consider CFGs that come with an *ownership partitioning*  $N = N_\circ \cup N_\square$  of the set of non-terminals. We say that the non-terminals in  $N_\star$  are owned by player  $\star \in \{\circ, \square\}$ . The ownership partitioning is lifted to the sentential forms ( $\vartheta = \vartheta_\circ \cup \vartheta_\square$ ) as follows:  $\alpha \in \vartheta_\square$  if the leftmost non-terminal in  $\alpha$  is owned by  $\square$ , and  $\vartheta_\circ = \vartheta \setminus \vartheta_\square$ . In particular,  $\circ$  owns all terminal words. Combined with the left-derivation relation, this yields a game arena.

► **Definition 1.** Let  $G = (N_\circ \cup N_\square, T, P)$  be a CFG with ownership partitioning. The *arena induced by  $G$*  is the directed graph  $(\vartheta_\circ \cup \vartheta_\square, \Rightarrow_L)$ .

A *play*  $p = p_0 p_1 \dots$  is a finite or infinite path in the arena. Being a path means  $p_i \Rightarrow_L p_{i+1}$  for all positions. If it is finite, the path ends in a vertex denoted  $p_{last} \in \vartheta$ . A path corresponds to a sequence of left-derivations, where for each leftmost non-terminal the owning player selects the rule that should be applied. A play is *maximal* if it has infinite length or if the last position is a terminal word.

The winning condition of the game is defined by inclusion or non-inclusion in a regular language (depending on who is the player) for the terminal words derived in maximal plays. If the maximal play is infinite, it does not derive a terminal word and satisfies inclusion. The regular language is given by a (non-deterministic) finite automaton  $A = (T, Q, q_0, Q_F, \rightarrow)$ .

Here,  $T$  is a finite alphabet,  $Q$  is a finite set of states,  $q_0 \in Q$  is the initial state,  $Q_F \subseteq Q$  is the set of final states, and  $\rightarrow \subseteq Q \times T \times Q$  is the transition relation. Instead of  $(q, a, q') \in \rightarrow$ , we write  $q \xrightarrow{a} q'$  and extend the relation to words:  $q \xrightarrow{w} q'$  means there is a sequence of states starting in  $q$  and ending in  $q'$  labeled by  $w$ . The language  $\mathcal{L}(A)$  consists of all words  $w \in T^*$  with  $q_0 \xrightarrow{w} q_f$  for some  $q_f \in Q_F$ . We write  $\overline{\mathcal{L}(A)} = T^* \setminus \mathcal{L}(A)$  for the complement language.

From now on, we use  $A = (T, Q, q_0, Q_F, \rightarrow)$  for finite automata and  $G = (N_\circ \cup N_\square, T, P)$  for grammars with ownership. Note that both use the terminal symbols  $T$ .

► **Definition 2.** The *inclusion game* and the *non-inclusion game* wrt.  $A$  on the arena induced by  $G$  are defined by the following winning conditions. A maximal play  $p$  satisfies the *inclusion winning condition* if it is either infinite or we have  $p_{last} \in \overline{\mathcal{L}(A)}$ . A maximal play satisfies the *non-inclusion winning condition* if it is finite and  $p_{last} \in \mathcal{L}(A)$ .

The two games are complementary: For every maximal play, exactly one of the winning conditions is satisfied. We will fix player  $\circ$  as the *refuter*, the player wanting plays to satisfy non-inclusion, which is a reachability condition. The opponent  $\square$  is the *prover*, wanting plays to satisfy inclusion, which is a safety condition. Since refuter has a single goal to achieve and has to enforce termination, we will always explain our constructions from refuter's point of view. To win, prover just has to ensure that she stays in her winning region. She does not need to care about termination.

A *strategy* for player  $\star \in \{\circ, \square\}$  is a function that takes a non-maximal play  $p$  with  $p_{last} \in \vartheta_\star$  (it is  $\star$ 's turn) and returns a successor of this last position. A play *conforms* to a strategy if whenever it is the turn of  $\star$ , her next move coincides with the position returned by the strategy. A strategy is *winning from a position*  $p_0$  if every play starting in  $p_0$  that is conform to the strategy eventually satisfies the winning condition of the game. The *winning region* for a player is the set of all positions from which the player has a winning strategy.

► **Example 3.** Consider the grammar  $G_{ex} = (\{X, Y\}, \{a, b\}, \{X \rightarrow aY, X \rightarrow \varepsilon, Y \rightarrow bX\})$ . The automaton  $A_{ex}$  is given in Figure 1 and accepts  $(ab)^*$ . If refuter owns  $X$  and prover owns  $Y$ , then prover has a winning strategy for the inclusion game from position  $X$ . Indeed, finite plays only derive words in  $(ab)^*$ . Moreover, if refuter enforces an infinite derivation, prover wins inclusion as no terminal word is being derived. Refuter can win non-inclusion starting from  $Y$ . After prover has chosen  $Y \rightarrow bX$ , refuter selects  $X \rightarrow \varepsilon$  to derive  $b \notin (ab)^*$ .

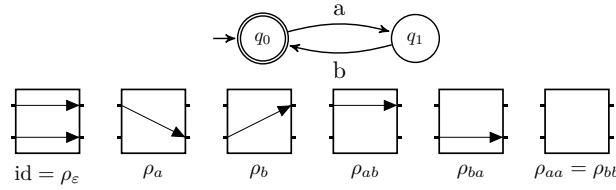
Our contribution is an algorithm to compute (a representation of) both, the winning region of the non-inclusion game for  $\circ$  and the winning region of the inclusion game for  $\square$ .

### 3 From Inclusion Games to Fixed Points

We give a summary-based representation of the set of all plays from each non-terminal and a fixed-point analysis to compute it. We lift the information to the sentential forms.

**Domain.** The idea of the analysis domain is to use Boolean formulas over words. To obtain a finite set of propositions, we consider words equivalent that induce the same state changes on  $A$ , denoted by  $\sim_A$ . The winning condition is insensitive to the choice of  $\sim_A$ -equivalent words. This means it is sufficient to take formulas over  $\sim_A$ -equivalence classes.

To finitely represent the  $\sim_A$ -equivalence classes, we rely on the *transition monoid* of  $A$ , defined as  $M_A = (\mathcal{P}(Q \times Q), ;, \text{id})$ . We refer to the elements  $\rho, \tau \in M_A$  as *boxes*. Since boxes are relations over the states of  $A$ , their relational composition is defined as usual,



■ **Figure 1** The automaton  $A_{ex}$  accepting  $(ab)^*$  and all its boxes with non-empty language. The first dash on each side of a box represents state  $q_0$ , the second dash represents  $q_1$ .

$\rho; \tau = \{(q, q'') \mid \exists q' \in Q : (q, q') \in \rho \text{ and } (q', q'') \in \tau\}$ . Relational composition is associative. The identity box  $\text{id} = \{(q, q) \mid q \in Q\}$  is the neutral element wrt. relational composition.

A box  $\rho$  represents the language  $\mathcal{L}(\rho) = \{w \in \mathcal{L}(\rho) \mid \forall q, q' \in Q : q \xrightarrow{w} q' \text{ iff } (q, q') \in \rho\}$ . That is, the words induce the state changes specified by the box. Hence,  $\mathcal{L}(\rho)$  is an equivalence class of  $\sim_A$ , finitely represented by  $\rho$ . The function  $\rho_- : T^* \rightarrow M_A$  maps  $w$  to the unique box  $\rho_w$  representing the word,  $w \in \mathcal{L}(\rho_w)$ . More explicitly,  $\rho_\varepsilon = \text{id}$ ,  $\rho_a = \{(q, q') \mid q \xrightarrow{a} q'\}$  for all  $a \in T$ , and  $\rho_{uv} = \rho_u; \rho_v$ . The image  $\rho_{T^*}$  contains exactly the boxes  $\rho$  with  $\mathcal{L}(\rho) \neq \emptyset$ . Figure 1 illustrates the representation of words as boxes.

The terminal words generated by maximal plays are represented by boxes, disjunction gives the alternatives of refuter, and conjunction expresses the options for prover. The set of plays from a given position is thus represented by a formula  $F$  from the set  $\text{BF}_A$  of *negation-free Boolean formulas over the transition monoid* (propositions are boxes). This set includes the unsatisfiable formula *false*. We use the rules  $\text{false} \wedge F = F \wedge \text{false} = \text{false}$  and  $\text{false} \vee F = F \vee \text{false} = F$  to evaluate conjunctions and disjunctions involving *false* on the syntactical level. As a consequence, *false* is the only syntactic representation of the unsatisfiable formula. This will simplify the definition of relational composition. From now on and without further mentioning,  $F$  and  $G$  will refer to formulas from  $\text{BF}_A$ .

Our goal is to decide whether refuter can force the plays from an initial position to end in a terminal word rejected by  $A$ . To mimic this, we define a formula to be *rejecting* if it is satisfied under the assignment  $\nu : M_A \rightarrow \{\text{true}, \text{false}\}$  such that  $\nu(\rho) = \text{true}$  if and only if  $\rho$  does not contain a pair  $(q_0, q_f)$  with  $q_f \in Q_F$ .

To use formulas in a Kleene iteration, we have to define a partial ordering on them. Intuitively,  $F$  should be smaller than  $G$  if  $G$  makes it easier for refuter to win. Taking the logical perspective, it is easier for refuter to win if  $F$  implies  $G$ . Implication on  $\text{BF}_A$  is not antisymmetric. To factor out the symmetries, we reason modulo logical equivalence,  $\text{BF}/\Leftrightarrow$ . Every formula is understood as a representative of the class of logically equivalent formulas of  $\text{BF}_A$ . Extending  $\Rightarrow$  to  $\text{BF}/\Leftrightarrow$  by comparing representatives then yields a partial order. The least element of the partial order is the equivalence class of *false*.

**Operations.** We combine formulas by conjunction, disjunction, and by an operation of relational composition that lifts  $;$  from the transition monoid to formulas over boxes. To explain the definition of relational composition, note that every finite maximal play from  $\alpha\beta$  proceeds in two phases. It starts with a maximal play turning  $\alpha$  into a terminal word, say  $w$ , followed by a play from  $w\beta$ . Since there are no more derivations for  $w$ , the play from  $w\beta$  coincides with a play from  $\beta$ , except that all sentential forms have a prefix  $w$ .

Let  $F$  and  $G$  represent all plays starting in  $\alpha$  and  $\beta$ , respectively. In  $F$ , terminal words like  $w$  are represented by boxes  $\rho$ . We append the plays from  $\beta$  by replacing  $\rho$  with  $\rho; G$ . To take into account all plays from  $\alpha$ , we do this replacement for all boxes in  $F$ . It remains to

add the prefix  $w$  to the sentential forms in the plays from  $\beta$ . In  $\rho; G$ , every box  $\tau$  in  $G$  is replaced by  $\rho; \tau$ . The so-defined formula  $F; G$  will represent all plays from  $\alpha\beta$ .

► **Example 4.** Let  $F = \rho_a \vee \rho_b$  and  $G = \rho_c \wedge \rho_d$ . We have  $(\rho_a \vee \rho_b); (\rho_c \wedge \rho_d) = \rho_a; (\rho_c \wedge \rho_d) \vee \rho_b; (\rho_c \wedge \rho_d) = (\rho_a; \rho_c \wedge \rho_a; \rho_d) \vee (\rho_b; \rho_c \wedge \rho_b; \rho_d)$ . The first equality replaces  $\rho_a$  and  $\rho_b$  by  $\rho_a; G$  and  $\rho_b; G$ , respectively. The second equality prefixes  $\rho_c$  and  $\rho_d$  in  $G$  by the corresponding box  $\rho_a$  or  $\rho_b$ .

► **Definition 5.** *Relational composition* over  $\text{BF}_A$  is defined by  $F; \text{false} = \text{false}; G = \text{false}$  and for composite formulas ( $\star \in \{\wedge, \vee\}$ ,  $\rho \in \text{M}_A$ ) by

$$(F_1 \star F_2); G = F_1; G \star F_2; G \quad \text{and} \quad \rho; (G_1 \star G_2) = \rho; G_1 \star \rho; G_2 .$$

Note that the composition of two non-*false* formulas is not *false*. Therefore, the result of a relational composition is *false* if and only if at least one of the arguments was *false*.

Relational composition equips the set of formulas with a monoid structure. In particular, relational composition is associative. For a fixed-point iteration, the operations also have to be monotonic wrt.  $\Rightarrow$ . For conjunction and disjunction, monotonicity obviously holds.

► **Lemma 6.** *If  $F \Rightarrow F'$  and  $G \Rightarrow G'$ , then  $F; G \Rightarrow F'; G'$ .*

We lift the three operations to  $\Leftrightarrow$ -equivalence classes by applying them to arbitrary representatives. Since implication is transitive, monotonicity of the operations ensures well-definedness. Moreover, the operations still behave monotonically on  $\text{BF}/\Leftrightarrow$ . From now on, we can thus identify formulas with the classes they represent.

**System of Equations.** We introduce one variable  $\Delta_X$  for each non-terminal  $X \in N$ . Terminals  $a \in T$  yield boxes, and we write  $\Delta_a$  for  $\rho_a$ . We lift the notation  $\Delta_-$  to sentential forms:  $\Delta_\varepsilon = \text{id}$  and  $\Delta_{\alpha\beta} = \Delta_\alpha; \Delta_\beta$ . This means concatenation in rules is replaced by relational composition. All rules for the same non-terminal are combined into one equation using disjunction or conjunction, depending on who is the owner of the non-terminal.

► **Definition 7.** The *system of equations (over  $\text{BF}/\Leftrightarrow$ ) induced by  $G$  and  $A$*  has one equation for each non-terminal  $X \in N_{\star}$  with  $\star \in \{\circlearrowleft, \square\}$ . If  $X \rightarrow \eta_1, \dots, X \rightarrow \eta_k$  are all rules with  $X$  as their left-hand side, the equation is  $\Delta_X = \Delta_{\eta_1} \wedge \dots \wedge \Delta_{\eta_k}$  if  $X \in N_{\square}$  and  $\Delta_X = \Delta_{\eta_1} \vee \dots \vee \Delta_{\eta_k}$  if  $X \in N_{\circlearrowleft}$ .

With Lemma 6, for each non-terminal  $X$  we can understand the right-hand side of the associated equation as a monotonic function  $f_X : (\text{BF}/\Leftrightarrow)^N \rightarrow \text{BF}/\Leftrightarrow$ . It takes as input a vector of formulas (one for each non-terminal) and computes a new formula for  $\Delta_X$ . We combine the functions for each non-terminal to a single function  $f : (\text{BF}/\Leftrightarrow)^N \rightarrow (\text{BF}/\Leftrightarrow)^N$ . It is monotonic on the product domain wrt. the product order  $\Rightarrow^N$ .

Since  $\text{BF}/\Leftrightarrow$  with  $\Rightarrow$  is a finite bottomed partial order, there is a unique least solution  $\sigma$  for the equation  $\Delta = f(\Delta)$ , namely  $\sigma = \bigsqcup_{i \in \mathbb{N}} f^i(\perp)$  [15]. The least element of the product domain is the vector with the  $\Leftrightarrow$ -equivalence class of *false* in every component. Note that the solution is computed by iteratively applying  $f$  until a fixed point is reached. This procedure terminates since the chain  $\perp \Rightarrow^N f(\perp) \Rightarrow^N f(f(\perp)) \Rightarrow^N \dots$  stabilizes on a finite domain.

The solution  $\sigma : N \rightarrow \text{BF}/\Leftrightarrow$  yields a value  $\sigma_X$  for each non-terminal  $X \in N$ . We lift the notation to sentential forms by  $\sigma_\varepsilon = \text{id}$ ,  $\sigma_a = \rho_a$  for all  $a \in T$ , and  $\sigma_{\alpha\beta} = \sigma_\alpha; \sigma_\beta$ . From now on,  $\sigma$  will always be the least solution to a system of equations. The system will be clear from the context (either  $G, A$  from the development or  $G_{ex}, A_{ex}$  from the running example).



► **Example 8.** For  $G_{ex}$  and  $A_{ex}$  from Example 3, the system of equations consists of  $\Delta_X = \Delta_a; \Delta_Y \vee \Delta_\varepsilon = \rho_a; \Delta_Y \vee \text{id}$  and  $\Delta_Y = \Delta_b; \Delta_X = \rho_b; \Delta_X$ . Its least solution is  $\sigma_X = \text{id} \vee \rho_{ab}$  and  $\sigma_Y = \rho_b$ . To terminate the iteration, use  $\rho_{bab} = \rho_b$ .

The main result in this section states that the fixed point  $\sigma_\alpha$  is equivalent to the tree  $\mathcal{T}_\alpha$  of all plays starting in  $\alpha$ . To be precise, we understand the tree as a (typically infinite) formula where inner nodes owned by refuter yield disjunctions, prover's nodes are conjunctions, and terminal words are boxes. For the proof (see [25]), we develop machinery for infinite formulas.

► **Theorem 9.**  $\mathcal{T}_\alpha \Leftrightarrow \sigma_\alpha$ .

## 4 Winning Regions and Strategy Synthesis

Define the set of sentential forms  $W^{\subseteq \mathcal{L}(A)} = \{\alpha \in \vartheta \mid \sigma_\alpha \text{ is not rejecting}\}$  and denote the complement by  $W^{\not\subseteq \mathcal{L}(A)} = \vartheta \setminus W^{\subseteq \mathcal{L}(A)} = \{\alpha \in \vartheta \mid \sigma_\alpha \text{ is rejecting}\}$ . Our goal is to prove the following result in a constructive way, by synthesizing strategies guided by the fixed-point solution to the system of equations.

► **Theorem 10.** *Inclusion games are determined:  $\vartheta = W^{\subseteq \mathcal{L}(A)} \sqcup W^{\not\subseteq \mathcal{L}(A)}$ , where  $W^{\subseteq \mathcal{L}(A)}$  is the winning region of prover and  $W^{\not\subseteq \mathcal{L}(A)}$  is the winning region of refuter.*

As a consequence, it is *decidable* whether a given sentential form  $\alpha$  is winning for a player: Compute the formula  $\sigma_\alpha$  and evaluate it under  $\nu$  to check whether it is rejecting.

It has been shown in [36] that for all games on pushdown systems with  $\omega$ -regular winning conditions, the winning regions are regular. Indeed, the winning region of the non-inclusion game can be accepted by a deterministic automaton. The set of equivalence classes of formulas forms its set of control states, the equivalence class of  $\text{id}$  is the initial state and rejecting formulas are final states. If the automaton in state  $F$  reads symbol  $x \in N \cup T$ , it switches to the state  $F; \sigma_x$ .

Representing sentential forms by formulas is too imprecise to do strategy synthesis. (In fact, the leftmost non-terminal is not even encoded in the formula.) Since relational composition is associative, we can represent the set of all sentential forms  $\alpha = wX\beta$  by a set of triples  $(\sigma_w, X, \sigma_\beta)$ , where  $\sigma_w$  and  $\sigma_\beta$  are taken from a finite set of formulas (up to logical equivalence) and  $X$  is a non-terminal from a finite set. This finite representation will be sufficient for the strategy synthesis. Our synthesis operates on normalized formulas in CNF.

**Conjunctive Normal Form.** A formula in CNF is a conjunction of clauses, each clause being a disjunction of boxes. We use set notation and write clauses as sets of boxes and formulas as sets of clauses. The set of *CNF-formulas* over  $M_A$  is thus  $\text{CNF}_A = \mathcal{P}(\mathcal{P}(M_A))$ . Identify  $\text{true} = \{\}$  and  $\text{false} = \{\{\}\}$ . In this section, all formulas will stem from  $\text{CNF}_A$ .

When computing a disjunction, we have to apply distributivity to obtain a CNF.

► **Lemma 11.**  $F \vee G \Leftrightarrow \{K \cup H \mid K \in F, H \in G\}$  and  $F \wedge G \Leftrightarrow F \cup G$ .

As the result of the relational composition  $F;G$  of CNF-formulas, we obtain a formula with three alternations between conjunction and disjunction. We apply distributivity to normalize  $F;G$ . Lemma 13 gives a closed-form representation of the result. To understand the idea, consider the composition of one clause with a CNF.

► **Example 12.** Consider  $F;G = (\rho_a \vee \rho_b); (\rho_c \wedge \rho_d) = (\rho_a; \rho_c \wedge \rho_a; \rho_d) \vee (\rho_b; \rho_c \wedge \rho_b; \rho_d)$ . Distributivity yields  $(\rho_a; \rho_c \vee \rho_b; \rho_c) \wedge (\rho_a; \rho_c \vee \rho_b; \rho_d) \wedge (\rho_a; \rho_d \vee \rho_b; \rho_c) \wedge (\rho_a; \rho_d \vee \rho_b; \rho_d)$ .

To turn  $F; G$  to CNF, we normalize the composition  $K; G$  for every clause  $K \in F$ . The formula  $K; G$  is an alternation of disjunction (not in the example), conjunction, and disjunction. Distributivity, when applied to the topmost two operations, selects for every box  $\rho \in K$  a clause  $H \in G$  to compose  $\rho$  with. This justifies the following set-theoretic characterization.

► **Lemma 13.**  $F; G \Leftrightarrow \bigcup_{K \in F} \bigcup_{z: K \rightarrow G} \left\{ \bigcup_{\rho \in K} \rho; z(\rho) \right\}$ .

**Inclusion (for Prover).** Prover wins on infinite plays, and therefore does not have to care about termination. This yields a simple positional winning strategy. We focus on the more complex case of refuter.

► **Theorem 14.** *The strategy  $s \subseteq \mathcal{L}(A)$  that applies a rule such that the formula for the resulting position is not rejecting (if possible) is a winning strategy for prover for the inclusion game from all positions in  $W \subseteq \mathcal{L}(A)$ .*

**Non-Inclusion (for Refuter).** A CNF-formula is rejecting iff for each clause chosen by prover, refuter can select a rejecting box in this clause. We formalize the selection process using the notion of choice functions. A *choice function* on  $F \in \text{CNF}_A$  is a function  $c : F \rightarrow M_A$  selecting a box from each clause,  $c(K) \in K$  for all  $K \in F$ . We show that there is a strategy for refuter to derive a terminal word from one of the chosen boxes. In particular, the strategy will only generate finite plays. Note that a choice function can only exist if  $F$  does not contain the empty clause. Otherwise, the formula is equivalent to *false*, and refuter cannot enforce termination of the derivation process.

We show that by appropriately selecting the moves of refuter, we can refine the choice function along each play, independent on the choices of prover. Given a choice function  $c$  on a CNF-formula  $F$ , a choice function  $c'$  on  $G$  *refines*  $c$  if  $\{c'(H) \mid H \in G\} \subseteq \{c(K) \mid K \in F\}$ , denoted by  $c'(G) \subseteq c(F)$ . Given equivalent CNF-formulas, a choice function on the one can be refined to a choice function on the other formula. Hence, we can deal with representative formulas in the following development.

► **Lemma 15.** *Consider  $F \Rightarrow G$ . For any choice function  $c$  on  $F$ , there is a choice function  $c'$  on  $G$  that refines it.*

The strategy for refuter has to enforce termination. To this end, we consider formulas obtained from Kleene approximants. When composing the formulas for the non-terminals to obtain a formula for a sentential form, we use an intermediary solution of the Kleene iteration instead of the fixed-point solution. Define a *sequence of levels*  $lvl$  associated to a sentential form  $\alpha$  to be a sequence of natural numbers of the same length as  $\alpha$ . The formula  $\sigma_\alpha^{lvl}$  corresponding to  $\alpha$  and  $lvl$  is defined by  $\sigma_a^i = \{\{\rho_a\}\}$  for all  $a \in T \cup \{\varepsilon\}$ ,  $\sigma_X^i$  the solution to  $X$  from the  $i^{\text{th}}$  Kleene iteration, and  $\sigma_{\alpha.\beta}^{lvl.lvl'} = \sigma_\alpha^{lvl}, \sigma_\beta^{lvl'}$ . A choice function for  $\alpha$  and  $lvl$  is a choice function on  $\sigma_\alpha^{lvl}$ . Note that  $\sigma_a^i$  is independent of  $i$  for terminals  $a$ . Moreover, there is an  $i_0$  so that  $\sigma_X^{i_0} = \sigma_X$  for all non-terminals  $X$ . This means a choice function on  $\sigma_\alpha$  can be understood as a choice function on  $\sigma_\alpha^{i_0}$ . Here, we use a single number  $i_0$  to represent a sequence  $lvl = i_0 \dots i_0$  of the appropriate length.

By definition,  $\sigma_X^0$  is *false* for all non-terminals, and *false* propagates through relational composition by definition. We combine this observation with the fact that choice functions do not exist on formulas that are equivalent to *false*.

► **Lemma 16.** *If there is a choice function for  $\alpha$  and  $lvl$ , then  $lvl$  does not assign zero to any non-terminal  $X$  in  $\alpha$ .*

The lemma has an important consequence. Consider a sentential form  $\alpha$  with an associated sequence  $lvl \in 0^*$  and a choice function  $c$  for  $\alpha$  and  $lvl$ . Then  $\alpha$  has to be a terminal word,  $\alpha = w \in T^*$ ,  $\sigma_\alpha^{lvl} = \{\{\rho_w\}\}$ , and the choice function has to select  $\rho_w$ . In particular,  $w$  itself forms a maximal play from this position on, and indeed the play ends in a word whose box is contained in the image of the choice function.

Consider now  $\alpha = wX\beta$  and  $lvl$  an associated sequence of levels. Assume  $lvl$  assigns a positive value to all non-terminals. Let  $j$  be the position of  $X$  in  $\alpha$  and let  $i = lvl_j$  be the corresponding entry of  $lvl$ . We split  $lvl = lvl'.i.lvl''$  into the prefix for  $w$ , the entry  $i$  for  $X$ , and the suffix for  $\beta$ . For each rule  $X \rightarrow \eta$ , we define  $lvl_\eta = lvl'.(i-1)\dots(i-1).lvl''$  to be the sequence associated to  $w\eta\beta$ . It coincides with  $lvl$  on  $w$  and  $\beta$  and has entry  $i-1$  for all symbols in  $\eta$ . Note that for a terminal word, the formula is independent of the associated level, so we have  $\sigma_{wX}^{lvl'.i} = \sigma_{wX}^i$  and  $\sigma_{w\eta}^{lvl'.(i-1)\dots(i-1)} = \sigma_{w\eta}^{i-1}$ .

We show that we can (1) always refine a choice function  $c$  on  $\sigma_\alpha^{lvl}$  along the moves of prover and (2) whenever it is refuter's turn, pick a specific move to refine  $c$ .

► **Lemma 17.** *Let  $c$  be a choice function for  $\alpha = wX\beta$  and  $lvl$ .*

- (1) *If  $X \in N_\square$ , for all  $X \rightarrow \eta$  there is a choice function  $c_\eta$  for  $w\eta\beta$  and  $lvl_\eta$  that refines  $c$ .*
- (2) *If  $X \in N_\circ$ , there is  $X \rightarrow \eta$  and a choice function  $c_\eta$  for  $w\eta\beta$  and  $lvl_\eta$  that refines  $c$ .*

**Proof.** We prove (2). We show that there is a rule  $X \rightarrow \eta$  and a choice function  $c_\eta$  on  $\sigma_{w\eta\beta}^{lvl_\eta}$  refining  $c$ . Towards a contradiction, assume this is not the case. Then for each rule  $X \rightarrow \eta$ , there is at least one clause  $K''_\eta$  of  $\sigma_{w\eta\beta}^{lvl_\eta}$  that does not contain a box in the image of  $c$ . By Lemma 13, this clause is defined by a clause  $\rho_w; K'_\eta$  of  $\sigma_{w\eta}^{i-1}$  and a function  $z_\eta$  mapping the boxes from this clause to  $\sigma_\beta^{lvl''}$ .

We have  $\sigma_X^i = \bigvee_{X \rightarrow \eta} \sigma_\eta^{i-1}$ . A clause of  $\sigma_{wX}^i$  is thus (Lemma 11) of the form  $K = \rho_w; (\bigcup_{X \rightarrow \eta} K_\eta) = \bigcup_{X \rightarrow \eta} \rho_w; K_\eta$ , where each  $K_\eta$  is a clause of  $\sigma_\eta^{i-1}$ . We construct the clause  $K' = \rho_w; (\bigcup_{X \rightarrow \eta} K'_\eta)$  of  $\sigma_{wX}^i$  using the  $K'_\eta$  from above. On this clause, we define the map  $z' = \bigcup_{X \rightarrow \eta} z_\eta$  that takes a box  $\rho_w; \rho \in \rho_w; K'_\eta$  and returns  $z_\eta(\rho_w; \rho)$ . (If a box  $\rho_w; \rho$  is contained in  $\rho_w; K'_\eta$  for several  $\eta$ , pick an arbitrary  $\eta$  among these.) By Lemma 13,  $K'$  and  $z'$  define a clause of  $\sigma_\alpha^{lvl}$ . The choice function  $c$  selects a box  $\rho_w; \rho; \tau$  out of this clause, where there is a rule  $X \rightarrow \eta$  such that  $\rho \in K'_\eta$  and  $\tau \in z'(\rho_w; \rho) = z_\eta(\rho_w; \rho)$ . This box is also contained in  $K''_\eta$ . A contradiction to the assumption that no box from  $K''_\eta$  is in the image of  $c$ . ◀

Notice that the sequence  $lvl_\eta$  is smaller than  $lvl$  in the following ordering  $\prec$  on  $\mathbb{N}^*$ . Given  $v, w \in \mathbb{N}^*$ , we define  $v \prec w$  if there are decompositions  $v = xyz$  and  $w = xiz$  so that  $i > 0$  is a positive number and  $y \in \mathbb{N}^*$  is a sequence of numbers that are all strictly smaller than  $i$ . Note that requiring  $i$  to be positive will prevent the sequence  $xz$  from being smaller than  $x0z$ , since we are not allowed to replace zeros by  $\varepsilon$ .

The next lemma states that  $\prec$  is well founded. Consequently, the number of derivations  $wX\beta \Rightarrow w\eta\beta$  following the strategy that refines an initial choice function will be finite.

► **Lemma 18.**  *$\prec$  on  $\mathbb{N}^*$  is well founded with minimal elements  $0^*$ .*

Lemma 18 is used in the main technical result of this section. Proposition 19 in particular says that all maximal plays that conform to  $s_{\alpha,c}$  are finite. If  $\sigma_\alpha$  is rejecting, there is a choice function on  $\sigma_\alpha$  that only selects rejecting boxes. The desired theorem is then immediate.

► **Proposition 19.** *Let  $c$  be a choice function on  $\sigma_\alpha$ . There is a strategy  $s_{\alpha,c}$  such that all maximal plays starting in  $\alpha$  that conform to  $s_{\alpha,c}$  end in a terminal word  $w$  with  $\rho_w \in c(\sigma_\alpha)$ .*

**Proof.** We show the following stronger claim: Given any triple consisting of a sentential form  $\alpha$ , an associated sequence of levels  $lvl$ , and a choice function  $c$  for  $\alpha$  and  $lvl$ , there is a strategy  $s_{\alpha,c}$  such that all maximal plays conform to it and starting in  $\alpha$  end in a terminal word  $w$  with  $\rho_w \in \{c(K) \mid K \in \sigma_\alpha\}$ . This proves the proposition by choosing  $\alpha$  and  $c$  as given and  $lvl = i_0 \dots i_0$ , where  $i_0 \in \mathbb{N}$  is a number such that  $\sigma = \sigma^{i_0}$ .

To show the claim, note that  $\prec$  on  $\mathbb{N}^*$  is well founded and the minimal elements are exactly  $0^*$  by Lemma 18, and  $lvl_\eta \prec lvl$ . This means we can combine Lemma 16 and Lemma 17 (for the step case) into a Noetherian induction. The latter lemma does not state that  $lvl_\eta$  assigns a positive value to each non-terminal, which was a requirement on  $lvl$ . This follows from Lemma 16 and the fact that  $c_\eta$  is a choice function. The strategy  $s_{\alpha,c}$  for refuter always selects the rule that affords a refinement of the initial choice function  $c$ . ◀

► **Theorem 20.** *Let  $\alpha \in W^{\mathcal{L}(A)}$  and let  $c$  select a rejecting box in each clause of  $\sigma_\alpha$ . Then  $s_{\alpha,c}$  is a winning strategy for refuter for the non-inclusion game played from  $\alpha$ .*

There are several ways of implementing a winning strategy for refuter. First, the strategy  $s_{\alpha,c}$  from Proposition 19 can be implemented using bounded memory, but one then needs linear time to decide which move to pick. Alternatively, it can be implemented using a pushdown, which then only needs constant time to pick moves. Second,  $s_{\alpha,c}$  is not positional, but one can obtain a position strategy by a breadth-first search in the tree of all plays from  $\alpha$ .

► **Example 21.** In the running example, formula  $\sigma_Y = \{\{\rho_b\}\}$  is rejecting. In fact, refuter can win the non-inclusion game played from  $Y$ . The initial choice function on  $\sigma_Y$  has to be  $c(\{\rho_b\}) = \rho_b$ . In the first step, prover has no alternative but  $Y \rightarrow bX$ . Position  $bX$  has the formula  $\sigma_{bX} = \{\{\rho_b\}\}; \{\{\text{id}, \rho_{ab}\}\} = \{\{\rho_b, \rho_{bab}\}\} = \{\{\rho_b\}\}$ . Pick the same choice function as before. The rule  $X \rightarrow \varepsilon$  causes  $\text{id}$  to enter  $\sigma_X$  in the first Kleene step. This causes  $\rho_b$  to enter  $\sigma_{bX}$  also in the first step. Indeed, by choosing  $X \rightarrow \varepsilon$  refuter wins non-inclusion.

## 5 Complexity

We show that deciding whether refuter has a winning strategy for non-inclusion from a given position is a 2EXPTIME-complete problem. Moreover, the algorithm presented in the previous sections achieves this optimal time complexity. Our proof of the lower bound follows the proof of the analogue result for the games considered in [30].

► **Theorem 22.** *Given a non-inclusion game and an initial position, deciding whether refuter has a winning strategy from the specified position is 2EXPTIME-hard.*

The following algorithm implements the fixed-point iteration discussed in Section 3, executed on formulas in CNF (see the paragraph on CNF of Section 4).

► **Algorithm 23.** Given a non-inclusion game and an initial position  $\alpha$ , the following algorithm computes whether refuter has a winning strategy from the given position.

1. Set  $\sigma_X^0 = \text{false}$  for all  $X \in N$ . Set  $i = 0$ .
2. Do until  $\sigma_X^i \Leftrightarrow \sigma_X^{i-1}$  for all  $X \in N$ :  $i = i + 1$ ;  $\sigma^i = f(\sigma^{i-1})$ .
3. Compute  $\sigma_\alpha$ , and return *true* iff  $\sigma_\alpha$  is rejecting.

Here,  $f$  is the function combining the right-hand sides of the equations as in Definition 7.

► **Theorem 24.** *Given a non-inclusion game and an initial position, Algorithm 23 computes whether refuter has a winning strategy from the given position in time  $\mathcal{O}\left(|G|^2 \cdot 2^{2^{|\mathcal{Q}|c_1}} + |\alpha| \cdot 2^{2^{|\mathcal{Q}|c_2}}\right)$  for some constants  $c_1, c_2 \in \mathbb{N}$ .*

The following corollary is an immediate consequence of Theorem 22 and Theorem 24.

► **Corollary 25.** *Deciding whether refuter has a winning strategy for a given non-inclusion game and an initial position is 2EXPTIME-complete.*

One should note that the running time of the algorithm is only exponential in the size of the automaton. If the automaton is assumed to be fixed, the running time of the algorithm is polynomial in the size of the grammar and in the length  $l$  of the initial position. Namely, it can be executed in  $\mathcal{O}(|G|^2 + l)$  steps.

Furthermore, the algorithm can solve games on the game arena induced by a grammar not only in the case of the non-inclusion winning condition, but also in a more general setting, e.g. if the winning condition is deriving a word in the regular target language after finitely many steps.

## 6 Experiments

We have implemented our algorithm in C++ [1] and compared it to an implementation of Cachat’s algorithm [13] for games on pushdown systems. Cachat’s input instances consist of a pushdown system  $P$  with an ownership partitioning on the control states and an alternating finite automaton over the stack alphabet of  $P$  ( $P$ -AFA). The first player wins by enforcing a run into a configuration accepted by the  $P$ -AFA. Cachat’s algorithm constructs the winning region of the first player by saturating the automaton.

To convert instances of our game to that of Cachat, we construct a pushdown system  $P$  that encodes both the grammar  $G$  and the target automaton  $A$ . A sentential form  $wX\beta$  (where  $X$  is the leftmost non-terminal) will be represented in  $P$  by a configuration  $(Q, X\beta)$ , where  $Q$  is the set of states  $A$  can be in after processing  $w$ . The translation thus embeds a determinized version of  $A$  in  $P$ . This may cause an exponential blow-up in the size of the input instance, which reflects the worst case complexity: Our problem is 2EXPTIME-complete while Cachat’s algorithm is exponential.

For the experiments, we generated random automata using the Tabakov-Vardi model [40]. The generator is parameterized in the number of letters and control states, the percentage of final states, and the number of transitions per letter (given as a fraction of the number of states). We adapt the model to generate also grammars with rules of the form  $X \rightarrow aYb$ , with parameters being the number of rules and non-terminals for each player, and the chances of  $a$ ,  $Y$ , and  $b$  to be present. Since sparse automata and grammars are likely to yield simpler instances, we focus on dense examples.

For the parameters  $|Q|$ ,  $|T|$ ,  $|N_{\circ}|$  and  $|N_{\square}|$ , we tried out several combinations. The entry  $x/y/z$  in the table below stands for  $|Q| = x$ ,  $|T| = y$ ,  $|N_{\circ}| = |N_{\square}| = z$ . For each combination, we generated 50 random automata and grammars, applied three algorithms to them, and measured how many instances could be solved within 10 seconds and how much time was consumed for the instances that could be solved on average.

We compared: (1) Our algorithm with a naive Kleene iteration, i.e. all components of the current solution are updated in each step. (2) Our algorithm with chaotic iteration implemented using a worklist, i.e. only components whose dependencies have been updated are modified. This is the common way of implementing a Kleene iteration. (3) Cachat’s algorithm applied to our problem as described above. To improve the runtime, the target automaton has been determinized and minimized before creating the pushdown system.

We ran our experiments on an Intel i7-6700K, 4GHz. The durations are milliseconds.

Already the naive implementation of Kleene iteration outperforms Cachat’s algorithm, which was not able to solve any instance with parameters greater than 10/15/15. The

worklist implementation is substantially faster, by three orders of magnitude on average. This confirms our hypothesis: The stack content is more information than needed for safety verification, and getting rid of it by moving to the summary domain speeds up the analysis.

One can also implement Cachat’s algorithm using a worklist, but due to the shape of the instances resulting from the reduction (deterministic automata), this is not helpful. In our experiments, the worklist variant was slower by at least one order of magnitude, even on small examples.

	naive Kleene		worklist Kleene		Cachat	
	avg	%t/o	avg	%t/o	avg	%t/o
5/ 5/ 5	65.2	2	0.8	0	94.7	0
5/ 5/10	5.4	4	7.4	0	701.7	0
5/10/ 5	13.9	0	0.3	0	375.7	0
5/ 5/15	6.0	0	1.1	0	1618.6	0
5/10/10	32.0	2	122.1	0	2214.4	0
5/15/ 5	44.5	0	0.2	0	620.7	0
5/ 5/20	3.4	0	1.4	0	3434.6	4
5/10/15	217.7	0	7.4	0	5263.0	16
10/ 5/ 5	8.8	2	0.6	0	2737.8	2
10/ 5/10	9.0	6	69.8	0	6484.9	66
15/ 5/ 5	30.7	0	0.2	0	5442.4	52
10/10/ 5	9.7	0	0.2	0	7702.1	92
10/15/15	252.3	0	1.9	0	n/a	100
10/15/20	12.9	0	1.8	0	n/a	100

## 7 Algorithmic Considerations

We discuss how to further speed-up the worklist implementation by two heuristics prominent in verification: Lazy evaluation [17] and antichains [19, 3, 4]. The heuristics are not meant to be a contribution of the paper and they are not yet implemented. The point is to demonstrate that the proposed summary domain combines well with algorithmic techniques. For both heuristics, it is not clear to us how to apply them to the domain of alternating automata.

The idea of *lazy evaluation* is to keep composed formulas  $(F \vee F')$ ;  $M$  symbolic, i.e. we store them as a term rather than computing the resulting formula. When having to evaluate the formula represented by the term, we only compute up to the point where the value influences the overall answer. Consider the test whether  $(F \vee F')$ ;  $M$  is rejecting. If already  $F$ ;  $M$  is rejecting, the whole formula represented by the term will be rejecting and the evaluation of  $F'$ ;  $M$  can be skipped.

The idea of the *antichain optimization* is to identify representative elements in the search space that allow us to draw conclusions about all other elements. Here, the search space consists of formulas (representing the intermediate steps of the fixed-point computation). By Lemma 6, it is sufficient to reason modulo logical equivalence. This allows us to remove redundant disjuncts and conjuncts, in particular, if  $F \Rightarrow G$  we can prune  $F$  from  $F \vee G$  and  $G$  from  $F \wedge G$ . When reasoning over CNFs, this removes from a formula all clauses that are subsumed by other clauses. It is thus enough to store the CNFs in the form of antichains of  $\subseteq$ -minimal clauses. The antichain approach benefits from a weaker notion of implication.

In Boolean satisfiability, the antichain optimization corresponds to the subsumption rule, and it is known to have a limited impact on the performance of solvers. The setting we



consider, however, is different. Our formulas are enriched during the computation by new clauses (that are not derived from others as in SAT). The antichain optimization can therefore be expected to yield better results for inclusion games and, in fact, has been successfully implemented for automata models [19, 3, 4].

## 8 Related Work

We already discussed the relation with Cachat's work [13]. Walukiewicz [42] studies games given by a pushdown automaton with a parity function on the states. Similar to our case, the derived strategies are implementable using a stack. The problem [42] is concerned with is different from ours in several respects. The game aspect is given by the specification (a  $\mu$ -calculus formula), not by the system as in our case. Moreover, (infinite) parity games are generally harder than safety: [42] is exponential both in the system and in the specification, while our construction is exponential only in the specification. Piterman and Vardi [31] study a similar variant of the problem and come up with a solution originating in the automata-theoretic approach [28].

Walukiewicz reduces solving parity games on the infinite computation tree of a pushdown system to solving parity games on a finite graph. To do so, instead of the full stack, only the topmost stack symbol is stored. Whenever a push should be executed, one player guesses the behavior of the game until the corresponding pop, i.e. she proposes a set of control states. The other player can decide to skip the subgame between push and pop by selecting a control state from the set, and the game continues. Alternatively, she can decide to verify the subgame. In this case, the new symbol becomes top-of-stack, and the game continues until it is popped. After the pop, the game ends, and which player wins is dependent on whether the current control state is in the proposed set of states.

This approach can be applied to a context-free game to reduce it to a reachability game on a doubly-exponentially-sized graph. Before applying a rule to the leftmost non-terminal  $X$ , we let refuter propose a set of boxes that describes the effect of terminal words derivable from  $X$ . Prover can either accept the proposal and select one of the boxes, or she verifies the proposal. In the latter case, the rest of the sentential form can be dropped. A winning strategy for refuter in the finite game has to guess the effect of each non-terminal, while our method deterministically computes it: The guessed effects that will not lead to refuter losing the subgame are exactly the sets of boxes occurring as the image of a choice function.

The work [30] considers active context-free games where in each turn, player A picks the position of a non-terminal in the current sentential form and player B picks the rule that is applied to the non-terminal. It is shown to be undecidable whether player A can enforce the derivation of a word in a regular language. If one limits the moves of player A to left-to-right strategies (skipped non-terminals cannot be touched again, the regular target language may contain non-terminals), one obtains a game that is closely related to our setting. In fact, the authors show that allowing player A to pick the rules for some of the non-terminals does not increase the expressive power. Therefore, there are polynomial-time reductions of our type of game to their type of game and vice versa. In [30], the focus lies on establishing the lower bounds for the time complexity of various type of active context-free games. The authors show that deciding the existence of a left-to-right winning strategy is 2EXPTIME-complete, like the problem considered in this paper (Section 5). The upper bound is shown by using an exponential-time reduction to Walukiewicz [42], and they also present an optimal algorithm that uses Cachat's algorithm for pushdown systems. Our algorithm also has optimal time complexity, but contrary to [30], it is based on procedure summaries



rather than on saturation. The lower bound is shown by encoding an alternating Turing machine with exponential space as a grammar game, and we adapted their proof to show Theorem 22 in [25]. [30] was further elaborated on and generalized in [8, 34].

Methods for solving variants of pushdown games, related mostly to saturation (see [14] for a survey on saturation-based methods), are implemented in several tools. [10] targets higher-order pushdown systems, related to it is [11], [39] implements an optimized saturation-based method, [23] solves the full case of parity games. [32] implements a type directed algorithm not based on saturation. None of the tools implements procedure summaries, but some can be used to solve instances of our problem. We plan to carry out a thorough comparison with these implementations in the future.

Antichain heuristics, discussed in Section 7, were developed in the context of finite automata and games [43, 44], and generalized to Büchi automata [19, 3, 4] with a fixed point over sets of boxes. Our lazy evaluation is inspired by [17]. Our framework is compatible with techniques for reachability in well-structured transition systems (WSTS) that proceed backwards [2]. We believe that techniques like [27, 20, 26, 5, 21] can be adapted to our setting. To instantiate general WSTS reachability algorithms, the ordering of configurations would be based on implication among formulas, the target set would be the upward closure of the assignment  $\sigma$  where  $\sigma_S$  is the conjunction of all rejecting boxes and  $\sigma_X = \text{false}$  for every other  $X \in N$ , and the initial state would be the assignment  $\perp$ . Another interesting possibility would be to adapt Newton iteration [16].

The transition monoid can be traced back at least to Büchi [12], and was prominently used e.g. in [38].

**Acknowledgments.** We thank Olivier Serre, Matthew Hague, Georg Zetsche, and Emanuele D’Osualdo for helpful discussions. We thank the reviewers for their feedback.

---

## References

- 1 Implementation of our algorithm. Published: 2016-17-07. URL: <https://concurrency.informatik.uni-kl.de/rigg.html>.
- 2 P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.
- 3 P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Simulation subsumption in Ramsey-based Büchi automata universality and inclusion testing. In *CAV*, volume 6174 of *LNCS*, pages 132–147. Springer, 2010.
- 4 P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Advanced Ramsey-based Büchi automata inclusion testing. In *CONCUR*, volume 6901 of *LNCS*, pages 187–202. Springer, 2011.
- 5 P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *VMCAI*, volume 7737 of *LNCS*, pages 476–495. Springer, 2013.
- 6 D. Beyer. Software verification and verifiable witnesses (report on sv-comp). In *TACAS*, volume 9035 of *LNCS*, pages 401–416. Springer, 2015.
- 7 D. Beyer. Reliable and reproducible competition results with benchexec and witnesses (report on sv-comp). In *TACAS*, volume 9636 of *LNCS*, pages 887–904. Springer, 2016.
- 8 H. Björklund, M. Schuster, T. Schwentick, and J. Kulbatzki. On optimum left-to-right strategies for active context-free games. In *ICDT*, pages 105–116. ACM, 2013.
- 9 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.

- 10 C. Broadbent, A. Carayol, M. Hague, and Olivier O. Serre. C-SHORE: A collapsible approach to higher-order verification. *ACM SIGPLAN Notices*, 48(9):13–24, 2013.
- 11 C. Broadbent and N. Kobayashi. Saturation-Based Model Checking of Higher-Order Recursion Schemes. In *CSL*, volume 23 of *LIPICs*, pages 129–148. Dagstuhl, 2013.
- 12 J. R. Büchi. *On a Decision Method in Restricted Second Order Arithmetic*, pages 425–435. Springer, 1990.
- 13 T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP*, volume 2380 of *LNCS*, pages 704–715. Springer, 2002.
- 14 A. Carayol and M. Hague. Saturation algorithms for model-checking pushdown systems. In *AFL*, volume 151 of *EPTCS*, pages 1–24, 2014.
- 15 B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. CUP, 1990.
- 16 J. Esparza, S. Kiefer, and M. Luttenberger. Newtonian program analysis. *JACM*, 57(6), 2010.
- 17 J. Fiedor, L. Holík, P. Jankøu, O. Lengál, and T. Vojnar. Lazy automata techniques for WS1S. Technical Report FIT-TR-2016-01, Brno University of Technology, 2016.
- 18 A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. *ENTCS*, 9:27–37, 1997.
- 19 S. Fogarty and M. Y. Vardi. Efficient Büchi universality checking. In *TACAS*, volume 6015 of *LNCS*, pages 205–220. Springer, 2010.
- 20 Z. Ganjei, A. Rezine, P. Eles, and Z. Peng. Lazy constrained monotonic abstraction. In *VMCAI*, volume 9583 of *LNCS*, pages 147–165. Springer, 2016.
- 21 P. Ganty, J.-F. Raskin, and L. Begin. A complete abstract interpretation framework for coverability properties of WSTS. In *VMCAI*, pages 49–64. Springer, 2006.
- 22 M. Hague and C.-H.L. Ong. Winning regions of pushdown parity games: A saturation method. In *CONCUR*, volume 5710 of *LNCS*, pages 384–398. Springer, 2009.
- 23 M. Hague and C.-H.L. Ong. Analysing mu-calculus properties of pushdown systems. In *SPIN*, volume 6349 of *LNCS*, pages 187–192. Springer, 2010.
- 24 M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.
- 25 L. Holík, R. Meyer, and S. Muskalla. Summaries for context-free games. *CoRR*, abs/1603.07256, 2016. URL: <http://arxiv.org/abs/1603.07256>.
- 26 A. Kaiser, D. Kroening, and T. Wahl. Efficient coverability analysis by proof minimization. In *CONCUR*, pages 500–515. Springer, 2012.
- 27 J. Kloos, R. Majumdar, F. Niksic, and R. Piskac. Incremental, inductive coverability. In *CAV*, volume 9206 of *LNCS*, pages 158–173. Springer, 2013.
- 28 O. Kupferman, N. Piterman, and M. Y. Vardi. An automata-theoretic approach to infinite-state systems. In *Time for Verification: Essays in Memory of Amir Pnueli*, volume 6200 of *LNCS*, pages 202–259. Springer, 2010.
- 29 Z. Long, G. Calin, R. Majumdar, and R. Meyer. Language-theoretic abstraction refinement. In *FASE*, volume 7212 of *LNCS*, pages 362–376. Springer, 2012.
- 30 A. Muscholl, T. Schwentick, and L. Segoufin. Active context-free games. *Theory of Computing Systems*, 39(1):237–276, 2005.
- 31 N. Piterman and M. Y. Vardi. Global model-checking of infinite-state systems. In *CAV*, volume 3114 of *LNCS*, pages 387–400. Springer, 2004.
- 32 S. J. Ramsay, R. P. Neatherway, and C.-H.L. Ong. A type-directed abstraction refinement approach to higher-order model checking. In *POPL*, pages 61–72. ACM, 2014.
- 33 T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL*, pages 49–61. ACM, 1995.
- 34 M. Schuster and T. Schwentick. Games for active XML revisited. In *ICDT*, volume 31 of *LIPICs*, pages 60–75. Dagstuhl, 2015.

- 35 S. Schwoon. *Model-Checking Pushdown Systems*. PhD thesis, TU Munich, 2002.
- 36 O. Serre. Note on winning positions on pushdown games with omega-regular conditions. *IPL*, 85(6):285–291, 2003.
- 37 M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. Technical Report 2, New York University, 1978.
- 38 A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. In *ICALP*, volume 194 of *LNCS*, pages 217–237. Springer, 1985.
- 39 D. Suwimonteerabuth, S. Schwoon, and J. Esparza. Efficient algorithms for alternating pushdown systems with an application to the computation of certificate chains. In *ATVA*, volume 4218 of *LNCS*, pages 141–153. Springer, 2006.
- 40 D. Tabakov and M. Y. Vardi. Experimental evaluation of classical automata constructions. In *LPAR*, volume 3835 of *LNCS*, pages 396–411. Springer, 2005.
- 41 WALi. Visited: 2016-16-07. URL: <https://research.cs.wisc.edu/wpis/wpds/download.php>.
- 42 I. Walukiewicz. Pushdown processes: Games and model-checking. *IC*, 164(2):234–263, 2001.
- 43 M. Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV*, volume 4144 of *LNCS*. Springer, 2006.
- 44 M. De Wulf, L. Doyen, and J.-F. Raskin. A lattice theory for solving games of imperfect information. In *HSCC*, volume 3927 of *LNCS*, pages 153–168. Springer, 2006.

# Admissibility in Quantitative Graph Games\*

Romain Brenguier<sup>1</sup>, Guillermo A. Pérez<sup>†2</sup>, Jean-François Raskin<sup>3</sup>,  
and Ocan Sankur<sup>4</sup>

1 University of Oxford, Oxford, UK

2 Université Libre de Bruxelles (ULB), Brussels, Belgium

3 Université Libre de Bruxelles (ULB), Brussels, Belgium

4 CNRS, Irisa, Rennes, France

---

## Abstract

*Admissibility* has been studied for games of infinite duration with Boolean objectives. We extend here this study to games of infinite duration with *quantitative* objectives. First, we show that, under the assumption that optimal worst-case and cooperative strategies exist, admissible strategies are guaranteed to exist. Second, we give a characterization of admissible strategies using the notion of adversarial and cooperative values of a history, and we characterize the set of outcomes that are compatible with admissible strategies. Finally, we show how these characterizations can be used to design algorithms to decide relevant verification and synthesis problems.

**1998 ACM Subject Classification** F.1.1 Automata, D.2.4 Formal methods

**Keywords and phrases** Quantitative games, Verification, Reactive synthesis, Admissibility

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.42

## 1 Introduction

Two-player *zero-sum* graph games are the most studied mathematical model to formalize the reactive synthesis problem [15, 16]. Unfortunately, this mathematical model is often an abstraction that is too coarse. Realistic systems are usually made up of several components, all of them with their *own* objectives. These objectives are not necessarily antagonistic. Hence, the setting of *non-zero sum* graph games is now investigated in order to *unleash* the full potential of automatic synthesis algorithms for reactive systems, see *e.g.* [9, 2, 5, 6, 14, 12].

For a player with objective  $\varphi$ , a strategy  $\sigma$  is said to be *dominated* by a strategy  $\sigma'$  if  $\sigma'$  does as well as  $\sigma$  with respect to  $\varphi$  against all the strategies of the other players and strictly better for some of them. A strategy  $\sigma$  is *admissible* for a player if it is *not* dominated by any other of his strategies. Clearly, playing a strategy which is not admissible is sub-optimal and a *rational* player should only play admissible strategies. The elimination of dominated strategies can be *iterated* if one assumes that each player knows the other players know that only admissible strategies are played, and so on.

While admissibility is a classical notion for finite games in normal form, see *e.g.* [13] and pointers therein, its generalization to infinite duration games is challenging and was only considered more recently. In 2007, Berwanger was the first to show [2] that admissibility, *i.e.* the avoidance of dominated strategies, is well-behaved in infinite duration  $n$ -player non-zero sum turn-based games with perfect information and Boolean outcomes (two possible payoffs:

---

\* This work was partially supported by the ERC Starting Grant inVEST (279499) and EPSRC grant EP/M023656/1.

† Author supported by F.R.S.-FNRS fellowship.



win or lose). This framework encompasses games with omega-regular objectives. The main contributions of Berwanger were to show that

- (i) in all  $n$ -player game structures, for all objectives, players have *admissible strategies*, (Berwanger even shows the existence of strategies that survive the iterated elimination of strategies)
- (ii) every strategy that is dominated by a strategy is dominated by an admissible strategy, and
- (iii) for finite game structures, the set of admissible strategies forms a regular set.

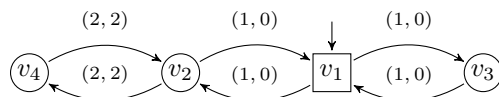
While the iterated admissibility formalizes a strong notion of rationality [1], it has been shown recently that the non-iterated version is strong enough to synthesize relevant strategies for non-zero sum games of infinite duration modelling reactive systems. In [11], Faella considers games played on finite graphs and focuses on the states from which one designated player cannot force a win. He compares several criteria for establishing what is the preferable behavior of this player from those states, eventually settling on the notion of admissible strategy. In [4], starting from the notion of admissible strategy, we have defined a novel rule for the compositional synthesis of reactive systems, applicable to systems made of  $n$  components which have each their own objective. We have shown that this synthesis rule leads to solutions which are robust and resilient.

Here, we study the notion of admissible strategy in infinite horizon  $n$ -player turn-based *quantitative* games played on a finite game structure. We give a comprehensive picture of the properties related to the existence of such strategies and to their characterization. Contrary to the Boolean case, the number of payoffs in our setting is potentially infinite making the characterization challenging. As in [2], we assume all players have perfect information.

**Main contributions.** First, contrary to the Boolean case, we show that in the quantitative setting, there are dominated strategies that are not dominated by any admissible strategy (Example 9). Second, we show that the existence of worst-case optimal and cooperatively optimal strategies for all players is a sufficient condition for the existence of admissible strategies (Thm. 4). Additionally, we show that there are games without worst-case optimal or without cooperative optimal strategies that do not have admissible strategies (Lem. 3). Third, we provide a characterization of admissible strategies in terms of antagonistic and cooperative values – that are classical values defined for quantitative games – (Thm. 11) and a characterization of the outcomes compatible with admissible strategies (Thm. 13). While the first characterization allows one to precisely describe admissible strategies, the characterization of the set of outcomes is given in linear temporal logic, and is a useful tool to reason about the outcomes that can be generated by such strategies. Finally, we show how to use the aforementioned characterizations to obtain algorithms to solve relevant decision problems for games with classical quantitative measures such as  $\text{Inf}$ ,  $\text{Sup}$ ,  $\text{LimInf}$ ,  $\text{LimSup}$  and mean-payoff (Thms. 17, 18, and 19).

**Example.** Let us consider the game from Fig. 1 to illustrate several notions and decision problems introduced and solved in this paper. The game is played by two players: Player 1, who owns the square vertices, and Player 2, owner of the round vertices. The measure that we consider here is the mean-payoff. (But note that, the arguments we will develop in this example are applicable to the limit inferior and limit superior measures as well.)

First, we note that the (best) worst-case value (or, the antagonistic value) that Player 1 can force is equal to 1, while the antagonistic value for Player 2 is equal to 0. The latter values are meaningful under the hypothesis that the other player is playing fully antagonistically



■ **Figure 1** Player 1 controls the square vertices, and Player 2 the round vertices. The payoff of Player  $i$  is the mean-payoff of the dimension  $i$  of the weights seen along the run.

and not pursuing their own objective. Now, if we account for the fact that Player 2 aims at maximizing his own payoff and so plays only admissible strategies towards this goal, then we conclude that he will never play the edge  $(v_2, v_1)$ . This is because, from vertex  $v_2$ , Player 2 has a strategy to enforce value 2 and taking edge  $(v_2, v_1)$  is unreasonable because, in the worst case, from  $v_1$  he will only obtain 0. As we show in Sec. 6, this kind of reasoning can be made formal and automated. We will show that, for games with classical quantitative measures, it can indeed be decided algorithmically if a finite memory strategy given, for instance, as a finite state Moore machine, is admissible or not.

Second, a similar but more subtle reasoning to the one presented above allows us to conclude that Player 1 will eventually play the edge  $(v_1, v_2)$ . Indeed, from vertex  $v_1$ , Player 1 can force a payoff equal to 1 by either taking edge  $(v_1, v_3)$  or  $(v_1, v_2)$ . Nevertheless, it is not reasonable for him to play edge  $(v_1, v_3)$  because, while this choice enforces a worst-case payoff equal to 1 (the antagonistic value), playing edge  $(v_1, v_2)$  is better because it ensures the same worst-case payoff and additionally leaves a possibility for Player 2 to help him by taking the cycle  $v_2-v_4$ , giving him a payoff of 2. If we take into account that the adversary is playing admissible strategies, then, in the words of [4], we can solve the assume-admissible synthesis problem. In this example, we conclude that Player 1 has a strategy to enforce a payoff of 2 against all admissible strategies of Player 2. A strategy which eventually chooses edge  $(v_1, v_2)$  ensures this payoff. The formalization of this reasoning and elements necessary for its automation are presented in Sec. 6.

**Structure of the paper.** Sec. 2 contains definitions. In Sec. 3, we study conditions under which the existence of admissible strategies is guaranteed. In Sec. 4, we give a characterization of admissible strategies, and in Sec. 5, a description of the set of outcomes compatible with admissible strategies. In Sec. 6, we apply our results to solve relevant decision problems on games with classical quantitative measures.

## 2 Preliminaries

We denote by  $\mathbb{R}$  the set of *real numbers*,  $\mathbb{Q}$  the set of *rational numbers*,  $\mathbb{N}$  the set of *natural numbers*, and  $\mathbb{N}_{>0}$  the set of *positive integers*.

A *game* is a tuple  $\mathcal{G} = \langle P, (V_i)_{i \in P}, v_{\text{init}}, E, (\text{payoff}_i)_{i \in P} \rangle$  where:

- (i)  $P$  is the non-empty and finite set of players.
- (ii)  $V \stackrel{\text{def}}{=} \bigsqcup_{i \in P} V_i$  where for every  $i \in P$ ,  $V_i$  is the finite set of player  $i$ 's vertices, and  $v_{\text{init}} \in V$  is the *initial vertex*.
- (iii)  $E \subseteq V \times V$  is the set of edges (it is assumed, w.l.o.g., that each vertex in  $V$  has at least one outgoing edge.)
- (iv) For every  $i$  in  $P$ ,  $\text{payoff}_i$  is a *payoff function* from infinite paths in the digraph  $\langle V, E \rangle$  to  $\mathbb{R}$  that, intuitively, player  $i$  will attempt to maximize.

An *outcome*  $\rho$  is an infinite path in the digraph  $\langle V, E \rangle$ , *i.e.* an infinite sequence of vertices  $(\rho_j)_{j \in \mathbb{N}_{>0}}$  such that  $(\rho_j, \rho_{j+1}) \in E$ , for all  $j \in \mathbb{N}_{>0}$ . A finite prefix of an outcome is called a

*history*. The length  $|h|$  of a history  $h = (\rho_j)_{1 \leq j \leq n}$  is  $n$ . Given an outcome  $\rho = (\rho_j)_{j \in \mathbb{N}_{>0}}$  and an integer  $k$ , we write  $\rho_{\leq k}$  for the history  $(\rho_j)_{1 \leq j \leq k}$ , that is, the prefix of length  $k$  of  $\rho$ . For a history  $h$  and a history or outcome  $\rho$ , we write  $h \subseteq_{\text{pref}} \rho$  if  $h$  is a prefix of  $\rho$ . If  $h \subseteq_{\text{pref}} \rho$ , we write  $h^{-1} \cdot \rho$  for the unique history (resp. outcome) that satisfies  $\rho = h \cdot (h^{-1} \cdot \rho)$ . The *first* (resp. *last*) vertex of a history  $h$  is  $\text{first}(h) = h_1$  (resp.  $\text{last}(h) \stackrel{\text{def}}{=} h_{|h|}$ ). The *longest common prefix* of two outcomes or histories  $\rho, \rho'$  is denoted  $\text{lcp}(\rho, \rho')$ . Given vertex  $v$  from  $\mathcal{G}$ , let us denote the set of *successors of  $v$*  by  $E_v \stackrel{\text{def}}{=} \{v' \in V \mid (v, v') \in E\}$ .

A *strategy* of player  $i$  is a function  $\sigma_i$  that maps any history  $h$  such that  $\text{last}(h) \in V_i$  to a vertex from  $E_{\text{last}(h)}$ . A *strategy profile* for the set of players  $P' \subseteq P$  is a tuple of strategies, one for each player of  $P'$ .

Let  $\Sigma_i(\mathcal{G})$  be the set of all strategies of player  $i$  in  $\mathcal{G}$ . We write  $\Sigma(\mathcal{G}) \stackrel{\text{def}}{=} \prod_{i \in P} \Sigma_i(\mathcal{G})$  for the set of all strategy profiles for  $P$  in  $\mathcal{G}$ , and  $\Sigma_{-i}(\mathcal{G})$  for the set of strategy profiles for all players but  $i$  in  $\mathcal{G}$ . We omit  $\mathcal{G}$  when it is clear from the context. Given  $\sigma_i \in \Sigma_i$  and  $\sigma_{-i} = (\sigma_j)_{j \in P \setminus \{i\}} \in \Sigma_{-i}$ , we write  $(\sigma_i, \sigma_{-i})$  for  $(\sigma_j)_{j \in P}$ .

A strategy profile  $\sigma_P \in \Sigma$  defines a unique *outcome* from any given history  $h$ . Formally,  $\mathbf{Out}_h(\mathcal{G}, \sigma_P)$  is the outcome  $\rho = (\rho_j)_{j \in \mathbb{N}_{>0}}$  such that  $\rho_{\leq |h|} = h$  and for  $j > |h|$ , if  $\rho_j \in V_i$ , then  $\rho_{j+1} = \sigma_i(\rho_{\leq j})$ . Notice that when  $h$  is a vertex, then this corresponds to starting the game at that vertex. When  $\mathcal{G}$  is clear from the context we shall omit it and write simply  $\mathbf{Out}_h(\sigma_P)$ . If  $S_i$  is a set of strategies for player  $i$ , we write  $\mathbf{Out}_h(S_i)$  for  $\{\rho \mid \exists \sigma_i \in S_i, \sigma_{-i} \in \Sigma_{-i} : \mathbf{Out}_h(\sigma_i, \sigma_{-i}) = \rho\}$ . Here,  $\mathbf{Out}_h(S_i)$  is the set of outcomes that are *compatible with  $S_i$* . All notations for outcomes are lifted to histories in the obvious way. For a strategy profile  $\sigma_P \in \Sigma$ , we write  $\mathbf{Hist}_h(\sigma_P)$  for the set  $\{\rho_{\leq j} \mid \rho \in \mathbf{Out}_h(\sigma_P), j \geq |h|\}$ .

Consider two strategies  $\sigma$  and  $\tau$  for player  $i$ , and a history  $h$ . We denote by  $\sigma[h \leftarrow \tau]$  the strategy that follows strategy  $\sigma$  and *shifts* to  $\tau$  at history  $h$ .

Formally, given a history  $h'$  such that  $\text{last}(h') \in V_i$ :

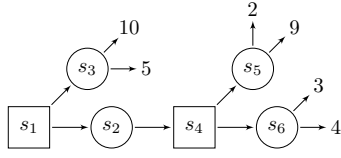
$$\sigma[h \leftarrow \tau](h') \stackrel{\text{def}}{=} \begin{cases} \tau(h^{-1} \cdot h') & \text{if } h \subseteq_{\text{pref}} h' \\ \sigma(h') & \text{otherwise;} \end{cases}$$

We now formally define dominance and admissibility. We recall the intuition: a player's strategy  $\sigma$  is dominated by another strategy  $\sigma'$  of his if  $\sigma'$  yields a payoff which is as good as that of  $\sigma$  against all strategies for the other players, and is strictly better against some of them. A strategy is admissible if no other strategy dominates it. More formally, we have:

**Dominance.** A strategy  $\sigma_i \in \Sigma_i$  *very weakly dominates* strategy  $\sigma'_i \in \Sigma_i$ , written  $\sigma_i \succcurlyeq \sigma'_i$ , if  $\forall \sigma_{-i} \in \Sigma_{-i}, \text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\sigma'_i, \sigma_{-i})) \leq \text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\sigma_i, \sigma_{-i}))$ . Strategy  $\sigma_i$  *weakly dominates* strategy  $\sigma'_i$ , written  $\sigma_i \succ \sigma'_i$ , if  $\sigma_i \succcurlyeq \sigma'_i$  and  $\neg(\sigma'_i \succcurlyeq \sigma_i)$ . A strategy  $\sigma \in \Sigma_i$  is weakly dominated if there exists  $\sigma' \in \Sigma_i$  such that  $\sigma' \succ \sigma$ . A strategy that is not weakly dominated is *admissible*. We denote by  $\mathfrak{A}_i(\mathcal{G})$  the set of all admissible strategies for player  $i$  in  $\mathcal{G}$ .

Our characterizations and algorithms are based on the notions of *cooperative* and *antagonistic values* of a history. The antagonistic value, denoted  $\mathbf{aVal}_i(\mathcal{G}, h)$ , is the maximum payoff that player  $i$  can secure from  $h$  in the worst case, *i.e.* against all strategies of other players. The cooperative value, denoted  $\mathbf{cVal}_i(\mathcal{G}, h)$ , is the best value player  $i$  can achieve from  $h$  with the help of other players. We also define a third type of value: the *antagonistic-cooperative value*, denoted  $\mathbf{acVal}_i(\mathcal{G}, h)$ , which is the maximum value player  $i$  can achieve in  $\mathcal{G}$  with the help of other players while guaranteeing the antagonistic value of the current history  $h$ . Formal definitions follow.





■ **Figure 2** Example game where local conditions fail to capture admissibility.



■ **Figure 3** Example game with an infinite dominance chain and no admissible strategy as witness of their being dominated.

**Antagonistic & Cooperative Values.** The *antagonistic value* of a strategy and the *cooperative value* of a strategy  $\sigma_i$  of player  $i$  in  $\mathcal{G}$ , for a history  $h$  are

$$\mathbf{aVal}_i(\mathcal{G}, h, \sigma_i) \stackrel{\text{def}}{=} \inf_{\tau \in \Sigma_{-i}} \text{payoff}_i(\text{Out}_h(\sigma_i, \tau));$$

$$\mathbf{cVal}_i(\mathcal{G}, h, \sigma_i) \stackrel{\text{def}}{=} \sup_{\tau \in \Sigma_{-i}} \text{payoff}_i(\text{Out}_h(\sigma_i, \tau)).$$

The *antagonistic value* of a history  $h$  for player  $i$ , and the *cooperative value* of a history  $h$  for player  $i$  are defined as  $\mathbf{aVal}_i(\mathcal{G}, h) \stackrel{\text{def}}{=} \sup_{\sigma_i \in \Sigma_i} \mathbf{aVal}_i(\mathcal{G}, h, \sigma_i)$ , and  $\mathbf{cVal}_i(\mathcal{G}, h) \stackrel{\text{def}}{=} \sup_{\sigma_i \in \Sigma_i} \mathbf{cVal}_i(\mathcal{G}, h, \sigma_i)$ , respectively. Finally, the *antagonistic-cooperative value* of a history  $h$  for player  $i$  is

$$\mathbf{acVal}_i(\mathcal{G}, h) \stackrel{\text{def}}{=} \sup\{\mathbf{cVal}_i(\mathcal{G}, h, \sigma_i) \mid \sigma_i \in \Sigma_i, \mathbf{aVal}_i(\mathcal{G}, h, \sigma_i) \geq \mathbf{aVal}_i(\mathcal{G}, h)\}.$$

We omit  $\mathcal{G}$  when it is clear from the context.

Observe that  $\mathbf{aVal}_i(h)$  of a history is the value of a zero-sum two-player game where player  $i$  is playing against players  $-i$ ; while  $\mathbf{cVal}_i(h)$  is the value in a one-player game, when all players play together.  $\mathbf{acVal}_i(h)$  is a new notion which is the supremum of the values player  $i$  can obtain when he plays *worst-case optimal strategies*. A strategy  $\sigma_i \in \Sigma_i$  is said to be *worst-case optimal* for player  $i$  at history  $h$  if  $\mathbf{aVal}_i(h, \sigma_i) = \mathbf{aVal}_i(h)$ ; it is said to be *cooperatively optimal* for him at history  $h$  if  $\mathbf{cVal}_i(h, \sigma_i) = \mathbf{cVal}_i(h)$ . Observe that  $\mathbf{acVal}_i(h) = -\infty$  if there are no worst-case optimal strategies from  $h$ .

► **Example 1** (Local conditions are not sufficient). The game in Fig. 2 shows that admissibility requires one to consider the values of the histories both in the past and in the future of the current history. This shows that a local condition cannot capture admissibility. In fact, consider strategy  $\sigma_1$  of player 1 (who controls all square vertices) that takes the edges  $(s_1, s_2)$ ,  $(s_4, s_6)$ . If the game starts at  $s_2$ ,  $\sigma_1$  is admissible, since the choice  $(s_4, s_5)$  could yield a payoff of 2 which is worse than any payoff from  $s_6$ . Indeed, we have that  $\mathbf{aVal}_1(s_5) < \mathbf{aVal}_1(s_6)$ . However, when the game starts at  $s_1$ ,  $\sigma_1$  is weakly dominated by the strategy that chooses  $(s_1, s_3)$  since the worst payoff in the latter case is 5. In fact, when a strategy takes the edge  $(s_1, s_2)$ , the antagonistic value decreases from  $\mathbf{aVal}_1(s_1) = 5$  to  $\mathbf{aVal}_1(s_2) = 3$ ; so to be admissible, it should have a better cooperative value than 5, which is not the case if  $(s_4, s_6)$  is taken. The strategy taking  $(s_1, s_2)$ ,  $(s_4, s_5)$  is admissible. Indeed, in one outcome, the payoff is 9, which is greater than 5 as required. Thus, an admissible strategy from  $s_1$  either goes to  $s_3$ , or goes to  $s_2$  but commits to taking  $(s_4, s_5)$  later.

We use temporal logic to describe sets of outcomes. We consider an extension of standard LTL with inequality conditions on payoffs for each player as in [3]. The logic, denoted  $\text{LTL}_{\text{payoff}}$ , extends LTL, and its syntax is defined as follows.

$$\varphi ::= Q \mid \neg\varphi \mid X\varphi \mid G\varphi \mid F\varphi \mid \varphi_1 \text{ U } \varphi_2 \mid \varphi_1 \text{ V } \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \text{payoff}_i \bowtie v,$$



■ **Figure 4** Two games in which Player 1 has no admissible strategy.

where  $Q \in \text{AP}$  is a set of atomic propositions on edges,  $\mathbf{G}$  and  $\mathbf{F}$  are the standard LTL modalities,  $\bowtie \in \{\leq, \geq, <, >\}$ , and  $v \in \mathbb{Q}$ . A formula is interpreted over an outcome  $\rho$  at index  $k$  as follows. We have, for instance,  $(\rho, k) \models Q$  if, and only if,  $(\rho_k, \rho_{k+1})$  is labelled with  $Q$ . For convenience, we write  $\rho \models \varphi$  instead of  $(\rho, 1) \models \varphi$ . Note that we define our predicates on edges rather than vertices; this simplifies our presentation. The semantics of the LTL modalities are standard; we refer to *e.g.* [3]. For payoff conditions, we have  $(\rho, k) \models \text{payoff}_i \bowtie v \stackrel{\text{def}}{\iff} \text{payoff}_i(\rho_{\geq k}) \bowtie v$ .

**Residual Games.** Given game  $\mathcal{G}$ , and history  $h$ , let us define  $\mathcal{G}_h$  as the *residual game of  $\mathcal{G}$  from  $h$*  by modifying the initial state to  $\text{last}(h)$ , and the payoff functions to  $\text{payoff}_i'$  defined as follows. For all outcomes  $\rho$  that start in  $\text{last}(h)$ ,  $\text{payoff}_i'(\rho) = \text{payoff}_i(h \hat{\ } \rho)$ , where  $h \hat{\ } \rho = h_{\leq |h|-1} \cdot \rho$ . Notice that the strategy sets of  $\mathcal{G}$  and  $\mathcal{G}_h$  are identical, and that for any  $\sigma_P \in \Sigma_P$ , we have  $\mathbf{Out}(\mathcal{G}_h, \sigma_P) = \mathbf{Out}_{\text{last}(h)}(\mathcal{G}, \sigma_P)$ .

► **Lemma 2.** *For all  $h' \in \mathbf{Hist}_{\text{last}(h)}(\mathcal{G})$ , it holds that  $\mathbf{aVal}_i(\mathcal{G}_h, h') = \mathbf{aVal}_i(\mathcal{G}, h \hat{\ } h')$ ,  $\mathbf{acVal}_i(\mathcal{G}_h, h') = \mathbf{acVal}_i(\mathcal{G}, h \hat{\ } h')$ , and  $\mathbf{cVal}_i(\mathcal{G}_h, h') = \mathbf{cVal}_i(\mathcal{G}, h \hat{\ } h')$ .*

### 3 Existence of Admissible Strategies

We start this section with two examples of quantitative games with no admissible strategies (for player 1). Then we identify a large and natural class of games for which the existence of admissible strategies is guaranteed.

Consider the games  $\mathcal{A}$  and  $\mathcal{G}$  in Fig. 4. Starting at  $s_1$ , the payoff of player 1, in the two games is defined as follows: an outcome that does not visit  $s_3$  has a payoff equal to 0, otherwise, the payoff is equal to the number of times vertex  $a$  appears in the outcome. The lemma below states that player 1 does not have admissible strategies in those two games. We sketch the proof idea.

Consider first the one-player game  $\mathcal{A}$ . The antagonistic value at vertex  $s_1$  is  $\infty$ . Any strategy which never visits  $s_3$  is weakly dominated by strategies that visit  $a$  at least once (i.e. with outcome  $(s_1 a s_1)^+ s_3^0$ ). Furthermore, a strategy which does visit  $s_3$  and  $k$  times  $a$  is weakly dominated by any strategy that visits  $a$  at least  $k + 1$  times and then goes to  $s_3$ .

The idea is similar for  $\mathcal{G}$  where the cooperative value at  $s_1$  is  $\infty$ . Every strategy which does not allow outcomes visiting  $s_3$  are weakly dominated by those that attempt to visit  $a$  by visiting  $s_2$  at least once (as from  $s_2$ , the other player can cooperate and visit  $a$ ), and then go to  $s_3$ . Moreover, it is always possible to attempt to visit  $a$  once more before going to  $s_3$ , thus any strategy which eventually goes to  $s_3$  is also weakly dominated.

► **Lemma 3.** *Player 1 does not have admissible strategies in games  $\mathcal{G}$  and  $\mathcal{A}$ .*

In the two examples above, either the  $\mathbf{aVal}$  or the  $\mathbf{cVal}$  (which are both equal to  $\infty$ ) are not achievable. This is not a coincidence. We now show that all the games that admit witnessing strategies for those values are guaranteed to have admissible strategies.

**Games with strategies witnessing aVal and cVal.** A game is *well-formed* whenever it admits witnessing strategies for **aVal** and **cVal**, *i.e.* it satisfies:

1. For all  $i \in P$ , and  $h \in \mathbf{Hist}_{v_{\text{init}}}(\mathcal{G})$ ,  $\exists \sigma_i \in \Sigma_i$ ,  $\mathbf{aVal}_i(h, \sigma_i) = \mathbf{aVal}_i(h)$ .
2. For all  $i \in P$ , and  $h \in \mathbf{Hist}_{v_{\text{init}}}(\mathcal{G})$ ,  $\exists \sigma_i \in \Sigma_i$ ,  $\mathbf{cVal}_i(h, \sigma_i) = \mathbf{cVal}_i(h)$ .

These conditions will also be referred as Assumption 1 and 2.

We now establish the existence of admissible strategies for all well-formed games.

► **Theorem 4.** *In all well-formed games all players have admissible strategies.*

The result follows from Lemmas 6 and 7 below: the proof consists in showing that a particular type of admissible strategies, called *strongly cooperative-optimal*, always exists. Usually, those strategies are only a strict subset of the admissible strategies available to a player. Nevertheless, they are peculiar as they are guaranteed to exist.

► **Definition 5.** A strategy  $\sigma_i$  is *strongly cooperative-optimal (SCO)* if for all  $h \in \mathbf{Hist}_{v_{\text{init}}}(\sigma_i)$ , if  $\mathbf{cVal}_i(h) > \mathbf{aVal}_i(h)$  then  $\mathbf{cVal}_i(h, \sigma_i) = \mathbf{cVal}_i(h)$ , and if  $\mathbf{aVal}_i(h) = \mathbf{cVal}_i(h)$  then  $\mathbf{aVal}_i(h, \sigma_i) = \mathbf{aVal}_i(h)$ .

Strongly cooperative-optimal strategies are admissible because their cooperative values are always maximal, and moreover, if a payoff better than the antagonistic value cannot be achieved ( $\mathbf{aVal}_i(h) = \mathbf{cVal}_i(h)$ ), then they are worst-case optimal. Any strategy which obtains a better payoff than a SCO strategy against some adversary will obtain a worse payoff against another one.

► **Lemma 6.** *All strongly cooperative-optimal strategies are admissible.*

**Proof.** Let  $\sigma_i$  be a strongly cooperative-optimal strategy for player  $i$ . Assume towards a contradiction that some  $\sigma'_i$  weakly dominates  $\sigma_i$ . Let  $h$  be any minimal history compatible with  $\sigma_i$  such that  $\sigma_i(h) \neq \sigma'_i(h)$ .

If  $\mathbf{aVal}_i(h) < \mathbf{cVal}_i(h)$ , then since  $\text{last}(h)$  is controlled by player  $i$ ,  $\mathbf{aVal}_i(h\sigma'_i(h)) \leq \mathbf{aVal}_i(h) < \mathbf{cVal}_i(h)$ , and since  $\sigma_i$  is strongly cooperative optimal  $\mathbf{cVal}_i(h\sigma_i(h), \sigma_i) = \mathbf{cVal}_i(h)$ . Therefore, as the histories  $h\sigma_i(h)$  and  $h\sigma'_i(h)$  are distinct, there is a strategy  $\tau \in \Sigma_{-i}$  such that  $\text{payoff}_i(\mathbf{Out}_{h\sigma_i(h)}(\sigma_i, \tau)) = \mathbf{cVal}_i(h) > \mathbf{aVal}_i(h) \geq \text{payoff}_i(\mathbf{Out}_{h\sigma'_i(h)}(\sigma'_i, \tau))$ . This contradicts that  $\sigma'_i$  weakly dominates  $\sigma_i$ .

Otherwise  $\mathbf{aVal}_i(h) = \mathbf{cVal}_i(h)$ , then since  $\sigma_i$  is strongly cooperative optimal, for all  $\tau \in \Sigma_{-i}$ ,  $\text{payoff}_i(\mathbf{Out}_h(\sigma_i, \tau)) = \mathbf{cVal}_i(h)$  and  $\text{payoff}_i(\mathbf{Out}_h(\sigma'_i, \tau)) \leq \mathbf{cVal}_i(h)$ . It follows that no outcome of  $\sigma'_i$  obtains a better payoff than  $\sigma_i$ . We thus obtain a contradiction. ◀

By Lem. 6, to prove the existence of admissible strategies, it suffices to prove the existence of strongly cooperative-optimal strategies. We actually give a constructive proof.

► **Lemma 7.** *In all well-formed games all players have SCO strategies.*

Let us describe the idea of the construction. Consider any player  $i$ . We define the strategy  $\sigma$  of player  $i$  as follows. For any history  $h$ , if  $\mathbf{aVal}_i(h) = \mathbf{cVal}_i(h)$ , then  $\sigma$  plays a worst-case optimal strategy from  $h$ , say  $\sigma_h^{\text{wco}}$ . Otherwise, we define  $\sigma$  starting from an outcome  $\rho_h$  with  $\text{payoff}_i = \mathbf{cVal}_i(h)$ , and we define  $\sigma$  in such a way that it follows  $\rho_h$ . In this case, whenever another player deviates from  $\rho_h$ , say, at history  $h'$ , we reevaluate how to play according to whether  $\mathbf{aVal}_i(h') < \mathbf{cVal}_i(h')$  or  $\mathbf{aVal}_i(h') = \mathbf{cVal}_i(h')$ . Here, the existence of  $\sigma_h^{\text{wco}}$  and that of  $\rho_h$  are guaranteed by the fact that the game is well-formed.

In subsequent sections, we consider SCO strategies in residual games  $\mathcal{G}_h$ , so let us note that these games satisfy the required assumptions if  $\mathcal{G}$  does, which follows from Lem. 2.

► **Lemma 8.** *For any well-formed game  $\mathcal{G}$ , for all histories  $h \in \mathbf{Hist}_{v_{\text{init}}}$ , the residual game  $\mathcal{G}_h$  is also well-formed.*

We end this section with an interesting observation: an infinite weak dominance chain is not necessarily dominated by an admissible strategy, as shown in the next example. The reader should contrast the example with the fact that in the Boolean case all dominated strategies are dominated by an admissible strategy [2, Thm. 11].

► **Example 9 (Non-dominated weak dominance chains).** There are quantitative games that have infinite dominance chains and no “maximal” admissible strategy weakly dominating them. Consider the game depicted in Fig. 3. Denote by  $\sigma^k$  the strategy of player 1 (controlling square vertices) which consists in moving from  $s_1$  to  $s_2$  exactly  $k$  times, and then going left (unless payoff of 2 was reached in the meantime). Then for all  $k \in \mathbb{N}$ ,  $\sigma^k$  is weakly dominated by  $\sigma^{k+1}$  because if the adversary decides to move right from  $s_2$  at the  $(k+1)$ -th step,  $\sigma^{k+1}$  performs better than  $\sigma^k$ , and otherwise they yield identical outcomes. It follows that all strategies  $\sigma^k$  for  $k \geq 0$ , are dominated. Here, the only admissible strategy  $\sigma^\infty$  consists in looping in the cycle forever, which does not dominate any  $\sigma^k$  since if the adversary always moves left from  $s_2$ , then  $\sigma^\infty$  yields less than  $\sigma^k$ .

► **Remark.** Above, we have defined strongly cooperative-optimal strategies that favour cooperation whenever it can have an added value. We have established that those strategies are always admissible. There are other classes of strategies that are always admissible, and we define another interesting class here. A strategy  $\sigma_i$  is a *worst-case cooperative optimal strategy*, if for all  $h \in \mathbf{Hist}_{v_{\text{init}}}(\sigma_i)$ :  $\mathbf{aVal}_i(h, \sigma_i) = \mathbf{aVal}_i(h)$ , and  $\mathbf{cVal}_i(h, \sigma_i) = \mathbf{acVal}_i(h)$ .

So those strategies ensure the worst-case value at all times and leave open the best cooperation possible under that worst-case guarantee.

► **Lemma 10.** *All worst-case cooperative optimal strategies are admissible.*

However, some well-formed games do not have worst-case cooperative optimal strategies.

## 4 Value-based Characterization of Admissible Strategies

We present our main result, which is, a value-based characterization of admissible strategies.

For any game  $\mathcal{G}$ , and player  $i$ , let us define the following property, denoted  $\star(h, \sigma)$ , for a given strategy  $\sigma \in \Sigma_i(\mathcal{G})$  and history  $h$ :

$$\mathbf{cVal}_i(h, \sigma) > \mathbf{aVal}_i(h) \tag{1}$$

$$\vee \mathbf{aVal}_i(h, \sigma) = \mathbf{cVal}_i(h, \sigma) = \mathbf{aVal}_i(h) = \mathbf{acVal}_i(h), \tag{2}$$

Intuitively, we will show that a strategy is admissible if at all histories, either the strategy promises a cooperative value greater than the antagonistic value at the current vertex, or a higher cooperative value cannot be obtained without risking a lower antagonistic value (*i.e.*  $\mathbf{aVal}_i(h) = \mathbf{acVal}_i(h)$ ) and the strategy is worst-case optimal.

It turns out that requiring this property at all histories ending in a player’s vertices characterize admissible strategies. We state our result in the following theorem.

► **Theorem 11.** *Under Assumption 1, for any game  $\mathcal{G}$ , player  $i$ , and  $\sigma_i \in \Sigma_i(\mathcal{G})$ ,  $\sigma_i$  is admissible if, and only if, for all  $h \in \mathbf{Hist}_{v_{\text{init}}}(\mathcal{G}, \sigma_i)$  with  $\text{last}(h) \in V_i$ ,  $\star(h, \sigma_i)$  holds.*

It will be useful to consider the negation of  $\star(h, \sigma)$ , which we simplify as follows:

► **Lemma 12.** For all histories  $h$  and strategy  $\sigma$ , the negation of  $\star(h, \sigma)$  is equivalent to

$$\mathbf{cVal}_i(h, \sigma) \leq \mathbf{aVal}_i(h) \wedge \mathbf{aVal}_i(h, \sigma) < \mathbf{aVal}_i(h) \quad (3)$$

$$\vee \mathbf{cVal}_i(h, \sigma) = \mathbf{aVal}_i(h, \sigma) = \mathbf{aVal}_i(h) \wedge \mathbf{acVal}_i(h) > \mathbf{aVal}_i(h). \quad (4)$$

**Proof of Thm. 11.**  $\Rightarrow$  We prove the contrapositive. Assume that  $\exists h \in \mathbf{Hist}_{v_{\text{init}}}(\mathcal{G}, \sigma_i)$ ,  $\text{last}(h) \in V_i$  and  $\neg \star(h, \sigma_i)$ . Then by Lem. 12, either (3) or (4) holds for  $(h, \sigma_i)$ .

Assume (3) holds for  $(h, \sigma_i)$ . By Assumption 1, there exists a *worst-case optimal* strategy  $\sigma_h^{\text{wco}}$  from  $h$ , with  $\mathbf{aVal}_i(h, \sigma_h^{\text{wco}}) = \mathbf{aVal}_i(h)$ . Define  $\sigma'_i \stackrel{\text{def}}{=} \sigma_i[h \leftarrow \sigma_h^{\text{wco}}]$ . We claim that  $\sigma'_i$  weakly dominates  $\sigma_i$ . In fact, for any  $\sigma_{-i} \in \Sigma_{-i}(\mathcal{G})$  with  $h \notin \mathbf{Hist}_{v_{\text{init}}}(\mathcal{G}, \sigma_{-i})$ , we have  $\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma_i, \sigma_{-i}) = \mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma'_i, \sigma_{-i})$ . For any  $\sigma_{-i} \in \Sigma_{-i}(\mathcal{G})$  compatible with  $h$ , both outcomes go through  $h$ . By definition of  $\sigma'_i$ ,  $\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma'_i, \sigma_{-i}) = h_{\leq |h|-1} \cdot \mathbf{Out}_h(\mathcal{G}, \sigma_h^{\text{wco}}, \sigma_{-i})$ . Therefore, we have that  $\text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma'_i, \sigma_{-i})) \geq \mathbf{aVal}_i(h)$  by definition of  $\sigma_h^{\text{wco}}$ . The latter is greater than  $\mathbf{cVal}_i(h, \sigma_i)$  from (3), so greater than  $\text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma_i, \sigma_{-i}))$  by definition of  $\mathbf{cVal}_i(\cdot)$ . Thus,  $\sigma'_i$  very weakly dominates  $\sigma_i$ . Since by assumption,  $\mathbf{aVal}_i(h, \sigma_i) < \mathbf{aVal}_i(h)$ , and  $h$  is compatible with  $\sigma_i$ , there is a strategy  $\sigma_{-i} \in \Sigma_{-i}$  such that  $h \subseteq_{\text{pref}} \mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma_i, \sigma_{-i})$  and  $\text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma_i, \sigma_{-i})) < \mathbf{aVal}_i(h)$ . As shown before,  $\mathbf{aVal}_i(h) \leq \text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma'_i, \sigma_{-i}))$ . Hence,  $\sigma'_i$  weakly dominates  $\sigma_i$ .

Assume now that (4) holds. Consider  $\varepsilon > 0$  small enough so that  $\mathbf{acVal}_i(h) > \mathbf{aVal}_i(h) + \varepsilon$ . By definition of  $\mathbf{acVal}_i(h)$ , there exists a strategy  $\tau_i \in \Sigma_i$  such that  $\mathbf{cVal}_i(h, \tau_i) \geq \mathbf{aVal}_i(h) + \varepsilon$ , and moreover  $\mathbf{aVal}_i(h, \tau_i) \geq \mathbf{aVal}_i(h)$ . Consider  $\tau_{-i} \in \Sigma_{-i}$  compatible with  $h$  such that  $\text{payoff}_i(\mathbf{Out}_h(\mathcal{G}, h, (\tau_i, \tau_{-i}))) \geq \mathbf{cVal}_i(h, \tau_i) - \frac{\varepsilon}{2} \geq \mathbf{aVal}_i(h) + \frac{\varepsilon}{2} > \mathbf{aVal}_i(h)$ . Note that such a  $\tau_{-i}$  exists by definition of  $\mathbf{cVal}_i(h, \tau_i)$ . It follows that  $\sigma_i[h \leftarrow \tau_i]$  weakly dominates  $\sigma_i$ . In fact, the outcomes are identical for any outcome not compatible with  $h$ . For any  $\sigma_{-i}$  compatible with  $h$ , we have  $\text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma_i, \sigma_{-i})) = \mathbf{aVal}_i(h)$  by (4). Moreover,  $\text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma'_i, \sigma_{-i})) \geq \mathbf{aVal}_i(h)$  since at  $h$  we have that  $\mathbf{aVal}_i(h, \tau_i) \geq \mathbf{aVal}_i(h)$ ; thus  $\mathbf{aVal}_i(h, \sigma'_i) \geq \mathbf{aVal}_i(h)$ . Furthermore, we have  $\text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma'_i, \tau_{-i})) > \mathbf{aVal}_i(h) \geq \text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma_i, \tau_{-i}))$ .

$\Leftarrow$  Assume that for all  $h \in \mathbf{Hist}_{v_{\text{init}}}(\mathcal{G}, \sigma_i)$  with  $\text{last}(h) \in V_i$ , we have  $\star(h, \sigma_i)$ , and that  $\sigma_i$  is weakly dominated by some strategy  $\sigma'_i$ . We will show a contradiction.

Let  $\sigma_{-i}$  be a strategy in  $\Sigma_{-i}(\mathcal{G})$  and  $\rho = \mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma_i, \sigma_{-i})$  and  $\rho' = \mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma'_i, \sigma_{-i})$ . If  $\rho = \rho'$  then  $\text{payoff}_i(\rho') \leq \text{payoff}_i(\rho)$  and otherwise let  $j$  be the first index where they differ, and  $h = \rho_{\leq j-1} = \rho'_{\leq j-1}$ . We have that  $h$  is compatible with both strategies,  $\text{last}(h) \in V_i$  and  $\sigma_i(h) \neq \sigma'_i(h)$ .

If (1) holds, that is,  $\mathbf{cVal}_i(h, \sigma_i) > \mathbf{aVal}_i(h)$ , consider  $\varepsilon > 0$  such that  $\mathbf{cVal}_i(h, \sigma_i) > \mathbf{aVal}_i(h) + \varepsilon$ , and a strategy  $\sigma'_{-i} \in \Sigma_{-i}$  which ensures that  $\text{payoff}_i(\mathbf{Out}_{h\sigma_i(h)}(\mathcal{G}, \sigma_i, \sigma'_{-i})) \geq \mathbf{cVal}_i(h, \sigma_i) - \frac{\varepsilon}{2}$ , and  $\text{payoff}_i(\mathbf{Out}_{h\sigma'_i(h)}(\mathcal{G}, \sigma'_i, \sigma'_{-i})) \leq \mathbf{aVal}_i(h, \sigma'_i) + \frac{\varepsilon}{2}$ . Such a strategy profile  $\sigma'_{-i}$  exists since  $h\sigma'_i(h)$  and  $h\sigma_i(h)$  are distinct, and since  $\text{last}(h) \in V_i$ . The latter also implies that  $\mathbf{aVal}_i(h) \geq \mathbf{aVal}_i(h, \sigma'_i)$ . It thus follows that

$$\text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma_i, \sigma_{-i}[h \leftarrow \sigma'_{-i}])) > \text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma'_i, \sigma_{-i}[h \leftarrow \sigma'_{-i}]))$$

contradicting the fact that  $\sigma'_i$  weakly dominates  $\sigma_i$ .

Therefore (2) must hold, and  $\mathbf{acVal}_i(h) = \mathbf{aVal}_i(h)$ . If there exists  $j \geq |h|$  such that  $\mathbf{aVal}_i(\rho'_{\leq j}) < \mathbf{aVal}_i(h)$ , then there exists  $\varepsilon > 0$  and a strategy profile  $\sigma'_{-i} \in \Sigma_{-i}$  compatible with  $h$  which ensures that  $\text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma'_i, \sigma'_{-i})) \leq \mathbf{aVal}_i(\rho'_{\leq j}) + \varepsilon < \mathbf{aVal}_i(h) \leq \text{payoff}_i(\mathbf{Out}_{v_{\text{init}}}(\mathcal{G}, \sigma_i, \sigma'_{-i}))$ . This contradicts  $\sigma'$  weakly dominating  $\sigma$ . Hence for all  $j \geq |h|$ ,  $\mathbf{aVal}_i(\rho'_{\leq j}) \geq \mathbf{aVal}_i(h)$ . Now, observe that  $\text{payoff}_i(\rho') \leq \mathbf{acVal}_i(h)$ . In fact, one can construct a strategy  $\tau$ , which, from  $h$  follows  $\rho'$ , and in case another player does not respect  $\rho$ , switches to a worst-case optimal strategy ensuring  $\mathbf{aVal}_i(\rho'_{\leq j}) \geq \mathbf{aVal}_i(h)$ . It follows

that  $\text{payoff}_i(\rho') \leq \mathbf{cVal}_i(h, \tau) \leq \mathbf{acVal}_i(h)$ . Furthermore, by (2),  $\text{payoff}_i(\rho) \geq \mathbf{acVal}_i(h) = \mathbf{aVal}_i(h, \sigma_i)$ , so  $\text{payoff}_i(\rho') \leq \text{payoff}_i(\rho)$ . This being true for all strategies of  $\Sigma_{-i}$  proves that  $\sigma_i$  very weakly dominates  $\sigma'_i$  and contradicts that  $\sigma'_i$  weakly dominates  $\sigma_i$ . ◀

## 5 Characterization of the Outcomes of Admissible Strategies

Observe that the characterization of Thm. 11 does not immediately yield an effective representation of the *set* of admissible strategies. In order to reason about the possible behaviors observable in a game under admissible strategies we are interested in describing the set of outcomes that can be observed when all players play admissible strategies. In this section, for each player, we give a linear temporal logic description of the outcomes that are each compatible with at least one admissible strategy.

Note that our main goal is to obtain such a characterization in full generality, for all well-formed games so we defer computability considerations to the next section. We will then see how the three types of values can be computed at all histories.

Let us fix a game  $\mathcal{G}$ , and player  $i$ . We present the intuition of the characterization. If an outcome  $\rho$  is compatible with an admissible strategy, say  $\sigma_i$ , then all prefixes  $h$  with  $\text{last}(h) \in V_i$  must satisfy (1) or (2). Given  $h$ , if (1) holds, then two things can happen. Either  $\text{payoff}_i(\rho) > \mathbf{aVal}_i(h)$ , and thus  $\rho$  witnesses  $\mathbf{cVal}_i(h, \sigma_i) > \mathbf{aVal}_i(h)$ , or this is not the case but there is another outcome  $\rho'$  – compatible with  $\sigma_i$  – extending  $h$  with  $\text{payoff}_i(\rho') > \mathbf{aVal}_i(h)$ . Notice how the longest common prefix of  $\rho$  and  $\rho'$  ends always with a vertex in  $V_{-i}$  since both outcomes are compatible with  $\sigma_i$ . If (2) holds at  $h$ , then, in particular,  $\text{payoff}_i(\rho) = \mathbf{aVal}_i(h)$  and, moreover,  $\mathbf{aVal}_i$  remains constant at all prefixes of  $\rho$  extending  $h$ . The last observation simply follows from  $\mathbf{aVal}_i(h, \sigma_i) = \mathbf{aVal}_i(h)$  which is implied by (2).

**Extended  $\text{LTL}_{\text{payoff}}$ .** Let  $\mathbf{aValues}_i = \{\mathbf{aVal}_i(h) \mid h \text{ is a history}\}$  be the set of antagonistic values of player  $i$ . We will now define atomic propositions attached to edges of a game. Formally, we have a *labelling function*  $\lambda : E \rightarrow \mathcal{P}(\text{AP})$  which assigns to every edge a set of propositions from AP. The set AP includes the proposition  $\mathbf{V}_i$  whose truth value, for every edge  $e = (u, v)$ , is determined as follows:  $\mathbf{V}_i \in \lambda(u, v) \stackrel{\text{def}}{\iff} u \in V_i$ .

We consider  $\text{LTL}_{\text{payoff}}$  with atomic propositions as defined above and additional propositions  $\mathbf{aVal}_q^i$ ,  $\mathbf{acVal}_q^i$ , and  $\mathbf{gAlt}_q^i$  defined for all  $q \in \mathbf{aValues}_i$ . The semantics of these are straightforward: for an outcome  $\rho$  and  $k \in \mathbb{N}_{>0}$  we have

$$\begin{aligned} (\rho, k) \models \mathbf{aVal}_q^i &\stackrel{\text{def}}{\iff} \mathbf{aVal}_i(\rho_{\leq k}) = q, \\ (\rho, k) \models \mathbf{acVal}_q^i &\stackrel{\text{def}}{\iff} \mathbf{acVal}_i(\rho_{\leq k}) = q, \text{ and} \\ (\rho, k) \models \mathbf{gAlt}_q^i &\stackrel{\text{def}}{\iff} \rho_k \in V_{-i} \wedge \exists v' \neq \rho_k : (\rho_k, v') \in E \wedge \mathbf{cVal}_i(\rho_{\leq k} \cdot v') > q, \end{aligned}$$

with the convention that, when  $k$  is omitted, we assume it is 1.

As mentioned earlier, we consider two cases depending on whether (1) or (2) hold. Thus, let us define the corresponding two sub-formulas:

$$\begin{aligned} \varphi_1 &\stackrel{\text{def}}{=} \bigvee_{q \in \mathbf{aValues}_i} (\mathbf{aVal}_q^i \wedge (\text{payoff}_i > q \vee \mathbf{F}(\mathbf{gAlt}_q^i))), \text{ and} \\ \varphi_2 &\stackrel{\text{def}}{=} \bigvee_{q \in \mathbf{aValues}_i} (\mathbf{aVal}_q^i \wedge \mathbf{acVal}_q^i \wedge \text{payoff}_i = q \wedge \mathbf{G}(\mathbf{aVal}_q^i)). \end{aligned}$$

We define the following formula which will be shown to capture the outcomes of admissible strategies:  $\Phi_{\text{adm}}^i \stackrel{\text{def}}{=} \mathbf{G}(\neg \mathbf{V}_i \vee \varphi_1 \vee \varphi_2)$ .



► **Theorem 13.** *For any well-formed game  $\mathcal{G}$ , outcome  $\rho$  satisfies  $\Phi_{\text{adm}}^i$  if, and only if, it is compatible with an admissible strategy for player  $i$ .*

We give the idea of the proof. For any outcome  $\rho$  compatible with an admissible strategy  $\sigma_i$  for player  $i$ . We show that for any prefix  $h$  of  $\rho$  with  $\text{last}(h) \in V_i$ ,  $(\rho, |h|)$  satisfies either  $\varphi_1$  or  $\varphi_2$ . In fact, by Thm. 11, either (1) or (2) hold, and we show that these correspond to  $\varphi_1$  and  $\varphi_2$ .

Conversely, for any  $\rho$  satisfying  $\Phi_{\text{adm}}^i$ , we construct an admissible strategy  $\sigma_i$  for player  $i$  compatible with  $\rho$ . The strategy follows  $\rho$ , and in case of deviation, it switches immediately either to an SCO – which is guaranteed to exist – or to a worst-case optimal strategy, depending on whether  $\varphi_1$  or  $\varphi_2$  holds at the current history. The resulting strategy is proven to be admissible.

**Assuming prefix-independence.** Before concluding this section, let us consider the consequences of further assuming that our payoff function is *prefix-independent*.

3. For all  $i \in P$ , for all outcomes  $\rho$ , it holds that  $\forall j \in \mathbb{N}, \text{payoff}_i((\rho_k)_{k \geq j}) = \text{payoff}_i(\rho)$ .

Observe that, under Assumption 3, the set  $\mathbf{aVal}_i$  can be equivalently defined as  $\{\mathbf{aVal}_i(v) \mid v \in V\}$  and is thus finite. One can also extend the labelling  $\lambda$  and set of atomic propositions AP such that, for every edge  $e = (u, v)$  and  $q \in \mathbf{aVal}_i$ :

$$\begin{aligned} \mathbf{aVal}_q^i \in \lambda(u, v) &\stackrel{\text{def}}{\iff} \mathbf{aVal}_i(u) = q, \\ \mathbf{acVal}_q^i \in \lambda(u, v) &\stackrel{\text{def}}{\iff} \mathbf{acVal}_i(u) = q, \text{ and} \\ \mathbf{gAlt}_q^i \in \lambda(u, v) &\stackrel{\text{def}}{\iff} u \in V_{-i} \wedge \exists v' \neq v : (u, v') \in E \wedge \mathbf{cVal}_i(v') > q. \end{aligned}$$

It immediately follows that:

► **Lemma 14.** *Under Assumption 3, for all  $i \in P$ ,  $\Phi_{\text{adm}}^i$  is expressible in  $\text{LTL}_{\text{payoff}}$ .*

## 6 Applications and Future Works

In this section, we show how to apply Theorem 11 (value-based characterization of admissible strategies) and Theorem 13 (characterization of the set of outcomes of admissible strategies) to solve relevant verification and synthesis problems.

**Classical payoff functions.** So far, we have assumed that games were equipped for each player  $i \in P$  with a payoff function. To define payoff functions, we proceed as usual by first assigning weights to edges of the game graph using *weight functions*  $w_i : E \rightarrow \mathbb{Q}$ , one for each player  $i \in P$ . With the weight function  $w_i$ , we associate to each outcome  $\rho = \rho_1 \rho_2 \dots \rho_n \dots$ , an infinite sequence of rational values  $w_i(\rho) = w_i(\rho_1 \rho_2) w_i(\rho_2 \rho_3) \dots w_i(\rho_n \rho_{n+1}) \dots$ , and we aggregate this sequence of values with measures such as  $\text{Inf}$ ,  $\text{Sup}$ ,  $\text{LimInf}$ ,  $\text{LimSup}$ , and mean payoff ( $\underline{\text{MP}}$  and  $\overline{\text{MP}}$ ). It is well known, see *e.g.* [7] and [18], that all the payoff functions defined above satisfy Assumptions 1-2. By Theorem 4, we get the following.

► **Lemma 15.** *In games with payoff functions from  $\text{Inf}$ ,  $\text{Sup}$ ,  $\text{LimInf}$ ,  $\text{LimSup}$ ,  $\underline{\text{MP}}$ , and  $\overline{\text{MP}}$ , all players have admissible strategies.*

It is also known that, in games defined with the payoff functions considered here, the antagonistic and cooperative values ( $\mathbf{cVal}$  and  $\mathbf{aVal}$ ) are computable. One can also show that  $\mathbf{acVal}$  is computable for prefix-independent payoff functions. Indeed, this value of a



vertex coincides with the  $\mathbf{cVal}$  inside the sub-graph induced by the vertices with the optimal antagonistic value. Furthermore, using a classical transformation on the game structure, we can guarantee that all payoff functions. We thus obtain the following result, by Lemma 14.

► **Lemma 16.** *In games with payoff functions from  $\text{Inf}$ ,  $\text{Sup}$ ,  $\text{LimInf}$ ,  $\text{LimSup}$ ,  $\underline{\text{MP}}$ , and  $\overline{\text{MP}}$ , the formulas  $\Phi_{\text{adm}}^i$  for all  $i \in P$  are effectively computable, finite, and expressible in  $\text{LTL}_{\text{payoff}}$ .*

We will now consider several problems of interest which can be solved using the characterizations that we have developed in the previous sections. All the results are applicable to the measures concerned by Lemmas 15 and 16.

**Deciding the admissibility of a finite memory strategy.** As a first example, we consider the problem of deciding, given a game structure  $\mathcal{G}$ , and a (finite memory) strategy  $\sigma_i$  for player  $i \in P$  described as a finite state transducer  $M_i$ , if  $\sigma_i$  is admissible in  $\mathcal{G}$ .

To solve this problem, we rely on Theorem 11 and proceed as follows. First, we compute for each vertex  $v$  of the game  $\mathcal{G}$ , the values  $\mathbf{aVal}_i(\mathcal{G}, v)$ ,  $\mathbf{cVal}_i(\mathcal{G}, v)$ , and  $\mathbf{acVal}_i(\mathcal{G}, v)$ . Second, we construct the synchronized product between the transducer  $M_i$  that defines the strategy  $\sigma_i$  and the game  $\mathcal{G}$ . States in this product are of the form  $(v, m)$  where  $v$  is a vertex of  $\mathcal{G}$  and  $m$  is a (memory) state of the transducer  $M_i$ . Third, we compute for each state  $(v, m)$  the values  $\mathbf{aVal}_i(\mathcal{G}, (v, m), \sigma_i)$ ,  $\mathbf{cVal}_i(\mathcal{G}, (v, m), \sigma_i)$ , and  $\mathbf{acVal}_i(\mathcal{G}, (v, m), \sigma_i)$ . Finally, we verify that there is no reachable vertex  $(v, m)$  in the product where condition (1) or condition (2) are falsified. We then obtain the following theorem:

► **Theorem 17.** *Given a game  $\mathcal{G}$  and a finite memory strategy  $\sigma_i$  for player  $i \in P$  specified as a finite state transducer  $M_i$ , we can decide if  $\sigma_i$  is an admissible strategy for player  $i$  in  $\text{PTime}$  for measures  $\text{Inf}$ ,  $\text{Sup}$ ,  $\text{LimInf}$ ,  $\text{LimSup}$ ; in  $\text{NP} \cap \text{coNP}$  for  $\underline{\text{MP}}$ , and  $\overline{\text{MP}}$ .*

**Model-checking under admissibility.** We now turn to the following problem. Given a game structure  $\mathcal{G}$  and a  $\text{LTL}_{\text{payoff}}$  formula  $\varphi$ , decide if all outcomes of the game that are compatible with the admissible strategies of all players satisfy  $\varphi$ , i.e. if  $\bigcap_{i \in P} \mathbf{Out}(\mathcal{G}, \mathfrak{A}_i(\mathcal{G})) \models \varphi$ . This problem was introduced in the Boolean setting in [5] and allows one to check that a property is induced by the rationality of the players in a game.

► **Theorem 18.** *For all measures  $\text{Inf}$ ,  $\text{Sup}$ ,  $\text{LimInf}$ ,  $\text{LimSup}$ ,  $\underline{\text{MP}}$ ,  $\overline{\text{MP}}$ , one can decide, given game  $\mathcal{G}$  and  $\text{LTL}_{\text{payoff}}$  formula  $\varphi$ , whether  $\bigcap_{i \in P} \mathbf{Out}(\mathcal{G}, \mathfrak{A}_i(\mathcal{G})) \models \varphi$ .*

**Proof Sketch.** For each player  $i \in P$ , consider the formula  $\Phi_{\text{adm}}^i$  from Theorem 13, which describes the set  $\mathbf{Out}(\mathcal{G}, \mathfrak{A}_i(\mathcal{G}))$ . The formula is finite and constructible by Lemma 16. The problem now amounts to verifying if  $\mathcal{G}$  satisfies the specification  $(\bigwedge_{i \in P} \Phi_{\text{adm}}^i) \implies \varphi$ . For all payoff functions, except mean-payoff, this can be reduced to model checking an  $\text{LTL}$  formula (since the measures are regular). For  $\overline{\text{MP}}$  and  $\underline{\text{MP}}$ , the result follows from [3] which shows that the model checking problem against  $\text{LTL}_{\text{payoff}}$  is decidable. ◀

**Quantitative assume-admissible synthesis.** In [4], a new rule for reactive synthesis in non-zero sum  $n$ -player games was proposed. The setting there is similar to the setting considered here but it is Boolean: each player  $i \in P$  has his own omega-regular objective  $O_i \subseteq V^\omega$ . The synthesis rule asks if player  $i \in P$  has a strategy to enforce its own objective  $O_i$  against admissible strategies of the other players. In other words, the rule asks for the existence of worst-case optimal strategies against rational adversaries.

The quantitative extension of this problem asks given a game  $\mathcal{G}$ , a player  $i \in P$ , and a  $\text{LTL}_{\text{payoff}}$  formula  $\varphi$ ,  $\exists \sigma \in \mathfrak{A}_i, \forall \tau \in \mathfrak{A}_{-i}, \varphi$ . Using Theorem 13, we can reduce this query to a plain two-player zero-sum game on the game structure  $\mathcal{G}$  with objective:

$$\exists \sigma \in \Sigma_i, \forall \tau \in \Sigma_{-i}, \Phi_{\text{adm}}^i \wedge \left( \left( \bigwedge_{j \in P \setminus \{i\}} \Phi_{\text{adm}}^j \right) \implies \varphi \right)$$

Since for  $\text{Inf}$ ,  $\text{Sup}$ ,  $\text{LimInf}$ ,  $\text{LimSup}$ ,  $\Phi_{\text{adm}}^i$  and  $\varphi$  are omega-regular, the problem reduces to deciding the winner in a two-player zero-sum game with omega-regular objectives. As a consequence, we obtain the following theorem:

► **Theorem 19.** *The quantitative assume-admissible synthesis problem for player  $i \in P$  is decidable for measures  $\text{Inf}$ ,  $\text{Sup}$ ,  $\text{LimInf}$ ,  $\text{LimSup}$ .*

For the measures  $\text{MP}$ ,  $\overline{\text{MP}}$ , we obtain objectives in which mean-payoff constraints and omega-regular constraints are mixed. On the one hand, those objectives are outside known decidable classes of objectives treated in [8] and in [10]. On the other hand, the undecidability results obtained in [17] do not apply to them. This motivates further research on zero-sum two player games with a mix of mean-payoff and omega-regular objectives.

**Towards iterative elimination.** Once we have computed the admissible strategies for each player, we restrict each player to these strategies, and repeat the computation of the admissible strategies in the restricted game. This can be iterated several times and gives a process that is called *iterative elimination of dominated strategies*, and well known in game theory. This process is difficult to analyze for mean-payoff, because objectives of different players interfere in non-trivial ways and games with Boolean combinations of mean-payoff objectives are undecidable [17]. However it seems feasible for regular payoffs, such as  $\text{Inf}$ ,  $\text{Sup}$ ,  $\text{LimInf}$  and  $\text{LimSup}$ , for which we can construct parity automata recognizing outcomes with  $\text{payoff}_i > q$ . Given  $i \geq 0$ , we can actually compute a parity automaton accepting the set of outcomes of  $\mathcal{S}^i$  which is the set of strategies that remain after  $i$  steps of elimination. We summarize here the ingredients but leave the details for future work. Assume we have a parity automaton representing the outcomes of  $\mathcal{S}^i$ . Note that for  $i = 0$  this is simply all outcomes. If the payoffs are regular, then we can compute values  $\mathbf{cVal}_i(h, \mathcal{S}^i)$ ,  $\mathbf{aVal}_i(h, \mathcal{S}^i)$  and  $\mathbf{acVal}_i(h, \mathcal{S}^i)$ , which correspond to cooperative, antagonistic, and antagonist-cooperative values when players only play strategies from  $\mathcal{S}^i$ . We can then use these values as atomic propositions for a  $\text{LTL}_{\text{payoff}}$  formulas similar to  $\Phi_{\text{adm}}^i$  of Section 5, which characterizes outcomes of strategies of  $\mathcal{S}^{i+1}$ . In the case of regular payoffs this yields a parity automaton which represents the outcomes of  $\mathcal{S}^{i+1}$ . This procedure can then be repeated to compute outcomes that are possible under iterative elimination.

---

## References

- 1 Brandenburger Adam, Friedenberg Amanda, H Jerome, et al. Admissibility in games. *Econometrica*, 2008.
- 2 Dietmar Berwanger. Admissibility in infinite games. In *STACS*, volume 4393 of *LNCS*, pages 188–199. Springer, February 2007.
- 3 Udi Boker, Krishnendu Chatterjee, Thomas A. Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. *ACM Trans. Comput. Logic*, 15(4):27:1–27:25, July 2014. doi:10.1145/2629686.
- 4 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. In Luca Aceto and David de Frutos-Escrig, editors, *CONCUR*, volume 42 of *LIPICs*, pages

- 100–113. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.100.
- 5 Romain Brenguier, Jean-François Raskin, and Mathieu Sassolas. The complexity of admissibility in omega-regular games. In *CSL-LICS'14, 2014*. ACM, 2014. doi:10.1145/2603088.2603143.
  - 6 Krishnendu Chatterjee, Laurent Doyen, Emmanuel Filiot, and Jean-François Raskin. Doomsday equilibria for omega-regular games. In *VMCAI'14*, volume 8318, pages 78–97. Springer, 2014.
  - 7 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Transactions on Computational Logic*, 11(4), 2010. doi:10.1145/1805950.1805953.
  - 8 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdziński. Mean-payoff parity games. In *LICS*, pages 178–187. IEEE Computer Society, 2005. doi:10.1109/LICS.2005.26.
  - 9 Krishnendu Chatterjee, Thomas A Henzinger, and Marcin Jurdziński. Games with secure equilibria. *Theoretical Computer Science*, 365(1):67–82, 2006.
  - 10 Krishnendu Chatterjee, Mickael Randour, and Jean-François Raskin. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Inf.*, 51(3-4):129–163, 2014. doi:10.1007/s00236-013-0182-6.
  - 11 Marco Faella. Admissible strategies in infinite games over graphs. In *MFCS 2009*, volume 5734 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2009.
  - 12 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In *TACAS'10*, volume 6015 of *LNCS*, pages 190–204. Springer, 2010.
  - 13 R. J. Gretlein. Dominance elimination procedures on finite alternative games. *International Journal of Game Theory*, 12(2):107–113, 1983. doi:10.1007/BF01774300.
  - 14 O. Kupferman, G. Perelli, and M.Y. Vardi. Synthesis with rational environments. In *Proc. 12th European Conference on Multi-Agent Systems*, LNCS. Springer, 2014.
  - 15 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989. doi:10.1145/75277.75293.
  - 16 Wolfgang Thomas. On the synthesis of strategies in infinite games. In *STACS*, pages 1–13, 1995. doi:10.1007/3-540-59042-0\_57.
  - 17 Yaron Velner. Robust multidimensional mean-payoff games are undecidable. In Andrew M. Pitts, editor, *FoSSaCS*, volume 9034 of *LNCS*, pages 312–327. Springer, 2015. doi:10.1007/978-3-662-46678-0\_20.
  - 18 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *TCS*, 158(1):343–359, 1996.

# Prompt Delay\*

Felix Klein<sup>1</sup> and Martin Zimmermann<sup>2</sup>

1 Reactive Systems Group, Saarland University, Saarbrücken, Germany  
klein@react.uni-saarland.de

2 Reactive Systems Group, Saarland University, Saarbrücken, Germany  
zimmermann@react.uni-saarland.de

---

## Abstract

Delay games are two-player games of infinite duration in which one player may delay her moves to obtain a lookahead on her opponent's moves. Recently, such games with quantitative winning conditions in weak MSO with the unbounding quantifier were studied, but their properties turned out to be unsatisfactory. In particular, unbounded lookahead is in general necessary.

Here, we study delay games with winning conditions given by PROMPT-LTL, Linear Temporal Logic equipped with a parameterized eventually operator whose scope is bounded. Our main result shows that solving PROMPT-LTL delay games is complete for triply-exponential time. Furthermore, we give tight triply-exponential bounds on the necessary lookahead and on the scope of the parameterized eventually operator. Thus, we identify PROMPT-LTL as the first known class of well-behaved quantitative winning conditions for delay games.

Finally, we show that applying our techniques to delay games with  $\omega$ -regular winning conditions answers open questions in the cases where the winning conditions are given by non-deterministic, universal, or alternating automata.

**1998 ACM Subject Classification** D.2.4 Software/Program Verification

**Keywords and phrases** Infinite Games, Delay Games, Prompt-LTL, LTL

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.43

## 1 Introduction

The synthesis of reactive systems concerns the automatic construction of an implementation satisfying a given specification against every behavior of its possibly antagonistic environment. A prominent specification language is Linear Temporal Logic (LTL), describing the temporal behavior of an implementation [21]. The LTL synthesis problem has been intensively studied since the seminal work of Pnueli and Rosner [22, 23], theoretical foundations have been established [2, 19], and several tools have been developed [4, 7, 8].

However, LTL is not able to express quantitative properties. As an example, consider the classical request-response condition [13], where every request  $q$  has to be answered eventually by some response  $r$ . This property is expressible in LTL via the formula  $\mathbf{G}(q \rightarrow \mathbf{F}r)$ , but the property cannot guarantee any bound on the waiting times between a request and its earliest response. To specify such a behaviour, parameterized logics have been introduced [1, 6, 18, 26], which extend LTL by quantitative operators.

The simplest of these logics is PROMPT-LTL [18], which extends LTL by the prompt eventually operator  $\mathbf{F}_P$ . The scope of this operator is bounded by some arbitrary but

---

\* The first author was supported by an IMPRS-CS PhD Scholarship, the second by the project “TriCS” (ZI 1516/1-1) of the German Research Foundation (DFG).



fixed number  $k$ . With this extension, we can express the aforementioned property by the PROMPT-LTL formula  $\mathbf{G}(q \rightarrow \mathbf{F}_P r)$ , expressing that every request is answered within  $k$  steps. To show that the PROMPT-LTL synthesis problem is as hard as the LTL synthesis problem, i.e., 2EXPTIME-complete, Kupferman et al. introduced the alternating-color technique to reduce the former problem to the latter [18]. Additionally, similar reductions have been proven to exist in other settings too, where PROMPT-LTL can be reduced to LTL using the alternating-color technique, e.g., for (assume-guarantee) model-checking [18]. Finally, the technique is also applicable to more expressive extensions of LTL, e.g., parametric LTL [25], parametric LDL [6], and their variants with costs [26].

Nevertheless, all these considerations assume that the specified implementation immediately reacts to inputs of the environment. However, this assumption might be too restrictive, e.g., in a buffered network, where the implementation may delay its outputs by several time steps. Delay games have been introduced by Hosch and Landweber [14] to overcome this restriction. In the setting of infinite games, the synthesis problem is viewed as a game between two players, the input player “Player  $I$ ”, representing the environment, and the output player “Player  $O$ ”, representing the implementation. The goal of Player  $O$  is to satisfy the specification, while Player  $I$  tries to violate it. Usually, the players move in strict alternation. On the contrary, in a delay game, Player  $O$  can delay her moves to obtain a lookahead on her opponent’s moves. This way, she gains additional information on her opponent’s strategy, which she can use to achieve her goal. Hence, many specifications are realizable, when allowing lookahead, which are unrealizable otherwise.

For delay games with  $\omega$ -regular winning conditions (given by deterministic parity automata) exponential lookahead is always sufficient and in general necessary, and determining the winner is EXPTIME-complete [16]. As LTL formulas can be translated into equivalent deterministic parity automata of doubly-exponential size, these results imply a triply-exponential upper bound on the necessary lookahead in delay games with LTL winning condition and yield an algorithm solving such games with triply-exponential running time. However, no matching lower bounds are known.

Recently, based on the techniques developed for the  $\omega$ -regular case, the investigation of delay games with quantitative winning conditions was initiated by studying games with winning conditions specified in weak monadic second order logic with the unbounding quantifier (WMSO+U) [3]. This logic extends the weak variant of monadic second order logic (WMSO), where only quantification over finite sets is allowed, with an additional unbounding quantifier that allows to express (un)boundedness properties. The resulting logic subsumes all parameterized logics mentioned above. The winner of a WMSO+U delay game with respect to bounded lookahead is effectively computable [27]. However, in general, Player  $O$  needs unbounded lookahead to win such games and the decidability of such games with respect to arbitrary lookahead remains an open problem. In the former aspect, delay games with WMSO+U winning conditions behave worse than those with  $\omega$ -regular ones.

**Our Contribution.** The results on WMSO+U delay games show the relevance of exploring more restricted classes of quantitative winning conditions which are better-behaved. In particular, bounded lookahead should always suffice and the winner should be effectively computable. To this end, we investigate delay games with PROMPT-LTL winning conditions. Formally, we consider the following synthesis problem: given some PROMPT-LTL formula  $\varphi$ , does there exist some lookahead and some bound  $k$  such that Player  $O$  has a strategy producing only outcomes that satisfy  $\varphi$  with respect to the bound  $k$  (and, if yes, compute such a strategy)?

We present the first results for delay games with PROMPT-LTL winning conditions. First, we show that the synthesis problem is in  $3\text{EXPTIME}$  by tailoring the alternating-color technique to delay games and integrating it into the algorithm developed for the  $\omega$ -regular case. In the end, we obtain a reduction from delay games with PROMPT-LTL winning conditions to delay-free parity games of triply-exponential size.

Second, from this construction, we derive triply-exponential upper bounds on the necessary lookahead for Player  $O$ , i.e., bounded lookahead always suffices, as well as a triply-exponential upper bound on the necessary scope of the prompt eventually operator. Thus, we obtain the same upper bounds as for LTL.

Third, we complement all three upper bounds by matching lower bounds, e.g., the problem is  $3\text{EXPTIME}$ -complete and there are triply-exponential lower bounds on the necessary lookahead and on the scope of the prompt eventually operator. The former two lower bounds already hold for the special case of LTL delay games. Thereby, we settle the case of delay games with LTL winning conditions as well as the case of delay games with PROMPT-LTL winning conditions and show that they are of equal complexity and that the same bounds on the necessary lookahead hold. Thus, we prove that delay games with PROMPT-LTL winning conditions are not harder than those with LTL winning conditions. The complexity of solving LTL games increases exponentially when adding lookahead, which is in line with the results in the  $\omega$ -regular case [16], where one also observes an exponential blowup.

Fourth, our proofs are all applicable to the stronger extensions of LTL like parametric LTL [25], parametric LDL [6], and their variants with costs [26], as the alternating-color technique is applicable to them as well and as their formulas can be compiled into equivalent exponential Büchi automata.

Fifth, we show that our lower bounds also answer open questions in the  $\omega$ -regular case mentioned in [16], e.g., on the influence of the branching mode of the specification automaton on the complexity. Recall that the tight exponential bounds on the complexity and the necessary lookahead for  $\omega$ -regular delay games were shown for winning conditions given by *deterministic* automata. Our lower bounds proven here can be adapted to show that both these bounds are doubly-exponential for non-deterministic and universal automata and triply-exponential for alternating automata. Hence, the lower bounds match the trivial upper bounds obtained by determinizing the automata and applying the results from [16]. Thus, we complete the picture in the  $\omega$ -regular case with regard to the branching mode of the specification automaton.

**Related Work.** Delay games with  $\omega$ -regular winning conditions have been introduced by Hosch and Landweber, who proved that the winner w.r.t. bounded lookahead can be determined effectively [14]. Later, they were revisited by Holtmann et al. who showed that bounded lookahead is always sufficient and who gave a streamlined algorithm with doubly-exponential running time and a doubly-exponential upper bound on the necessary lookahead [12]. Recently, the tight exponential bounds on the running time and on the lookahead mentioned above were proven [16]. Delay games with context-free winning conditions turned out to be undecidable for very small fragments [9]. The results of going beyond the  $\omega$ -regular case by considering WMSO+U winning conditions are mentioned above. Furthermore, all delay games with Borel winning conditions are determined [15]. Finally, from a more theoretical point of view, Holtman et al. also showed that delay games are a suitable representation of uniformization problems for relations by continuous functions [12].

All proofs omitted due to space constraints can be found in the full version [17].

## 2 Preliminaries

The set of non-negative (positive) integers is denoted by  $\mathbb{N}$  ( $\mathbb{N}_+$ ). An alphabet  $\Sigma$  is a non-empty finite set of letters,  $\Sigma^*$  is the set of finite words over  $\Sigma$ ,  $\Sigma^i$  the set of words of length  $i$ , and  $\Sigma^\omega$  the set of infinite words. The empty word is denoted by  $\varepsilon$  and the length of a finite word  $w$  by  $|w|$ . For  $w \in \Sigma^* \cup \Sigma^\omega$  we write  $w(i)$  for the  $i$ -th letter of  $w$ . Given two infinite words  $\alpha \in \Sigma_I^\omega$  and  $\beta \in \Sigma_O^\omega$  we write  $\binom{\alpha}{\beta}$  for the word  $\binom{\alpha(0)}{\beta(0)} \binom{\alpha(1)}{\beta(1)} \binom{\alpha(2)}{\beta(2)} \cdots \in (\Sigma_I \times \Sigma_O)^\omega$ . Analogously, we write  $\binom{x}{y}$  for finite words  $x$  and  $y$ , provided they are of equal length.

**Parity Games.** An arena  $\mathcal{A}$  is a tuple  $(V, V_I, V_O, E)$ , where  $(V, E)$  is a finite directed graph without terminal vertices and  $\{V_I, V_O\}$  is a partition of  $V$  into the positions of Player  $I$  and Player  $O$ . A parity game  $\mathcal{G} = (\mathcal{A}, \Omega)$  consists of an arena  $\mathcal{A}$  with vertex set  $V$  and of a priority function  $\Omega: V \rightarrow \mathbb{N}$ . A play  $\rho$  is an infinite sequence  $v_0 v_1 v_2 \cdots$  of vertices such that  $(v_i, v_{i+1}) \in E$  for all  $i$ . A strategy for Player  $O$  is a map  $\sigma: V^* V_O \rightarrow V$  such that  $(v_i, \sigma(v_0 \cdots v_i)) \in E$  for all  $v_i \in V_O$ . The strategy  $\sigma$  is positional, if  $\sigma(wv) = \sigma(v)$  for all  $wv \in V^* V_O$ . Hence, we denote it as mapping from  $V_O$  to  $V$ . A play  $v_0 v_1 v_2 \cdots$  is consistent with  $\sigma$ , if  $v_{i+1} = \sigma(v_0 \cdots v_i)$  for every  $i$  with  $v_i \in V_O$ . The strategy  $\sigma$  is winning from a vertex  $v \in V$ , if every play  $v_0 v_1 v_2 \cdots$  with  $v_0 = v$  that is consistent with  $\sigma$  satisfies the parity condition, i.e., the maximal priority appearing infinitely often in  $\Omega(v_0)\Omega(v_1)\Omega(v_2)\cdots$  is even. The definition of (winning) strategies for Player  $I$  is dual. Parity games are positionally determined [5, 20], i.e., from every vertex one of the players has a positional winning strategy.

**Delay Games.** A delay function is a mapping  $f: \mathbb{N} \rightarrow \mathbb{N}_+$ , which is said to be constant, if  $f(i) = 1$  for every  $i > 0$ . Given a winning condition  $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$  and a delay function  $f$ , the game  $\Gamma_f(L)$  is played by two players, Player  $I$  and Player  $O$ , in rounds  $i = 0, 1, 2, \dots$  as follows: in round  $i$ , Player  $I$  picks a word  $u_i \in \Sigma_I^{f(i)}$ , then Player  $O$  picks one letter  $v_i \in \Sigma_O$ . We refer to the sequence  $(u_0, v_0), (u_1, v_1), (u_2, v_2), \dots$  as a play of  $\Gamma_f(L)$ . Player  $O$  wins the play if the outcome  $\binom{u_0 u_1 u_2 \cdots}{v_0 v_1 v_2 \cdots}$  is in  $L$ , otherwise Player  $I$  wins.

Given a delay function  $f$ , a strategy for Player  $I$  is a mapping  $\tau_I: \Sigma_O^* \rightarrow \Sigma_I^*$  where  $|\tau_I(w)| = f(|w|)$ , and a strategy for Player  $O$  is a mapping  $\tau_O: \Sigma_I^* \rightarrow \Sigma_O$ . Consider a play  $(u_0, v_0), (u_1, v_1), (u_2, v_2), \dots$  of  $\Gamma_f(L)$ . Such a play is consistent with  $\tau_I$ , if  $u_i = \tau_I(v_0 \cdots v_{i-1})$  for every  $i \in \mathbb{N}$ . It is consistent with  $\tau_O$ , if  $v_i = \tau_O(u_0 \cdots u_i)$  for every  $i \in \mathbb{N}$ . A strategy  $\tau$  for Player  $P \in \{I, O\}$  is winning, if every play that is consistent with  $\tau$  is winning for Player  $P$ . We say that a player wins  $\Gamma_f(L)$ , if she has a winning strategy.

**Prompt LTL.** Fix a set AP of atomic propositions. PROMPT-LTL formulas are given by

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X} \varphi \mid \mathbf{F} \varphi \mid \mathbf{G} \varphi \mid \varphi \mathbf{U} \varphi \mid \varphi \mathbf{R} \varphi \mid \mathbf{F_P} \varphi,$$

where  $p \in \text{AP}$ . We use  $\varphi \rightarrow \psi$  as shorthand for  $\neg \varphi \vee \psi$ , where we require  $\varphi$  to be a  $\mathbf{F_P}$ -free formula (for which the negation can be pushed to the atomic propositions using the dualities of the classical temporal operators). The size  $|\varphi|$  of  $\varphi$  is the number of subformulas of  $\varphi$ .

The satisfaction relation is defined for an  $\omega$ -word  $w \in (2^{\text{AP}})^\omega$ , a position  $i$  of  $w$ , a bound  $k$  for the prompt eventually operators, and a PROMPT-LTL formula. The definition is standard for the classical operators and defined as follows for the prompt eventually:

$$(w, i, k) \models \mathbf{F_P} \varphi \text{ if, and only if, there exists a } j \text{ with } 0 \leq j \leq k \text{ such that } (w, i + j, k) \models \varphi.$$

For the sake of brevity, we write  $(w, k) \models \varphi$  instead of  $(w, 0, k) \models \varphi$ . Note that  $\varphi$  is an LTL formula [21], if it does not contain the prompt eventually operator. Then, we write  $w \models \varphi$ .



**The Alternating-color Technique.** Let  $p \notin \text{AP}$  be a fixed fresh proposition. An  $\omega$ -word  $w' \in (2^{\text{AP} \cup \{p\}})^\omega$  is a  $p$ -coloring of  $w \in (2^{\text{AP}})^\omega$  if  $w'(i) \cap \text{AP} = w(i)$  for all  $i$ .

A position  $i$  of a word in  $(2^{\text{AP} \cup \{p\}})^\omega$  is a change point, if  $i = 0$  or if the truth value of  $p$  at positions  $i - 1$  and  $i$  differs. A  $p$ -block is an infix  $w'(i) \cdots w'(i + j)$  of  $w'$  such that  $i$  and  $i + j + 1$  are adjacent change points. Let  $k \geq 1$ : we say that  $w'$  is  $k$ -spaced, if  $w$  has infinitely many changepoints and each  $p$ -block has length at least  $k$ ; we say that  $w'$  is  $k$ -bounded, if each  $p$ -block has length at most  $k$  (which implies that  $w'$  has infinitely many change points).

Given a PROMPT-LTL formula  $\varphi$ , let  $\text{rel}'(\varphi)$  denote the formula obtained by inductively replacing every subformula  $\mathbf{F}_p \psi$  by

$$(p \rightarrow (p \mathbf{U} (\neg p \mathbf{U} \text{rel}'(\psi)))) \wedge (\neg p \rightarrow (\neg p \mathbf{U} (p \mathbf{U} \text{rel}'(\psi))))$$

and let  $\text{rel}(\varphi) = \text{rel}'(\varphi) \wedge \mathbf{G} \mathbf{F} p \wedge \mathbf{G} \mathbf{F} \neg p$ , i.e., we additionally require infinitely many change points. Intuitively, instead of requiring  $\psi$  to be satisfied within a bounded number of steps,  $\text{rel}(\varphi)$  requires it to be satisfied within at most one change point. The relativization  $\text{rel}(\varphi)$  is an LTL formula of size  $\mathcal{O}(|\varphi|)$ . Kupferman et al. showed that  $\varphi$  and  $\text{rel}(\varphi)$  are “equivalent” on  $\omega$ -words which are bounded and spaced.

► **Lemma 1** ([18]). *Let  $\varphi$  be a PROMPT-LTL formula and  $k \in \mathbb{N}$ .*

1. *If  $(w, k) \models \varphi$ , then  $w' \models \text{rel}(\varphi)$  for every  $k$ -spaced  $p$ -coloring  $w'$  of  $w$ .*
2. *If  $w'$  is a  $k$ -bounded  $p$ -coloring of  $w$  such that  $w' \models \text{rel}(\varphi)$ , then  $(w, 2k) \models \varphi$ .*

### 3 Delay Games with Prompt-LTL Winning Conditions

In this section, we study delay games with PROMPT-LTL winning conditions. Player  $O$ 's goal in such games is to satisfy the winning condition  $\varphi$  with respect to a bound  $k$  which is uniform among all plays consistent with the strategy. We show that such games are reducible to delay games with LTL winning conditions by tailoring the alternating-color technique to delay games and integrating it into the algorithm for solving  $\omega$ -regular delay games [16].

Throughout this section, we fix a partition  $\text{AP} = I \cup O$  of the set of atomic propositions into input propositions  $I$  under Player  $I$ 's control and output propositions  $O$  under Player  $O$ 's control. Let  $\Sigma_I = 2^I$  and  $\Sigma_O = 2^O$ . Given  $\binom{\alpha}{\beta} \in (\Sigma_I \times \Sigma_O)^\omega$ , we write  $((\binom{\alpha}{\beta}), k) \models \varphi$  for  $((\alpha(0) \cup \beta(0)) (\alpha(1) \cup \beta(1)) (\alpha(2) \cup \beta(2)) \cdots, k) \models \varphi$ . Given  $\varphi$  and a bound  $k$ , we define  $L(\varphi, k) = \{(\binom{\alpha}{\beta}) \in (\Sigma_I \times \Sigma_O)^\omega \mid ((\binom{\alpha}{\beta}), k) \models \varphi\}$ . If  $\varphi$  is an LTL formula, then this language is independent of  $k$  and will be denoted by  $L(\varphi)$ .

A PROMPT-LTL delay game  $\Gamma_f(\varphi)$  consists of a delay function  $f$  and a PROMPT-LTL formula  $\varphi$ . We say that Player  $P \in \{I, O\}$  wins  $\Gamma_f(\varphi)$  for the bound  $k$ , if she wins  $\Gamma_f(L(\varphi, k))$ . If we are not interested in the bound itself, but only in the existence of some bound, then we also say that Player  $O$  wins  $\Gamma_f(\varphi)$ , if there is some  $k$  such that she wins  $\Gamma_f(\varphi)$  for  $k$ . If  $\varphi$  is an LTL formula, then we call  $\Gamma_f(\varphi)$  an LTL delay game. The winning condition  $L(\varphi)$  of such a game is  $\omega$ -regular and independent of  $k$ .

In this section, we solve the following decision problem: given a PROMPT-LTL formula  $\varphi$ , does Player  $O$  win  $\Gamma_f(\varphi)$  for some delay function  $f$ ? Furthermore, we obtain upper bounds on the necessary lookahead and the necessary bound  $k$ , which are complemented by matching lower bounds in the next section.

With all definitions at hand, we state our main theorem of this section.

► **Theorem 2.** *The following problem is in 3EXPTIME: given a PROMPT-LTL formula  $\varphi$ , does Player  $O$  win  $\Gamma_f(\varphi)$  for some delay function  $f$ ?*

**Proof.** We reduce PROMPT-LTL to LTL delay games using the alternating-color technique. To this end, we add the proposition  $p$ , which induces the coloring, to  $O$ , i.e., in a game with winning condition  $\text{rel}(\varphi)$  Player  $O$ 's alphabet is  $2^{O \cup \{p\}}$ . In Lemma 4, we prove that Player  $O$  wins  $\Gamma_f(\varphi)$  for some delay function  $f$  if, and only if, Player  $O$  wins  $\Gamma_f(\text{rel}(\varphi))$  for some delay function  $f$ . This equivalence proves our claim: Determining whether Player  $O$  wins a delay game (for some  $f$ ) whose winning condition is given by a deterministic parity automaton is EXPTIME-complete [16]. We obtain an algorithm with triply-exponential running time by constructing a doubly-exponential deterministic parity automaton recognizing  $L(\text{rel}(\varphi))$  and then running the exponential-time algorithm on it. ◀

Thus, it remains to prove the equivalence between the delay games with winning conditions  $\varphi$  and  $\text{rel}(\varphi)$ . The harder implication is the one from the LTL delay game to the PROMPT-LTL delay game. There is a straightforward extension of the solution to the delay-free case. There, one proves that a finite-state strategy for the LTL game with winning condition  $\text{rel}(\varphi)$  (which always exists, if Player  $O$  wins the game) only produces  $k$ -bounded outcomes, for some  $k$  that only depends on the size of the strategy. Hence, by projecting away the additional proposition  $p$  inducing the coloring, we obtain a winning strategy for the PROMPT-LTL game with winning condition  $\varphi$  with bound  $2k$  by applying Lemma 1.2.

Now, consider the case with lookahead: if Player  $O$  wins  $\Gamma_f(\text{rel}(\varphi))$ , which has an  $\omega$ -regular winning condition, then also  $\Gamma_{f'}(\text{rel}(\varphi))$  for some triply-exponential constant  $f'$  [16]. We can model  $\Gamma_{f'}(\text{rel}(\varphi))$  as a delay-free parity game of quadruply-exponential size by storing the lookahead explicitly in the state space of the parity game. A positional winning strategy in this parity game only produces  $k$ -bounded plays, where  $k$  is the size of the delay-free game, as the color has to change infinitely often. Hence, such a strategy can be turned into a winning strategy for Player  $O$  in  $\Gamma_{f'}(\varphi)$  with respect to some quadruply-exponential bound  $k$ . However, this naive approach is not optimal: we present a more involved construction that achieves a triply-exponential bound  $k$ . The problem with the aforementioned approach is that the decision to produce a change point depends on the complete lookahead. We show how to base this decision on an exponentially smaller abstraction of the lookaheads, which yields an asymptotically optimal bound  $k$ . To this end, we extend the construction underlying the algorithm for  $\omega$ -regular delay games [16] by integrating the alternating-color technique.

Intuitively, we assign to each  $w \in \Sigma_I^*$ , i.e., to each potential additional information Player  $O$  has access to due to the lookahead, the behavior  $w$  induces in a deterministic automaton  $\mathcal{A}$  accepting  $L(\text{rel}(\varphi))$ , namely the state changes induced by  $w$  and the most important color on these runs. We construct  $\mathcal{A}$  such that it keeps track of change points in its state space, which implies that they are part of the behavior of  $w$ . Then, we construct a parity game in which Player  $I$  picks such behaviors instead of concrete words over  $\Sigma_I$  and Player  $O$  constructs a run on suitable representatives. The resulting game is of triply-exponential size and a positional winning strategy for this game can be turned back into a winning strategy for  $\Gamma_f(\varphi)$  satisfying asymptotically optimal bounds on the initial lookahead and the bound  $k$ . Thus, we save one exponent by not explicitly considering the lookahead, but only its effects.

We first extend the construction of a delay-free parity game  $\mathcal{G}$  that has the same winner as  $\Gamma_f(\text{rel}(\varphi))$  from [16]. The extension is necessary to obtain a “small” bound  $k$  when applying the alternating-color technique, which turns a positional winning strategy for  $\mathcal{G}$  into a winning strategy for Player  $O$  in  $\Gamma_{f'}(\varphi)$  for some  $f'$ .

To this end, let  $\mathcal{A} = (Q, \Sigma_I \times \Sigma_O, q_I, \delta, \Omega)$  be a deterministic max-parity automaton<sup>1</sup>

<sup>1</sup> Recall that  $\Omega: Q \rightarrow \mathbb{N}$  is a coloring of the states and a run  $q_0q_1q_2 \dots$  is accepting, if the maximal color occurring infinitely often in  $\Omega(q_0)\Omega(q_1)\Omega(q_2) \dots$  is even. See, e.g., [11] for details.

recognizing  $L(\text{rel}(\varphi))$ . First, as in the original construction, we add a deterministic monitoring automaton to keep track of certain information of runs. In the  $\omega$ -regular case [16], this information is the maximal priority encountered during a run. Here, we additionally need to remember whether the input word contains a change point. Let  $T_0 = 2^{\{p\}}$  and  $T = T_0 \times \{0, 1\}$ . Furthermore, for  $(t, s) \in T$  and  $t' \in T_0$ , we define the update  $\text{upd}((t, s), t') \in T$  of  $(t, s)$  by  $t'$  to be  $(t', s')$ , where  $s' = 0$  if, and only if,  $s = 0$  and  $t = t'$ . Intuitively, the first component of a tuple in  $T$  stores the last truth value of  $p$  and the second component is equal to one if, and only if, there was a change point.

Now, we define the deterministic parity automaton  $\mathcal{T} = (Q_{\mathcal{T}}, \Sigma_I \times \Sigma_O, q_{\mathcal{T}}^I, \delta_{\mathcal{T}}, \Omega_{\mathcal{T}})$  with  $Q_{\mathcal{T}} = Q \times \Omega(Q) \times T$ ,  $q_{\mathcal{T}}^I = (q_I, \Omega(q_I), (t', 0))$  for some arbitrary  $t' \in T_0$ ,  $\Omega(q, m, t) = m$ , and  $\delta_{\mathcal{T}}((q, m, t), \binom{a}{b}) = (q', \max\{m, \Omega(q')\}, \text{upd}(t, b \cap \{p\}))$  with  $q' = \delta(q, \binom{a}{b})$ .

First, let us note that  $\mathcal{T}$  does indeed keep track of the information described above.

► **Remark 3.** Let  $w \in (\Sigma_I \times \Sigma_O)^+$  and let  $(q_0, m_0, t_0) \cdots (q_{|w|}, m_{|w|}, t_{|w|})$  be the run of  $\mathcal{T}$  on  $w$  starting in  $(q_0, m_0, t_0)$  such that  $m_0 = \Omega(q_0)$  and  $t_0 = (t'_0, 0)$  for some  $t'_0 \in T_0$ . Then,  $q_0 q_1 \cdots q_{|w|}$  is the run of  $\mathcal{A}$  on  $w$  starting in  $q_0$ ,  $m_{|w|} = \max\{\Omega(q_j) \mid 0 \leq j \leq |w|\}$ , and  $t_{|w|} = (t'_{|w|}, s_{|w|})$  such that  $t'_{|w|}$  is the color of the last letter of  $w$ , and such that  $s_{|w|} = 0$  if, and only if, all letters of  $w$  have color  $t'_0$ . In particular, if  $w$  is preceded by a word whose last letter has color  $t'_0$ , then there is a change point in  $w$  if, and only if,  $s_{|w|} = 1$ .

Next, we classify possible moves  $w \in \Sigma_I^*$  according to the behavior they induce on  $\mathcal{T}$ . Let  $\delta_{\mathcal{P}}: 2^{Q_{\mathcal{T}}} \times \Sigma_I \rightarrow 2^{Q_{\mathcal{T}}}$  denote the transition function of the power set automaton of the projection of  $\mathcal{T}$  to  $\Sigma_I$ , i.e.,  $\delta_{\mathcal{P}}(S, a) = \{\delta_{\mathcal{T}}(q, \binom{a}{b}) \mid q \in S \text{ and } b \in \Sigma_O\}$ . As usual, we define  $\delta_{\mathcal{P}}^*: 2^{Q_{\mathcal{T}}} \times \Sigma_I^* \rightarrow 2^{Q_{\mathcal{T}}}$  inductively via  $\delta_{\mathcal{P}}^*(S, \varepsilon) = S$  and  $\delta_{\mathcal{P}}^*(S, wa) = \delta_{\mathcal{P}}(\delta_{\mathcal{P}}^*(S, w), a)$ .

Let  $D \subseteq Q_{\mathcal{T}}$  be non-empty and let  $w \in \Sigma_I^+$ . We define the function  $r_w^D: D \rightarrow 2^{Q_{\mathcal{T}}}$  via

$$r_w^D(q, m, (t, s)) = \delta_{\mathcal{P}}^*(\{(q, \Omega(q), (t, 0))\}, w)$$

for every  $(q, m, (t, s)) \in D$ . Note that we use  $\Omega(q)$  and  $(t, 0)$  as the second and third component in the input for  $\delta_{\mathcal{P}}^*$ , not  $m$  and  $(t, s)$  from the input to  $r_w^D$ . This resets the tracking components of  $\mathcal{T}$ . If we have  $(q', m', (t', s')) \in r_w^D(q, m, (t, s))$ , then there is a word  $w'$  over  $\Sigma_I \times \Sigma_O$  whose projection to  $\Sigma_I$  is  $w$  and such that the run of  $\mathcal{A}$  processing  $w'$  from  $q$  has the maximal priority  $m'$ ,  $t'$  is the color of the last letter of  $w$ , and  $s'$  encodes the existence of change points in  $w'$ , as explained in Remark 3. Thus, this function captures the behavior induced by  $w$  on  $\mathcal{T}$ . We allow to restrict the domain of such a function, as we do not have to consider every possible state, only those that are reachable by the play prefix constructed thus far.

Let  $r: Q_{\mathcal{T}} \rightarrow 2^{Q_{\mathcal{T}}}$  be a partial function. We say that  $w$  is a witness for  $r$ , if  $r_w^{\text{dom}(r)} = r$ . Thus, we can assign a language  $W_r \subseteq \Sigma_I^*$  of witnesses to each such  $r$ . Let  $\mathfrak{R}$  denote the set of such functions  $r$  with infinite witness language  $W_r$ . If  $w$  is a witness of  $r \in \mathfrak{R}$ , then  $r$  encodes the state transformations induced by  $w$  in the projection of  $\mathcal{A}$  to  $\Sigma_I$  as well as the maximal color occurring on these runs and the existence of change points on these. The latter is determined by the letters projected away, but still stored explicitly in the state space of the automaton. Furthermore, as we require  $r \in \mathfrak{R}$  to have infinitely many witnesses, there are arbitrarily long words with the same behavior. On the other hand, the language  $W_r$  of witnesses of  $r$  is recognizable by a DFA of size  $2^{n^2}$  [16], where  $n$  is the size of  $\mathcal{T}$ . Hence, every  $r$  also has a witness of length at most  $2^{n^2}$ . This allows to replace *long* words  $w \in \Sigma_I^*$  by equivalent ones that are bounded exponentially in  $n$ .

Next, we define a delay-free parity game in which Player  $I$  picks functions  $r_i \in \mathfrak{R}$  while Player  $O$  picks states  $q_i$  such that there is a word  $w'_i$  in  $(\Sigma_I \times \Sigma_O)^*$  whose projection to  $\Sigma_I$  is a witness of  $r_i$  and such that  $w'_i$  leads  $\mathcal{T}$  from  $q_i$  to  $q_{i+1}$ . By construction, this property is

independent of the choice of the witness. Furthermore, to account for the delay, Player  $I$  is always two moves ahead. Thus, instead of picking explicit words over their respective alphabets, the players pick abstractions, Player  $I$  explicitly and Player  $O$  implicitly by constructing the run.

Formally we define the parity game  $\mathcal{G} = ((V, V_O, V_I, E), \Omega')$  where  $V = V_I \cup V_O$ ,  $V_I = \{v_I\} \cup \mathfrak{R} \times Q_{\mathcal{T}}$  with the designated initial vertex  $v_I$  of the game, and  $V_O = \mathfrak{R}$ . Further,  $E$  is the union of the following sets of edges: initial moves  $\{(v_I, r) \mid \text{dom}(r) = \{q_I^{\mathcal{T}}\}\}$  for Player  $I$ , regular moves  $\{((r, q), r') \mid \text{dom}(r') = r(q)\}$  for Player  $I$ , and moves  $\{(r, (r, q)) \mid q \in \text{dom}(r)\}$  for Player  $O$ . Finally,  $\Omega'(v) = m$ , if  $v = (r, (q, m, s)) \in \mathfrak{R} \times Q_{\mathcal{T}}$ , and zero otherwise.

This finishes the construction of the game  $\mathcal{G}$ . The following lemma states the relation between  $\mathcal{G}$  and the delay games with winning conditions  $\varphi$  and  $\text{rel}(\varphi)$  and implies the equivalence of the delay games with winning conditions  $\varphi$  and  $\text{rel}(\varphi)$ .

► **Lemma 4.** *Let  $n = |Q_{\mathcal{T}}|$ , where  $Q_{\mathcal{T}}$  is the set of states of  $\mathcal{T}$  as defined above.*

1. *If Player  $O$  wins  $\Gamma_f(\varphi)$  for some delay function  $f$ , then also  $\Gamma_f(\text{rel}(\varphi))$  for the same  $f$ .*
2. *If Player  $O$  wins  $\Gamma_f(\text{rel}(\varphi))$  for some delay function  $f$ , then also  $\mathcal{G}$ .*
3. *If Player  $O$  wins  $\mathcal{G}$ , then also  $\Gamma_f(\varphi)$  for the constant delay function  $f$  with  $f(0) = 2^{n^2+1}$  and some bound  $k \leq 2^{2n^2+2}$ .*

The automaton  $\mathcal{A}$  recognizing  $L(\text{rel}(\varphi))$  can be constructed such that  $|\mathcal{A}| \in 2^{2^{\mathcal{O}(|\varphi|)}}$ , which implies  $n \in 2^{2^{\mathcal{O}(|\varphi|)}}$ , using a standard construction for translating LTL into non-deterministic Büchi automata and then Schewe's determinization construction [24]. Applying all implications of Lemma 4 yields upper bounds on the necessary constant lookahead and on the necessary bound  $k$  on the scope of the prompt eventually operator.

► **Corollary 5.** *If Player  $O$  wins  $\Gamma_f(\varphi)$  for some delay function  $f$  and some  $k$ , then also for some constant delay function  $f$  with  $f(0) \in 2^{2^{\mathcal{O}(|\varphi|)}}$  and some  $k \in 2^{2^{\mathcal{O}(|\varphi|)}}$  simultaneously.*

## 4 Lower Bounds for LTL and Prompt-LTL Delay Games

We complement the upper bounds on the complexity of solving PROMPT-LTL delay games, on the necessary lookahead, and on the necessary bound  $k$  by proving tight lower bounds in all three cases. The former two bounds already hold for LTL.

All proofs share some similarities which we discuss first. In particular, they all rely on standard encodings of doubly-exponentially large numbers using *small* LTL formulas and the interaction between the players. Assume AP contains the propositions  $b_0, \dots, b_{n-1}, b_I, b_O$  and let  $w \in (2^{\text{AP}})^{\omega}$  and  $i \in \mathbb{N}$ . We interpret  $w(i) \cap \{b_0, \dots, b_{n-1}\}$  as binary encoding of a number in  $[0, 2^n - 1]$ , which we refer to as the address of position  $i$ . There is a formula  $\psi_{\text{inc}}$  of quadratic size in  $n$  such that  $(w, i) \models \psi_{\text{inc}}$  if, and only if,  $m + 1 \bmod 2^n = m'$ , where  $m$  is the address of position  $i$  and  $m'$  is the address of position  $i + 1$ . Now, let  $\psi_0 = \bigwedge_{j=0}^{n-1} \neg b_j \wedge \mathbf{G} \psi_{\text{inc}}$ . If  $w \models \psi_0$ , then the  $b_j$  form a cyclic addressing of the positions starting at zero, i.e., the address of position  $i$  is  $i \bmod 2^n$ . If this is the case, we define a block of  $w$  to be an infix that starts at a position with address zero and ends at the next position with address  $2^n - 1$ . We interpret the  $2^n$  bits  $b_I$  of a block as a number  $x$  in  $R = [0, 2^{2^n} - 1]$ . Similarly, we interpret the  $2^n$  bits  $b_O$  of a block as a number  $y$  from the same range  $R$ . Furthermore, there are *small* formulas that are satisfied at the start of the  $i$ -th block if, and only if,  $x_i = y_i$  ( $x_i < y_i$ , respectively). However, we cannot compare numbers from different blocks for equality with *small* formulas. Nevertheless, if  $x_i$  is unequal to  $x_{i'}$ , then there is a single bit that witnesses this, i.e., the bit is one in  $x_i$  if, and only if, it is zero in  $x_{i'}$ . We will check this by letting

one of the players specify the address of such a witness (but not the witness itself). The correctness of this claim is then verifiable by a *small* formula.

#### 4.1 Lower Bounds on Lookahead

Our first result concerns a triply-exponential lower bound on the necessary lookahead in LTL delay games, which matches the upper bound proven in the previous section. The exponential lower bound  $2^n$  on the necessary lookahead for  $\omega$ -regular delay games is witnessed by winning conditions over the alphabet  $1, \dots, n$ . These conditions require to remember letters and to compare them for equality and order [16]. Here, we show how to adapt the winning condition to the alphabet  $R$ , which yields a triply-exponential lower bound  $2^{|R|}$ . The main difficulty of the proof is the inability of small LTL formulas to compare letters from  $R$ . To overcome this, we exploit the interaction between the players of the game.

- **Theorem 6.** *For every  $n > 0$ , there is an LTL formula  $\varphi_n$  of size  $\mathcal{O}(n^2)$  such that*
- *Player  $O$  wins  $\Gamma_f(\varphi_n)$  for some delay function  $f$ , but*
  - *Player  $I$  wins  $\Gamma_f(\varphi_n)$  for every delay function  $f$  with  $f(0) \leq 2^{2^{2^n}}$ .*

**Proof.** Fix some  $n > 0$ . In the following, we measure all formula sizes in  $n$ . Furthermore, let  $I = \{b_0, \dots, b_{n-1}, b_I, \#\}$  and  $O = \{b_O, \rightarrow, \leftarrow\}$ . Assume  $(\frac{\alpha}{\beta}) \in (\Sigma_I \times \Sigma_O)^\omega$  satisfies  $\psi_0$  from above. Then,  $\alpha$  induces a sequence  $x_0x_1x_2 \dots \in R^\omega$  of numbers encoded by the bits  $b_I$  in each block. Similarly,  $\beta$  induces a sequence  $y_0y_1y_2 \dots \in R^\omega$ .

The winning condition is intuitively described as follows:  $x_i$  and  $x_{i'}$  with  $i < i'$  constitute a bad  $j$ -pair, if  $x_i = x_{i'} = j$  and  $x_{i''} < j$  for all  $i < i'' < i'$ . Every sequence  $x_0x_1x_2 \dots$  contains a bad  $j$ -pair, e.g., pick  $j$  to be the maximal number occurring infinitely often. In order to win, Player  $O$  has to pick  $y_0$  such that  $x_0x_1x_2 \dots$  contains a bad  $y_0$ -pair. It is known that this winning condition requires lookahead of length  $2^m$  for Player  $O$  to win, where  $m$  is the largest number that can be picked [16].

To specify this condition with a small LTL formula, we have to require Player  $O$  to copy  $y_0$  ad infinitum, i.e., to pick  $y_i = y_0$  for all  $i$ , and to mark the two positions constituting the bad  $y_0$ -pair. Furthermore, the winning condition allows Player  $I$  to mark one copy error introduced by Player  $O$  by specifying its address by a  $\#$  (which may appear anywhere in  $\alpha$ ). This forces Player  $O$  to implement the copying correctly and thus allows a small formula to check that Player  $O$  indeed marks a bad  $y_0$ -pair. Consider the following properties:

1.  $\#$  holds at most once. Player  $I$  uses  $\#$  to specify the address where he claims an error.
2.  $\rightarrow$  holds at exactly one position, which has to be the start of a block. Furthermore, we require the two numbers encoded by the propositions  $b_I$  and  $b_O$  within this block to be equal. Player  $O$  uses  $\rightarrow$  to denote the first component of a claimed bad  $j$ -pair.
3.  $\leftarrow$  holds at exactly one position, which has to be the start of a block and has to appear at a later position than  $\rightarrow$ . Again, we require the two numbers encoded by this block to be equal. Player  $O$  uses  $\leftarrow$  to denote the second component of the claimed bad  $j$ -pair.
4. For every block between the two marked blocks, we require the number encoded by the  $b_I$  to be strictly smaller than the number encoded by the  $b_O$ .
5. If there is a position  $i_\#$  marked by  $\#$ , then there are no two different positions  $i \neq i'$  such that the following two conditions are satisfied: the addresses of  $i$ ,  $i'$ , and  $i_\#$  are equal and  $b_O$  holds at  $i$  if, and only if,  $b_O$  does not hold at  $i'$ . Such positions witness an error in the copying process by Player  $O$ , which manifests itself in a single bit, whose address is marked by Player  $I$  at any time in the future.

Each of these properties  $i \in \{1, 2, 3, 4, 5\}$  can be specified by an LTL formula  $\psi_i$  of at most quadratic size. Now, let  $\varphi_n = (\psi_0 \wedge \psi_1) \rightarrow (\psi_2 \wedge \psi_3 \wedge \psi_4 \wedge \psi_5)$ . We show that Player  $O$  wins  $\Gamma_f(\varphi_n)$  for some triply-exponential constant delay function, but not for any smaller one.

Fix  $n' = 2^{2^n}$ . We begin by showing that Player  $O$  wins  $\Gamma_f(\varphi_n)$  for the constant delay function with  $f(0) = 2^n \cdot 2^{n'}$ . A simple induction shows that every word  $w \in R^*$  of length  $2^{n'}$  contains a bad  $j$ -pair for some  $j \in R$ . Thus, a move  $\Sigma_I^{f(0)}$  made by Player  $I$  in round 0 interpreted as sequence  $x_0 x_1 \cdots x_{2^{n'}-1} \in R^*$  contains a bad  $j$ -pair for some fixed  $j$ . Hence, Player  $O$ 's strategy  $\tau_O$  produces the sequence  $j^\omega$  and additionally marks the corresponding bad  $j$ -pair with  $\rightarrow$  and  $\leftarrow$ . Every outcome of a play that is consistent with  $\tau_O$  and satisfies  $\psi_0$  also satisfies  $\psi_2 \wedge \psi_3 \wedge \psi_4 \wedge \psi_5$ , as Player  $O$  correctly marks a bad  $j$ -pair and never introduces a copy-error. Hence,  $\tau_O$  is a winning strategy for Player  $O$ .

It remains to show that Player  $I$  wins  $\Gamma_f(\varphi_n)$ , if  $f(0) \leq 2^n \cdot (2^{n'} - 2) \geq 2^{n'} = 2^{2^{2^n}}$ . Let  $w_{n'} \in R^*$  be recursively defined via  $w_0 = 0$  and  $w_j = w_{j-1} j w_{j-1}$ . A simple induction shows that  $w_{n'}$  does not contain a bad  $j$ -pair, for every  $j \in R$ , and that  $|w_{n'}| = 2^{n'} - 1$ .

Consider the following strategy  $\tau_I$  for Player  $I$  in  $\Gamma_f(\varphi_n)$ :  $\tau$  ensures that  $\psi_0$  is satisfied by the  $b_j$ , which fixes them uniquely to implement a cyclic addressing starting at zero. Furthermore, he picks the  $b_I$ 's so that the sequence of numbers  $x_0 x_1 \cdots x_\ell$  he generates during the first  $2^n$  rounds is a prefix of  $w_{n'}$ . This is possible, as each  $x_i$  is encoded by  $2^n$  bits and by the choice of  $f(0)$ . As a response during the first  $2^n$  rounds, Player  $O$  determines some number  $y \in R$ . During the next rounds, Player  $I$  finishes  $w_{n'}$  and then picks some fixed  $x \neq y$  ad infinitum (while still implementing the cyclic addressing). In case Player  $O$  picks both markings  $\rightarrow$  and  $\leftarrow$  in way that is consistent with properties 2, 3, and 4 as above, let  $y_0 y_1 \cdots, y_i$  be the sequence of numbers picked by her up to and including the number marked by  $\leftarrow$ . If they are not all equal, then there is an address that witnesses the difference between two of these numbers. Player  $I$  then marks exactly one position with the same address using  $\#$ . If this is not the case, he never marks a position with  $\#$ .

Consider an outcome of a play that is consistent with  $\tau_I$  and let  $x_0 x_1 x_2 \cdots \in R^\omega$  and  $y_0 y_1 y_2 \cdots \in R^\omega$  be the sequences of numbers induced by the outcome. By definition of  $\tau_I$ , the antecedent  $\psi_0 \wedge \psi_1$  of  $\varphi_n$  is satisfied and  $x_0 x_1 x_2 \cdots = w_{n'} \cdot x^\omega$  for some  $x \neq y_0$ .

If Player  $O$  never uses her markers  $\rightarrow$  and  $\leftarrow$  in a way that satisfies  $\psi_2 \wedge \psi_3 \wedge \psi_4$ , then Player  $I$  wins the play, as it satisfies the antecedent of  $\varphi_n$ , but not the consequent. Thus, it remains to consider the case where the outcome satisfies  $\psi_2 \wedge \psi_3 \wedge \psi_4$ . Let  $y_0 y_1 \cdots y_i$  be the sequence of numbers picked by her up to and including the number marked by  $\leftarrow$ . Assume we have  $y_0 = y_1 = \cdots = y_i$ . Then,  $\rightarrow$  and  $\leftarrow$  specify a bad  $y_0$ -pair, as implied by  $\psi_2 \wedge \psi_3 \wedge \psi_4$  and the equality of the  $y_j$ . As  $w_{n'}$  does not contain a bad  $y_0$ -pair, we conclude  $y_0 = x$ . However,  $\tau_I$  ensures  $y_0 \neq x$ . Hence, our assumption is false, i.e., the  $y_j$  are not all equal. In this situation,  $\tau_I$  marks a position whose address witnesses this difference. This implies that  $\psi_5$  is not satisfied, i.e., the play is winning for Player  $I$ . Hence,  $\tau_I$  is winning for him.  $\blacktriangleleft$

## 4.2 Lower Bounds on the Bound $k$

Our next result is a lower bound on the necessary bound  $k$  in a PROMPT-LTL delay game, which is proven by a small adaption of the game constructed in the previous proof. The winning condition additionally requires Player  $O$  to use the mark  $\leftarrow$  at least once and  $k$  measures the number of rounds before Player  $O$  does so. It turns out Player  $I$  can enforce a triply-exponential  $k$ , which again matches the upper bound proven in the previous section.

- **Theorem 7.** *For every  $n > 0$ , there is a Prompt LTL formula  $\varphi'_n$  of size  $\mathcal{O}(n^2)$  such that*
- *Player  $O$  wins  $\Gamma_f(\varphi'_n)$  for some delay function  $f$  and some  $k$ , but*
  - *Player  $I$  wins  $\Gamma_f(\varphi'_n)$  for every delay function  $f$  and every  $k \leq 2^{2^{2^n}}$ .*



**Proof.** Let  $\varphi'_n = (\psi_0 \wedge \psi_1) \rightarrow (\psi_2 \wedge \psi_3 \wedge \psi_4 \wedge \psi_5 \wedge \mathbf{F_P} \leftarrow)$ , where the alphabets and the formulas  $\psi_i$  are as in the proof of Theorem 6.

Let  $k = f(0) = 2^n \cdot 2^{n'}$  with  $n' = 2^{2^n}$  as above. Then, the strategy  $\tau_O$  for Player  $O$  described in the proof of Theorem 6 is winning for  $\Gamma_f(\varphi_n)$  with bound  $k$ : it places both markers within the first  $f(0) = k$  positions, as it specifies a bad  $j$ -pair within this range.

Now, assume we have  $k < 2^n \cdot 2^{n'}$ , consider the strategy  $\tau_I$  for Player  $I$  as defined in the proof of Theorem 6, and recall that every outcome that is consistent with  $\tau_I$  starts with the sequence  $w_{n'}$  in the first component. Satisfying  $\psi_2 \wedge \psi_3 \wedge \psi_4 \wedge \psi_5$  against  $\tau_I$  requires Player  $O$  to mark a bad  $j$ -pair and to produce the sequence  $j^\omega$ . However, Player  $I$  does not produce a bad  $j$ -pair in the first  $k$  positions, i.e., the conjunct  $\mathbf{F_P} \leftarrow$  is not satisfied with respect to  $k$ . Hence,  $\tau_I$  is winning for Player  $I$  in  $\Gamma_f(\varphi_n)$  with bound  $k$ . ◀

### 4.3 Lower Bounds on Complexity

Our final result settles the complexity of solving PROMPT-LTL delay games. The triply-exponential algorithm presented in the previous section is complemented by proving the problem to be 3EXPTIME-complete, which even holds for LTL. The proof is a combination of techniques developed for the lower bound on the lookahead presented above and of techniques from the EXPTIME-hardness proof for solving delay games whose winning conditions are given by deterministic safety automata [16] and is presented in the full version [17].

► **Theorem 8.** *The following problem is 3EXPTIME-complete: given an LTL formula  $\varphi$ , does Player  $O$  win  $\Gamma_f(\varphi)$  for some delay function  $f$ ?*

## 5 Delay Games on Non-deterministic, Universal, and Alternating Automata

Finally, we argue that the lower bounds just proven for LTL delay games can be modified to solve open problems about  $\omega$ -regular delay games whose winning conditions are given by non-deterministic, universal, and alternating automata (note that non-determinism and universality are not dual here, as delay games are asymmetric).

Recall that solving delay games with winning conditions given by deterministic parity automata is EXPTIME-complete and that exponential constant lookahead is sufficient and in general necessary. These upper bounds yield doubly-exponential upper bounds on both complexity and lookahead for non-deterministic and universal parity automata via determinization, which incurs an exponential blowup. Similarly, we obtain triply-exponential upper bounds on both complexity and lookahead for alternating parity automata, as determinization incurs a doubly-exponential blowup in this case.

For alternating automata, these upper bounds are tight, as LTL can be translated into linearly-sized alternating automata (even with very weak acceptance conditions). Hence, the triply-exponential lower bounds proven in the previous section hold here as well.

To prove doubly-exponential lower bounds for the case of non-deterministic and universal automata, one has to modify the constructions presented in the previous section. Let us first consider the case of non-deterministic automata: to obtain a matching doubly-exponential lower bound on the necessary lookahead, we require Player  $I$  to produce an input sequence in  $\{0, 1\}^\omega$ , where we interpret every block of  $n$  bits as the binary encoding of a number in  $\{0, 1, \dots, 2^n - 1\}$ . In order to win, Player  $O$  also has to pick an encoding of a number  $j$  with her first  $n$  moves such that the sequence of numbers picked by Player  $I$  contains a bad  $j$ -pair. To allow the automaton to check the correctness of this pick, we require Player  $O$  to



Automaton type	complexity	lookahead
deterministic parity	EXPTIME-complete	exponential
non-deterministic parity	2EXPTIME-complete	doubly-exponential
universal parity	2EXPTIME-complete	doubly-exponential
alternating parity	3EXPTIME-complete	triply-exponential

■ **Figure 1** Overview of results for the  $\omega$ -regular case.

repeat the encoding of the number ad infinitum. Then, the automaton can guess and verify the two positions comprising the bad  $j$ -pair. Finally, to prevent Player  $O$  from incorrectly copying the encoding of  $j$  (which manifests itself in a single bit), we use the same marking construction as in the previous section: Player  $I$  can mark one position  $i$  by a  $\#$  to claim an error in some bit at position  $i \bmod n$ . The automaton can guess the value  $i \bmod n$  and verify that there is no such error (and that the guess was correct). Using similar ideas one can encode an alternating exponential space Turing machine proving the 2EXPTIME lower bound on the complexity for non-deterministic automata.

For universal automata, the constructions are even simpler, since we do not need the marking of Player  $I$ . Instead, we use the universality to check that Player  $O$  copies her pick  $j$  correctly. Altogether, we obtain the results presented in Figure 1, where careful analysis shows that the lower bounds already hold for weaker acceptance conditions than parity, e.g., safety and weak parity (the case of reachability acceptance is exceptional, as such games are PSPACE-complete for non-deterministic automata [16]).

## 6 Conclusion

We identified PROMPT-LTL as the first quantitative winning condition for delay games that retains the desirable qualities of  $\omega$ -regular delay games: in particular, bounded lookahead is sufficient to win PROMPT-LTL delay games and to determine the winner of such games is 3EXPTIME-complete. This complexity should be contrasted to that of delay-free LTL and PROMPT-LTL games, which are already 2EXPTIME-complete. We complemented the complexity result by giving tight triply-exponential bounds on the necessary lookahead and on the necessary bound  $k$  for the prompt eventually operator.

All our lower bounds already hold for LTL and therefore also for (very-weak) alternating Büchi automata, since LTL can be translated into such automata of linear size [10]. On the other hand, we obtained tight matching upper bounds: solving delay games on alternating automata is 3EXPTIME-complete and triply-exponential lookahead is in general necessary and always sufficient. Furthermore, our lower bounds can be modified to complete the picture in the  $\omega$ -regular case with regard to the branching mode of the specification automaton: solving delay games with winning conditions given by non-deterministic or universal automata is 2EXPTIME-complete and doubly-exponential lookahead is sufficient and in general necessary.

Finally, as usual for results based on the alternating-color technique, our results on PROMPT-LTL hold for the stronger logics PLTL [1], PLDL [6], and their variants with costs [26] as well.

---

## References

- 1 Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron Peled. Parametric temporal logic for “model measuring”. *ACM Trans. Comput. Log.*, 2(3):388–407, 2001.

- 2 Rajeev Alur and Salvatore La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Log.*, 5(1):1–25, 2004.
- 3 Mikołaj Bojańczyk. A bounding quantifier. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *CSL 2004*, volume 3210 of *LNCS*, pages 41–55. Springer, 2004.
- 4 Rüdiger Ehlers. Symbolic bounded synthesis. *Form. Method. Syst. Des.*, 40(2):232–262, 2012.
- 5 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS 1991*, pages 368–377. IEEE, 1991.
- 6 Peter Faymonville and Martin Zimmermann. Parametric linear dynamic logic. In Adriano Peron and Carla Piazza, editors, *GandALF 2014*, volume 161 of *EPTCS*, pages 60–73, 2014.
- 7 Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Antichains and compositional algorithms for LTL synthesis. *Form. Method. Syst. Des.*, 39(3):261–296, 2011.
- 8 Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *STTT*, 15(5-6):519–539, 2013.
- 9 Wladimir Fridman, Christof Löding, and Martin Zimmermann. Degrees of lookahead in context-free infinite games. In Marc Bezem, editor, *CSL 2011*, volume 12 of *LIPICs*, pages 264–276. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.
- 10 Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV 2001*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001.
- 11 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.
- 12 Michael Holtmann, Łukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. *LMCS*, 8(3), 2012.
- 13 Florian Horn, Wolfgang Thomas, Nico Wallmeier, and Martin Zimmermann. Optimal strategy synthesis for request-response games. *RAIRO – Theor. Inf. and Applic.*, 49(3):179–203, 2015.
- 14 Frederick A. Hosch and Lawrence H. Landweber. Finite delay solutions for sequential conditions. In *ICALP 1972*, pages 45–60, 1972.
- 15 Felix Klein and Martin Zimmermann. What are strategies in delay games? Borel determinacy for games with lookahead. In Stephan Kreutzer, editor, *CSL 2015*, volume 41 of *LIPICs*, pages 519–533. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.
- 16 Felix Klein and Martin Zimmermann. How much lookahead is needed to win infinite games? *LMCS*, 12(3), 2016.
- 17 Felix Klein and Martin Zimmermann. Prompt delay. *arXiv*, 1602.05045, 2016.
- 18 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Form. Method. Syst. Des.*, 34(2):83–103, 2009.
- 19 Orna Kupferman and Moshe Y. Vardi. Safriless decision procedures. In *FOCS 2005*, pages 531–542. IEEE Computer Society, 2005.
- 20 Andrzej Mostowski. Games with forbidden positions. Technical Report 78, University of Gdańsk, 1991.
- 21 Amir Pnueli. The temporal logic of programs. In *FOCS 1977*, pages 46–57. IEEE, 1977.
- 22 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL 1989*, pages 179–190. ACM Press, 1989.
- 23 Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *ICALP 1989*, volume 372 of *LNCS*, pages 652–671. Springer, 1989.
- 24 Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In Luca de Alfaro, editor, *FOSSACS 2009*, volume 5504 of *LNCS*, pages 167–181. Springer, 2009.
- 25 Martin Zimmermann. Optimal bounds in parametric LTL games. *Theor. Comput. Sci.*, 493:30–45, 2013.

## 43:14 Prompt Delay

- 26 Martin Zimmermann. Parameterized linear temporal logics meet costs: Still not costlier than LTL. In Javier Esparza and Enrico Tronci, editors, *GandALF 2015*, volume 193 of *EPTCS*, pages 144–157, 2015.
- 27 Martin Zimmermann. Delay games with WMSO+U winning conditions. *RAIRO – Theor. Inf. and Applic.*, 2016. To appear.

# Mean-Payoff Games on Timed Automata\*

Shibashis Guha<sup>1</sup>, Marcin Jurdziński<sup>2</sup>,  
Shankara Narayanan Krishna<sup>3</sup>, and Ashutosh Trivedi<sup>4</sup>

- 1 The Hebrew University of Jerusalem, Israel  
shibashis@cs.huji.ac.il
- 2 The University of Warwick, UK  
marcin@dcs.warwick.ac.uk
- 3 Indian Institute of Technology Bombay, India  
krishnas@cse.iitb.ac.in
- 4 University of Colorado Boulder, USA  
ashutosh.trivedi@colorado.edu

---

## Abstract

Mean-payoff games on timed automata are played on the infinite weighted graph of configurations of priced timed automata between two players – Player Min and Player Max – by moving a token along the states of the graph to form an infinite run. The goal of Player Min is to minimize the limit average weight of the run, while the goal of the Player Max is the opposite. Brenguier, Cassez, and Raskin recently studied a variation of these games and showed that mean-payoff games are undecidable for timed automata with five or more clocks. We refine this result by proving the undecidability of mean-payoff games with three clocks. On a positive side, we show the decidability of mean-payoff games on one-clock timed automata with binary price-rates. A key contribution of this paper is the application of dynamic programming based proof techniques applied in the context of average reward optimization on an uncountable state and action space.

**1998 ACM Subject Classification** F.1.1 Models of Computation

**Keywords and phrases** Timed Automata, Mean-Payoff Games, Controller-Synthesis

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.44

## 1 Introduction

The classical mean-payoff games [24, 13, 16, 4] are two-player zero-sum games that are played on weighted finite graphs, where two players – Max and Min – take turn to move a token along the edges of the graph to jointly construct an infinite play. The objectives of the players Max and Min are to respectively maximize and minimize the limit average reward associated with the play. Mean-payoff games are well-studied in the context of optimal controller synthesis in the framework of Ramadge-Wonham [22], where the goal of the game is to find a control strategy that maximises the average reward earned during the evolution of the system. Mean-payoff games enjoy a special status in verification, since  $\mu$ -calculus model checking and parity games can be reduced in polynomial-time to solving mean-payoff games. Mean-payoff objectives can also be considered as quantitative extensions [17] of classical Büchi objectives, where we are interested in the limit-average share of occurrences of

---

\* The work of Shankara Narayanan Krishna is partly supported by CEFIPRA project AVeRTS. The work of Ashutosh Trivedi is based on research sponsored by DARPA under agreement number FA8750-15-2-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.



accepting states rather than merely in whether or not infinitely many accepting states occur. For a broader discussion on quantitative verification, in general, and the transition from the classical qualitative to the modern quantitative interpretation of deterministic Büchi automata, we refer the reader to Henzinger’s excellent survey [17].

We study mean-payoff games played on an infinite configuration graph of timed automata. Asarin and Maler [3] were the first to study games on timed automata and they gave an algorithm to solve timed games with reachability time objective. Their work was later generalized and improved upon by Alur et al. [1] and Bouyer et al. [8]. Bouyer et al. [7, 5] also studied the more difficult average payoffs, but only in the context of scheduling, which in game-theoretic terminology corresponds to 1-player games. However, they left the problem of proving decidability of 2-player average reward games on priced timed automata open. Jurdziński and Trivedi [20] proved the decidability of the special case of average time games where all locations have unit costs. More recently, mean-payoff games on timed automata have been studied by Brenguier, Cassez and Raskin [10] where they consider average payoff per time-unit. Using the undecidability of *energy games* [9], they showed undecidability of mean-payoff games on weighted timed games with five or more clocks. They also gave a semi-algorithm to solve cycle-forming games on timed automata and characterized the conditions under which a solution of these games gives a solution for mean-payoff games.

On the positive side, we characterize general conditions under which dynamic programming based techniques can be used to solve the mean-payoff games on timed automata. As a proof-of-concept, we consider one-clock binary-priced timed games, and prove the decidability of mean-payoff games for this subclass. Our decidability result can be considered as the average-payoff analog of the decidability result by Brihaye et al. [11] for reachability-price games on timed automata. We strengthen the known undecidability results for mean-payoff games on timed automata in three ways: (i) we show that the mean-payoff games over priced timed games is undecidable for timed games with only three clocks; (ii) secondly, we show that undecidability can be achieved with binary price-rates; and finally, (iii) our undecidability results are applicable for problems where the average payoff is considered per move as well as for problems when it is defined per time-unit.

Howard [18, 21] introduced gain and bias optimality equations to characterize optimal average on one-player finite game arenas. Gain and bias optimality equations based characterization has been extended to two-player game arenas [14] as well as many subclasses of uncountable state and action spaces [12, 6]. The work of Bouyer et al. [6] is perhaps the closest to our approach – they extended optimality equations approach to solve games on hybrid automata with certain strong reset assumption that requires all continuous variables to be reset at each transition, which in the case of timed automata is akin to requiring all clocks to be reset at each transition. To the best of our knowledge, the exact decidability for timed games does not immediately follow from any previously known results.

Howard’s Optimality equations requires two variable per state: the gain of the state and the bias of the state. Informally speaking, the gain of a state corresponds to the optimal mean-payoff for games starting from that state, while the bias corresponds to the limit of transient sum of step-wise deviations from the optimal average. Hence, intuitively at a given point in a game, both players would prefer to first optimize the gain, and then choose to optimize bias among choices with equal gains. We give general conditions under which a solution of gain-bias equations for a finitary abstraction of timed games can provide a solution of gain-bias equations for the original timed game. For this purpose, we exploit a region-graph like abstraction of timed automata [19] called the boundary region abstraction (BRA). Our key contribution is the theorem that states that every solution of gain-bias

optimality equations for boundary region abstraction carries over to the original timed game, as long as for every region, the gain values are constant and the bias values are affine.

The paper is organized in the following manner. In Section 2 we describe mean-payoff games and introduce the notions of *gain* and *bias* optimality equations. This section also introduces mean-payoff games over timed automata and states the key results of the paper. Section 3 introduces the *boundary region abstraction* for timed automata and characterizes the conditions under which the solution of a game played over the boundary region abstraction can be lifted to a solution of mean payoff game over priced timed automata. In Section 4 we present the strategy improvement algorithm to solve optimality equations for mean-payoff games played over boundary region abstraction and connect them to solution of optimality equations over corresponding timed automata. Finally, Section 5 sketches the undecidability of mean-payoff games for binary-priced timed automata with three clocks.

## 2 Mean-Payoff Games on Timed Automata

We begin this section by introducing mean-payoff games on graphs with uncountably infinite vertices and edges, and show how, and under what conditions, gain-bias optimality equations characterize the value of mean-payoff games. We then set-up mean-payoff games for timed automata and state our key contributions.

### 2.1 Mean-Payoff Games

► **Definition 1** (Turn-Based Game Arena). A game arena  $\Gamma$  is a tuple  $(S, S_{\text{Min}}, S_{\text{Max}}, A, T, \pi)$  where  $S$  is a (potentially uncountable) set of states partitioned between sets  $S_{\text{Min}}$  and  $S_{\text{Max}}$  of states controlled by Player Min and Player Max, respectively;  $A$  is a (potentially uncountable) set of *actions*;  $T : S \times A \rightarrow S$  is a partial function called the *transition function*; and  $\pi : S \times A \rightarrow \mathbb{R}$  is a partial function called the *price function*.

We say that a game arena is *finite* if both  $S$  and  $A$  are finite. For any state  $s \in S$ , we let  $A(s)$  denote the set of actions available in  $s$ , i.e., the actions  $a \in A$  for which  $T(s, a)$  and  $\pi(s, a)$  are defined. A transition of a game arena is a tuple  $(s, a, s') \in S \times A \times S$  such that  $s' = T(s, a)$  and we write  $s \xrightarrow{a} s'$ . A finite play starting at a state  $s_0$  is a sequence of transitions  $\langle s_0, a_1, s_1, a_2, \dots, s_n \rangle \in S \times (A \times S)^*$  such that for all  $0 \leq i < n$  we have that  $s_i \xrightarrow{a_{i+1}} s_{i+1}$  is a transition. For a finite play  $\rho = \langle s_0, a_1, \dots, s_n \rangle$  we write  $\text{Last}(\rho)$  for the final state of  $\rho$ , here  $\text{Last}(\rho) = s_n$ . The concept of an infinite play  $\langle s_0, a_1, s_1, \dots \rangle$  is defined in an analogous way. We write  $\text{Runs}(s)$  and  $\text{Runs}_{\text{fin}}(s)$  for the set of plays and the set of finite plays starting at  $s \in S$  respectively.

A *strategy* of Player Min is a function  $\mu : \text{Runs}_{\text{fin}} \rightarrow A$  such that  $\mu(\rho) \in A(\text{Last}(\rho))$  for all finite plays  $\rho \in \text{Runs}_{\text{fin}}$ , i.e. for any finite play, a strategy of Min returns an action available to Min in the last state of the play. A strategy  $\chi$  of Max is defined analogously and we let  $\Sigma_{\text{Min}}$  and  $\Sigma_{\text{Max}}$  denote the sets of strategies of Min and Max, respectively. A strategy  $\sigma$  is *positional* if  $\text{Last}(\rho) = \text{Last}(\rho')$  implies  $\sigma(\rho) = \sigma(\rho')$  for all  $\rho, \rho' \in \text{Runs}_{\text{fin}}$ . This allows us to represent a positional strategy as a function in  $[S \rightarrow A]$ . Let  $\Pi_{\text{Min}}$  and  $\Pi_{\text{Max}}$  denote the set of positional strategies of Min and Max, respectively. For any state  $s$  and strategy pair  $(\mu, \chi) \in \Sigma_{\text{Min}} \times \Sigma_{\text{Max}}$ , let  $\text{Run}(s, \mu, \chi)$  denote the unique infinite play  $\langle s_0, a_1, s_1, \dots \rangle$  in which Min and Max play according to  $\mu$  and  $\chi$ , respectively, i.e. for all  $i \geq 0$  we have that  $s_i \in S_{\text{Min}}$  implies  $a_{i+1} = \mu(\langle s_0, a_1, \dots, s_i \rangle)$  and  $s_i \in S_{\text{Max}}$  implies  $a_{i+1} = \chi(\langle s_0, a_1, \dots, s_i \rangle)$ .

In a mean-payoff game on a game arena, players Min and Max move a token along the transitions indefinitely thus forming an infinite play  $\rho = \langle s_0, a_1, s_1, \dots \rangle$  in the game graph.

The goal of player Min is to minimize  $\mathcal{A}_{\text{Min}}(\rho) = \limsup_{n \rightarrow \infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} \pi(s_i, a_{i+1})$  and the goal of player Max is to maximize  $\mathcal{A}_{\text{Max}}(\rho) = \liminf_{n \rightarrow \infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} \pi(s_i, a_{i+1})$ . The *upper value*  $\text{Val}^*(s)$  and the lower value  $\text{Val}_*(s)$  of a state  $s \in S$  are defined as:

$$\text{Val}^*(s) = \inf_{\mu \in \Sigma_{\text{Min}}} \sup_{\chi \in \Sigma_{\text{Max}}} \mathcal{A}_{\text{Min}}(\text{Run}(s, \mu, \chi)) \text{ and } \text{Val}_*(s) = \sup_{\chi \in \Sigma_{\text{Max}}} \inf_{\mu \in \Sigma_{\text{Min}}} \mathcal{A}_{\text{Max}}(\text{Run}(s, \mu, \chi))$$

respectively. It is always the case that  $\text{Val}_*(s) \leq \text{Val}^*(s)$ . A mean-payoff game is called *determined* if for every state  $s \in S$  we have that  $\text{Val}_*(s) = \text{Val}^*(s)$ . Then, we write  $\text{Val}(s)$  for this number and we call it the *value* of the mean-payoff game at state  $s$ . We say that a game is *positionally-determined* if for every  $\varepsilon > 0$  we have strategies  $\mu_\varepsilon \in \Pi_{\text{Min}}$  and  $\chi_\varepsilon \in \Pi_{\text{Max}}$  such that for every initial state  $s \in S$ , we have that

$$\text{Val}_*(s) - \varepsilon \leq \inf_{\mu' \in \Sigma_{\text{Min}}} \mathcal{A}_{\text{Max}}(\text{Run}(s, \mu', \chi_\varepsilon)) \text{ and } \text{Val}^*(s) + \varepsilon \geq \sup_{\chi' \in \Sigma_{\text{Max}}} \mathcal{A}_{\text{Min}}(\text{Run}(s, \mu_\varepsilon, \chi')).$$

For a given  $\varepsilon$  we call each such strategy an  $\varepsilon$ -optimal strategy for the respective player.

Given two functions  $G : S \rightarrow \mathbb{R}$  (gain) and  $B : S \rightarrow \mathbb{R}$  (bias), we say that  $(G, B)$  is a solution to the optimality equations for mean-payoff game on  $\Gamma = (S, S_{\text{Min}}, S_{\text{Max}}, A, T, \pi)$ , denoted  $(G, B) \models \text{Opt}(\Gamma)$  if

$$\begin{aligned} G(s) &= \begin{cases} \sup_{a \in A(s)} \{G(s') : s \xrightarrow{a} s'\} & \text{if } s \in S_{\text{Max}} \\ \inf_{a \in A(s)} \{G(s') : s \xrightarrow{a} s'\} & \text{if } s \in S_{\text{Min}}. \end{cases} \\ B(s) &= \begin{cases} \sup_{a \in A(s)} \{\pi(s, a) - G(s) + B(s') : s \xrightarrow{a} s' \text{ and } G(s) = G(s')\} & \text{if } s \in S_{\text{Max}} \\ \inf_{a \in A(s)} \{\pi(s, a) - G(s) + B(s') : s \xrightarrow{a} s' \text{ and } G(s) = G(s')\} & \text{if } s \in S_{\text{Min}}. \end{cases} \end{aligned}$$

We prove the following theorem connecting a solution of the optimality equations with mean-payoff games. We exploit this theorem to solve mean-payoff games on timed automata.

► **Theorem 2.** *If there exists a function  $G : S \rightarrow \mathbb{R}$  with finite image and a function  $B : S \rightarrow \mathbb{R}$  with bounded image such that  $(G, B) \models \text{Opt}(\Gamma)$  then for every state  $s \in S$ , we have that  $G(s) = \text{Val}(s)$  and for every  $\varepsilon > 0$  both players have positional  $\varepsilon$ -optimal strategies.*

**Proof.** Assume that we are given the functions  $G : S \rightarrow \mathbb{R}$  with finite image and  $B : S \rightarrow \mathbb{R}$  with bounded image such that  $(G, B) \models \text{Opt}(\Gamma)$ . In order to prove the result we show, for every  $\varepsilon > 0$ , the existence of positional strategies  $\mu_\varepsilon$  and  $\chi_\varepsilon$  such that

$$G(s) - \varepsilon \leq \inf_{\mu' \in \Sigma_{\text{Min}}} \mathcal{A}_{\text{Max}}(\text{Run}(s, \mu', \chi_\varepsilon)) \text{ and } G(s) + \varepsilon \geq \sup_{\chi' \in \Sigma_{\text{Max}}} \mathcal{A}_{\text{Min}}(\text{Run}(s, \mu_\varepsilon, \chi')).$$

The proof is in two parts.

- Given  $\varepsilon > 0$  we compute the positional strategy  $\mu_\varepsilon \in \Pi_{\text{Min}}$  satisfying the following conditions:  $\mu_\varepsilon(s) = a$  if

$$G(s) = G(s') \tag{1}$$

$$B(s) \geq \pi(s, a) - G(s) + B(s') - \varepsilon, \tag{2}$$

where  $s \xrightarrow{a} s'$ . Notice that it is always possible to find such strategy since  $(G, B)$  satisfies optimality equations and  $G$  is finite image.

Now consider an arbitrary strategy  $\chi \in \Sigma_{\text{Max}}$  and consider the run  $\text{Run}(s, \mu_\varepsilon, \chi) = \langle s_0, a_1, s_1, \dots, s_n, \dots \rangle$ . Notice that for every  $i \geq 0$  we have that  $G(s_i) \geq G(s_{i+1})$  if  $s_i \in S_{\text{Max}}$  and  $G(s_i) = G(s_{i+1})$  if  $s_i \in S_{\text{Min}}$ . Hence  $G(s_0), G(s_1), \dots$  is a non-increasing sequence. Since  $G$  is finite image, the sequence eventually becomes constant. Assume that for  $i \geq N$  we have that  $G(s_i) = g$ . Now notice that for all  $i \geq N$  we have that  $B(s_i) \geq \pi(s_i, a_{i+1}) - g + B(s_{i+1})$  if  $s_i \in S_{\text{Max}}$  and  $B(s_i) \geq \pi(s_i, a_{i+1}) - g + B(s_{i+1}) - \varepsilon$



if  $s_i \in S_{\text{Min}}$ . Summing these equations sidewise from  $i = N$  to  $N + k$  we have that  $B(s_N) \geq \sum_{i=N}^{N+k} \pi(s_i, a_{i+1}) - (k+1) \cdot g + B(s_{N+k+1}) - (k+1) \cdot \varepsilon$ . Rearranging, we get

$$g \geq \frac{1}{k+1} \sum_{i=N}^{N+k} \pi(s_i, a_{i+1}) + \frac{1}{k+1} (B(s_{N+k+1}) - B(s_N)) - \varepsilon.$$

Hence

$$\begin{aligned} g &\geq \limsup_{k \rightarrow \infty} \frac{1}{k+1} \sum_{i=N}^{N+k} \pi(s_i, a_{i+1}) + \limsup_{k \rightarrow \infty} \frac{1}{k+1} (B(s_{N+k+1}) - B(s_N)) - \varepsilon \\ &= \limsup_{k \rightarrow \infty} \frac{1}{k} \sum_{i=0}^k \pi(s_i, a_{i+1}) - \varepsilon. \end{aligned}$$

Hence  $G(s) + \varepsilon \geq \mathcal{A}_{\text{Min}}(\text{Run}(s, \mu_\varepsilon, \chi))$ . Since  $\chi$  is an arbitrary strategy in  $\Sigma_{\text{Max}}$ , we have  $G(s) + \varepsilon \geq \sup_{\chi' \in \Sigma_{\text{Max}}} \mathcal{A}_{\text{Min}}(\text{Run}(s, \mu_\varepsilon, \chi'))$ .

■ This part is analogous to the first part of the proof and is omitted.

The proof is now complete. ◀

## 2.2 Timed Automata

Priced Timed Game Arenas (PTGAs) extend classical timed automata [2] with a partition of the actions between two players Min and Max. Before we present the syntax and semantics of PTGAs, we need to introduce the concept of clock variables and related notions.

**Clocks.** Let  $\mathcal{X}$  be a finite set of *clocks*. A *clock valuation* on  $\mathcal{X}$  is a function  $\nu : \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$  and we write  $V(\mathcal{X})$  (or just  $V$  when  $\mathcal{X}$  is clear from the context) for the set of clock valuations. Abusing notation, we also treat a valuation  $\nu$  as a point in  $(\mathbb{R}_{\geq 0})^{|\mathcal{X}|}$ . Let  $\mathbf{0}$  denote the clock valuation that assigns 0 to all clocks. If  $\nu \in V$  and  $t \in \mathbb{R}_{\geq 0}$  then we write  $\nu + t$  for the clock valuation defined by  $(\nu + t)(c) = \nu(c) + t$  for all  $c \in \mathcal{X}$ . For  $C \subseteq \mathcal{X}$ , we write  $\nu[C := 0]$  for the valuation where  $\nu[C := 0](c)$  equals 0 if  $c \in C$  and  $\nu(c)$  otherwise. For  $X \subseteq V(\mathcal{X})$ , we write  $\overline{X}$  for the smallest closed set in  $V$  containing  $X$ . Although clocks are usually allowed to take arbitrary non-negative values, for notational convenience we assume that there is a  $K \in \mathbb{N}$  such that for every  $c \in \mathcal{X}$  we have  $\nu(c) \leq K$ .

**Clock Constraints.** A *clock constraint* over  $\mathcal{X}$  with upper bound  $K \in \mathbb{N}$  is a conjunction of *simple constraints* of the form  $c \bowtie i$  or  $c - c' \bowtie i$ , where  $c, c' \in \mathcal{X}$ ,  $i \in \mathbb{N}$ ,  $i \leq K$ , and  $\bowtie \in \{<, >, =, \leq, \geq\}$ . For  $\nu \in V(\mathcal{X})$  and  $K \in \mathbb{N}$ , let  $\text{CC}(\nu, K)$  be the set of clock constraints with upper bound  $K$  which hold in  $\nu$ , i.e. those constraints that resolve to **true** after substituting each occurrence of a clock  $x$  with  $\nu(x)$ .

**Regions and Zones.** Every clock region is an equivalence class of the indistinguishability-by-clock-constraints relation. For a given set of clocks  $\mathcal{X}$  and upper bound  $K \in \mathbb{N}$  on clock constraints, a *clock region* is a maximal set  $\zeta \subseteq V(\mathcal{X})$  such that  $\text{CC}(\nu, K) = \text{CC}(\nu', K)$  for all  $\nu, \nu' \in \zeta$ . For the set of clocks  $\mathcal{X}$  and upper bound  $K$  we write  $\mathcal{R}(\mathcal{X}, K)$  for the corresponding finite set of clock regions. We write  $[\nu]$  for the clock region of  $\nu$ . A *clock zone* is a convex set of clock valuations that satisfies constraints of the form  $\gamma ::= c_1 \bowtie k \mid c_1 - c_2 \bowtie k \mid \gamma \wedge \gamma$ ,  $k \in \mathbb{N}$ ,  $c_1, c_2 \in \mathcal{X}$  and  $\bowtie \in \{\leq, <, =, >, \geq\}$ . We write  $\mathcal{Z}(\mathcal{X}, K)$  for the set of clock zones over the set of clocks  $\mathcal{X}$  and upper bound  $K$ . When  $\mathcal{X}$  and  $K$  are clear from the context we write  $\mathcal{R}$  and  $\mathcal{Z}$  for the set of regions and zones. In this paper we fix a positive integer  $K$ , and work with  $K$ -bounded clocks and clock constraints.

### 2.3 Priced Timed Game Arena: Syntax and Semantics

► **Definition 3.** A priced timed game arena is a tuple  $\mathsf{T} = (L_{\text{Min}}, L_{\text{Max}}, \text{Act}, \mathcal{X}, \text{Inv}, E, \rho, \delta, p)$  where  $L_{\text{Min}}$  and  $L_{\text{Max}}$  are sets of *locations* controlled by Player Min and Player Max and we write  $L = L_{\text{Min}} \cup L_{\text{Max}}$ ;  $\text{Act}$  is a finite set of *actions*;  $\mathcal{X}$  is a finite set of *clocks*;  $\text{Inv} : L \rightarrow \mathcal{Z}$  is an *invariant condition*;  $E : L \times \text{Act} \rightarrow \mathcal{Z}$  is an *action enabledness function*;  $\rho : \text{Act} \rightarrow 2^{\mathcal{C}}$  is a *clock reset function*;  $\delta : L \times \text{Act} \rightarrow L$  is a *transition function*; and  $p : L \cup L \times \text{Act} \rightarrow \mathbb{R}$  is a *price information function*. A PTGA is binary-priced when  $p(\ell) \in \{0, 1\}$  for all  $\ell \in L$ .

When we consider a PTGA as an input of an algorithm, its size is understood as the sum of the sizes of encodings of  $L$ ,  $\mathcal{X}$ ,  $\text{Inv}$ ,  $\text{Act}$ ,  $E$ ,  $\rho$ ,  $\delta$  and  $p$ . We draw the states of Min players as circles, while states of Max player as boxes.

Let  $\mathsf{T} = (L_{\text{Min}}, L_{\text{Max}}, \text{Act}, \mathcal{X}, \text{Inv}, E, \rho, \delta, p)$  be a PTGA. A *configuration* of a PTGA is a pair  $(\ell, \nu)$ , where  $\ell$  is a location and  $\nu$  a clock valuation such that  $\nu \in \text{Inv}(\ell)$ . For any  $t \in \mathbb{R}_{\geq 0}$ , we let  $(\ell, \nu) + t$  equal the configuration  $(\ell, \nu + t)$ . In a configuration  $(\ell, \nu)$ , a timed action (time-action pair)  $(t, a)$  is available if and only if the invariant condition  $\text{Inv}(\ell)$  is continuously satisfied while  $t$  time units elapse, and  $a$  is enabled (i.e. the enabling condition  $E(\ell, a)$  is satisfied) after  $t$  time units have elapsed. Furthermore, if the timed action  $(t, a)$  is performed, then the next configuration is determined by the transition relation  $\delta$  and the reset function  $\rho$ , i.e. the clocks in  $\rho(a)$  are reset and we move to the location  $\delta(\ell, a)$ .

A game on a PTGA starts in an *initial configuration*  $(\ell, \nu) \in L \times V$  and players Min and Max construct an infinite play by taking turns to choose available timed actions  $(t, a)$  whenever the current location is controlled by them and the price  $p(\ell) \cdot t + p(\ell, a)$  is paid to the Max by player Min. Formally, PTGA semantics is given as a game arena.

► **Definition 4 (PTGA Semantics).** Let  $\mathsf{T} = (L_{\text{Min}}, L_{\text{Max}}, \text{Act}, \mathcal{X}, \text{Inv}, E, \rho, \delta, p)$  be a PTGA. The semantics of  $\mathsf{T}$  is given by game arena  $\llbracket \mathsf{T} \rrbracket = (S, S_{\text{Min}}, S_{\text{Max}}, A, T, \pi)$  where

- $S \subseteq L \times V$  is the set of states such that  $(\ell, \nu) \in S$  if and only if  $\nu \in \text{Inv}(\ell)$ ;
- $(\ell, \nu) \in S_{\text{Min}}$  (or  $(\ell, \nu) \in S_{\text{Max}}$ ) if  $(\ell, \nu) \in S$  and  $\ell \in L_{\text{Min}}$  (or  $\ell \in L_{\text{Max}}$ , respectively).
- $A = \mathbb{R}_{\geq 0} \times \text{Act}$  is the set of *timed actions*;
- $T : S \times A \rightarrow S$  is the transition function such that for  $(\ell, \nu) \in S$  and  $(t, a) \in A$ , we have  $T((\ell, \nu), (t, a)) = (\ell', \nu')$  if and only if
  - $\nu + t' \in \text{Inv}(\ell)$  for all  $t' \in [0, t]$ ;  $\nu + t \in E(\ell, a)$ ;  $(\ell', \nu') \in S$ ,  $\delta(\ell, a) = \ell'$ ,  $(\nu + t)[\rho(a) := 0] = \nu'$ .
- $\pi : S \times A \rightarrow \mathbb{R}$  is the reward function where  $\pi((\ell, \nu), (t, a)) = p(\ell) \cdot t + p(\ell, a)$ .

We are interested in the mean-payoff decision problem for timed automata  $\mathsf{T}$  that asks to decide whether the value of the mean-payoff game for a given state is below a given budget. For a PTGA  $\mathsf{T}$  and budget  $r \in \mathbb{R}$ , we write  $\text{MPG}(\mathsf{T}, r)$  for the  $r$ -mean payoff decision problem that asks whether the value of the game at the state  $(\ell, \mathbf{0})$  is smaller than  $r$ . The following theorem summarizes the key contribution of this paper.

► **Theorem 5.** *The decision problem  $\text{MPG}(\mathsf{T}, r)$  for binary-priced timed automata  $\mathsf{T}$  is undecidable for automata with three clocks, and decidable for automata with one clock.*

## 3 Boundary Region Graph Abstraction

In this section we introduce an abstraction of priced timed games called the boundary region abstraction (that generalizes classical corner-point abstraction [7]), and characterize conditions under which a solution of optimality equations for the boundary region abstraction can be lifted to a solution of optimality equations for timed automata. Observe that in order

to keep our result as general as possible, we present the abstraction and corresponding results for timed automata with an arbitrary number of clocks. In the following section, we show that the required conditions hold for the case of one-clock binary-priced timed automata.

**Timed Successor Regions.** Recall that  $\mathcal{R}$  is the set of clock regions. For  $\zeta, \zeta' \in \mathcal{R}$ , we say that  $\zeta'$  is in the future of  $\zeta$ , denoted  $\zeta \xrightarrow{*} \zeta'$ , if there exist  $\nu \in \zeta, \nu' \in \zeta'$  and  $t \in \mathbb{R}_{\geq 0}$  such that  $\nu' = \nu + t$  and say  $\zeta'$  is the *time successor* of  $\zeta$  if  $\nu + t' \in \zeta \cup \zeta'$  for all  $t' \leq t$  and write  $\zeta \rightarrow \zeta'$ , or equivalently  $\zeta' \leftarrow \zeta$ , to denote this fact. For regions  $\zeta, \zeta' \in \mathcal{R}$  such that  $\zeta \xrightarrow{*} \zeta'$  we write  $[\zeta, \zeta']$  for the zone  $\bigcup \{ \zeta'' \mid \zeta \xrightarrow{*} \zeta'' \wedge \zeta'' \xrightarrow{*} \zeta' \}$ .

**Thin and Thick Regions.** We say that a region  $\zeta$  is *thin* if  $[\nu] \neq [\nu + \varepsilon]$  for every  $\nu \in \zeta$  and  $\varepsilon > 0$  and *thick* otherwise. We write  $\mathcal{R}_{\text{Thin}}$  and  $\mathcal{R}_{\text{Thick}}$  for the sets of thin and thick regions, respectively. Observe that if  $\zeta \in \mathcal{R}_{\text{Thick}}$  then, for any  $\nu \in \zeta$ , there exists  $\varepsilon > 0$ , such that  $[\nu] = [\nu + \varepsilon]$  and the time successor of a thin region is thick, and vice versa.

**Intuition for the Boundary Region Graph (BRG).** Recall that  $K$  is an upper bound on clock values and let  $\llbracket K \rrbracket_{\mathbb{N}} = \{0, 1, \dots, K\}$ . For any  $\nu \in V, b \in \llbracket K \rrbracket_{\mathbb{N}}$  and  $c \in \mathcal{X}$ , we define  $\text{time}(\nu, (b, c)) \stackrel{\text{def}}{=} b - \nu(c)$  if  $\nu(c) \leq b$ , and  $\text{time}(\nu, (b, c)) \stackrel{\text{def}}{=} 0$  if  $\nu(c) > b$ . Intuitively,  $\text{time}(\nu, (b, c))$  returns the amount of time that must elapse in  $\nu$  before the clock  $c$  reaches the integer value  $b$ . Observe that, for any  $\zeta' \in \mathcal{R}_{\text{Thin}}$ , there exists  $b \in \llbracket K \rrbracket_{\mathbb{N}}$  and  $c \in \mathcal{X}$ , such that  $\nu \in \zeta$  implies  $(\nu + (b - \nu(c))) \in \zeta'$  for all  $\zeta \in \mathcal{R}$  in the past of  $\zeta'$  and write  $\zeta \rightarrow_{b,c} \zeta'$ . The boundary region abstraction is motivated by the following. Consider  $a \in \text{Act}, (\ell, \nu)$  and  $\zeta \xrightarrow{*} \zeta'$  such that  $\nu \in \zeta, [\zeta, \zeta'] \subseteq \text{Inv}(\ell)$  and  $\nu' \in E(\ell, a)$ . (For illustration, see Figure 2 in the appendix in [15]).

- If  $\zeta' \in \mathcal{R}_{\text{Thick}}$ , then there are infinitely many  $t \in \mathbb{R}_{\geq 0}$  such that  $\nu + t \in \zeta'$ . However, amongst all such  $t$ 's, for one of the boundaries of  $\zeta'$ , the closer  $\nu + t$  is to this boundary, the ‘better’ the timed action  $(t, a)$  becomes for a player’s objective. However, since  $\zeta'$  is a thick region, the set  $\{t \in \mathbb{R}_{\geq 0} \mid \nu + t \in \zeta'\}$  is an open interval, and hence does not contain its boundary values. Let the closest boundary of  $\zeta'$  from  $\nu$  be defined by the hyperplane  $c = b_{\text{inf}}$  and the farthest boundary of  $\zeta'$  from  $\nu$  be defined by the hyperplane  $c = b_{\text{sup}}$ .  $b_{\text{inf}}, b_{\text{sup}} \in \mathbb{N}$  are such that  $b_{\text{inf}} - \nu(c)$  ( $b_{\text{sup}} - \nu(c)$ ) is the infimum (supremum) of the time spent to reach the lower (upper) boundary of region  $\zeta'$ . Let the zones that correspond to these boundaries be denoted by  $\zeta'_{\text{inf}}$  and  $\zeta'_{\text{sup}}$  respectively. Then  $\zeta \rightarrow_{b_{\text{inf}}, c} \zeta'_{\text{inf}} \rightarrow \zeta'$  and  $\zeta \rightarrow_{b_{\text{sup}}, c} \zeta'_{\text{sup}} \leftarrow \zeta'$ . In the boundary region abstraction we include these ‘best’ timed actions through  $(b_{\text{inf}}, c, a, \zeta')$  and  $(b_{\text{sup}}, c, a, \zeta')$ .
- If  $\zeta' \in \mathcal{R}_{\text{Thin}}$ , then there exists a unique  $t \in \mathbb{R}_{\geq 0}$  such that  $\nu + t \in \zeta'$ . Moreover since  $\zeta'$  is a thin region, there exists a clock  $c \in \mathcal{C}$  and a number  $b \in \mathbb{N}$  such that  $\zeta \rightarrow_{b,c} \zeta'$  and  $t = b - \nu(c)$ . In the boundary region abstraction we summarise this ‘best’ timed action from region  $\zeta$  via region  $\zeta'$  through the action  $(b, c, a, \zeta')$ .

Based on this intuition above the boundary region abstraction (BRA) is defined as follows.

► **Definition 6.** For a priced timed game arena  $\mathsf{T} = (L_{\text{Min}}, L_{\text{Max}}, \text{Act}, \mathcal{X}, \text{Inv}, E, \rho, \delta, p)$  the boundary region abstraction of  $\mathsf{T}$  is given by the game arena  $\widehat{\mathsf{T}} = (\widehat{S}, \widehat{S}_{\text{Min}}, \widehat{S}_{\text{Max}}, \widehat{A}, \widehat{T}, \widehat{\pi})$

- $\widehat{S} \subseteq L \times V \times \mathcal{R}$  is the set of states such that  $(\ell, \nu, \zeta) \in \widehat{S}$  if and only if  $\zeta \subseteq \text{Inv}(\ell)$  and  $\nu \in \bar{\zeta}$  (recall that  $\bar{\zeta}$  denotes the closure of  $\zeta$ );
- $(\ell, \nu, \zeta) \in \widehat{S}_{\text{Min}}$  (or  $(\ell, \nu, \zeta) \in \widehat{S}_{\text{Max}}$ ) if  $(\ell, \nu, \zeta) \in \widehat{S}$  and  $\ell \in L_{\text{Min}}$  (or  $\ell \in L_{\text{Max}}$ , resp.);
- $\widehat{A} = (\llbracket K \rrbracket_{\mathbb{N}} \times \mathcal{X} \times \text{Act} \times \mathcal{R})$  is the set of actions;
- For  $\hat{s} = (\ell, \nu, \zeta) \in \widehat{S}$  and  $\alpha = (b_{\alpha}, c_{\alpha}, a_{\alpha}, \zeta_{\alpha}) \in \widehat{A}$ , function  $\widehat{T}(\hat{s}, \alpha)$  is defined if  $[\zeta, \zeta_{\alpha}] \subseteq \text{Inv}(\ell)$  and  $\zeta_{\alpha} \subseteq E(\ell, a_{\alpha})$  and it equals  $(\ell', \nu', \zeta') \in \widehat{S}$  where  $\delta(\ell, a_{\alpha}) = \ell', \nu_{\alpha}[C := 0] = \nu'$  and

- $\zeta_\alpha[C:=0] = \zeta'$  with  $\nu_\alpha = \nu + \text{time}(\nu, (b_\alpha, c_\alpha))$  and one of the following conditions holds:  
 $\zeta \rightarrow_{b_\alpha, c_\alpha} \zeta_\alpha$ ;  $\zeta \rightarrow_{b_\alpha, c_\alpha} \zeta_{\text{inf}} \rightarrow \zeta_\alpha$  for some  $\zeta_{\text{inf}} \in \mathcal{R}$ ;  $\zeta \rightarrow_{b_\alpha, c_\alpha} \zeta_{\text{sup}} \leftarrow \zeta_\alpha$  for some  $\zeta_{\text{sup}} \in \mathcal{R}$ ;  
 ■ for  $(\ell, \nu, \zeta) \in \widehat{S}$  and  $(b_\alpha, c_\alpha, a_\alpha, \zeta_\alpha) \in \widehat{A}$  the reward function  $\widehat{\pi}$  is given by:  
 $\widehat{\pi}((\ell, \nu, \zeta), (b_\alpha, c_\alpha, a_\alpha, \zeta_\alpha)) = p(\ell, a_\alpha) + p(\ell) \cdot (b_\alpha - \nu(c_\alpha))$

Although the boundary region abstraction is not a finite game arena, every state has only finitely many time successors (the boundaries of the regions) and for a fixed initial state we can restrict attention to a finite game arena due to the following observation.

► **Lemma 7** ([23]). *Let  $\mathsf{T}$  be a priced timed game arena and  $\widehat{\mathsf{T}}$  the corresponding BRA. For any state of  $\widehat{\mathsf{T}}$ , its reachable sub-graph is finite and can be constructed in time exponential in the size of  $\mathsf{T}$  when  $\mathsf{T}$  has more than one clock. For one clock  $\mathsf{T}$ , the reachable sub-graph of  $\widehat{\mathsf{T}}$  can be constructed in time polynomial in the size of  $\mathsf{T}$ . Moreover, the reachable sub-graph from the initial location and clock valuation is precisely the corner-point abstraction.*

### 3.1 Reduction to Boundary Region Abstraction

In what follows, unless specified otherwise, we fix a PTGA  $\mathsf{T} = (L_{\text{Min}}, L_{\text{Max}}, \text{Act}, \mathcal{X}, \text{Inv}, E, \rho, \delta, p)$  with semantics  $\llbracket \mathsf{T} \rrbracket = (S, S_{\text{Min}}, S_{\text{Max}}, A, T, \pi)$  and BRA  $\widehat{\mathsf{T}} = (\widehat{S}, \widehat{S}_{\text{Min}}, \widehat{S}_{\text{Max}}, \widehat{A}, \widehat{T}, \widehat{\pi})$ . Let  $G : \widehat{S} \rightarrow \mathbb{R}$  and  $B : \widehat{S} \rightarrow \mathbb{R}$  be such that  $(G, B) \models \text{Opt}(\widehat{\mathsf{T}})$ , i.e. for every  $\hat{s} \in \widehat{S}$  we have that

$$G(\hat{s}) = \begin{cases} \max_{\alpha \in \widehat{A}(\hat{s})} \{G(\hat{s}') : \hat{s} \xrightarrow{\alpha} \hat{s}'\} & \text{if } \hat{s} \in \widehat{S}_{\text{Max}} \\ \min_{\alpha \in \widehat{A}(\hat{s})} \{G(\hat{s}') : \hat{s} \xrightarrow{\alpha} \hat{s}'\} & \text{if } \hat{s} \in \widehat{S}_{\text{Min}}. \end{cases}$$

$$B(\hat{s}) = \begin{cases} \max_{\alpha \in \widehat{A}(\hat{s})} \{\pi(\hat{s}, \alpha) - G(\hat{s}) + B(\hat{s}') : \hat{s} \xrightarrow{\alpha} \hat{s}' \text{ and } G(\hat{s}) = G(\hat{s}')\} & \text{if } \hat{s} \in \widehat{S}_{\text{Max}} \\ \min_{\alpha \in \widehat{A}(\hat{s})} \{\pi(\hat{s}, \alpha) - G(\hat{s}) + B(\hat{s}') : \hat{s} \xrightarrow{\alpha} \hat{s}' \text{ and } G(\hat{s}) = G(\hat{s}')\} & \text{if } \hat{s} \in \widehat{S}_{\text{Min}}. \end{cases}$$

For a function  $F : \widehat{S} \rightarrow \mathbb{R}$  we define a function  $F^\boxplus : S \rightarrow \mathbb{R}$  as  $(\ell, \nu) \mapsto F(\ell, \nu, [\nu])$ . In this section we show under what conditions we can lift a solution  $(G, B)$  of optimality equations of BRA to  $(G^\boxplus, B^\boxplus)$  for priced timed game arena. Given a set of valuations  $X \subseteq V$ , a function  $f : X \rightarrow \mathbb{R}_{\geq 0}$  is *affine* if for any valuations  $\nu_x, \nu_y \in X$  we have that for all  $\lambda \in [0, 1]$ ,  $f(\lambda\nu_x + (1-\lambda)\nu_y) = \lambda f(\nu_x) + (1-\lambda)f(\nu_y)$ . We say that a function  $f : \widehat{S} \rightarrow \mathbb{R}_{\geq 0}$  is *regionally affine* if  $f(\ell, \cdot, \zeta)$  is affine over a region for all  $\ell \in L$  and  $\zeta \in \mathcal{R}$ , and  $f$  is *regionally constant* if  $f(\ell, \cdot, \zeta)$  is constant over a region for all  $\ell \in L$  and  $\zeta \in \mathcal{R}$ . Some properties of affine functions that are useful in the proof of the key lemma are given in Lemma 8.

► **Lemma 8.** *Let  $X \subseteq V$  and  $Y \subseteq \mathbb{R}_{\geq 0}$  be convex sets. Let  $f : X \rightarrow \mathbb{R}$  and  $w : X \times Y \rightarrow \mathbb{R}$  be affine functions. Then for  $C \subseteq \mathcal{X}$  we have that  $\phi_C(\nu, t) = w(\nu, t) + f((\nu + t)[C:=0])$  is also an affine function, and  $\inf_{t_1 < t < t_2} \phi_C(\nu, t) = \min\{\overline{\phi}_C(\nu, t_1), \overline{\phi}_C(\nu, t_2)\}$  and  $\sup_{t_1 < t < t_2} \phi_C(\nu, t) = \max\{\overline{\phi}_C(\nu, t_1), \overline{\phi}_C(\nu, t_2)\}$ ,  $\overline{\phi}$  is the unique continuous closure of  $\phi$ .*

► **Theorem 9.** *Let  $G : \widehat{S} \rightarrow \mathbb{R}$  and  $B : \widehat{S} \rightarrow \mathbb{R}$  are such that  $(G, B) \models \text{Opt}(\widehat{\mathsf{T}})$  and  $G$  is regionally constant and  $B$  is regionally affine, then  $(G^\boxplus, B^\boxplus) \models \text{Opt}(\mathsf{T})$ .*

**Proof.** We need to show that  $(G^\boxplus, B^\boxplus) \models \text{Opt}(\mathsf{T})$ , i.e. for every

$$G^\boxplus(s) = \begin{cases} \sup_{(t,a) \in A(s)} \{G^\boxplus(s') : s \xrightarrow{(t,a)} s'\} & \text{if } s \in S_{\text{Max}} \\ \inf_{(t,a) \in A(s)} \{G^\boxplus(s') : s \xrightarrow{(t,a)} s'\} & \text{if } s \in S_{\text{Min}}. \end{cases}$$

$$B^\boxplus(s) = \begin{cases} \sup_{(t,a) \in A(s)} \{\pi(s, (t,a)) - G^\boxplus(s) + B^\boxplus(s') : s \xrightarrow{(t,a)} s' \text{ and } G^\boxplus(s) = G^\boxplus(s')\} & \text{if } s \in S_{\text{Max}} \\ \inf_{(t,a) \in A(s)} \{\pi(s, (t,a)) - G^\boxplus(s) + B^\boxplus(s') : s \xrightarrow{(t,a)} s' \text{ and } G^\boxplus(s) = G^\boxplus(s')\} & \text{if } s \in S_{\text{Min}}. \end{cases}$$

Consider the case when  $s = (\ell, \nu) \in S_{\text{Min}}$  and consider the right side of the gain equations.

$$\begin{aligned}
& \inf_{(t,a) \in A(s)} \{G^{\boxplus}(s') : s \xrightarrow{(t,a)} s'\} \\
= & \min_{\substack{\zeta'' : [\nu] \rightarrow^* \zeta'' \\ [\zeta, \zeta''] \in \text{Inv}(\ell)}} \min_{a \in \text{Act}} \inf_{\substack{t: \\ \nu+t \in \zeta''}} \{G(\delta(\ell, a), (\nu+t)[\rho(a):=0], [(\nu+t)][\rho(a):=0])\} \\
= & \min_{\alpha \in \hat{A}(\ell, \nu, [\nu])} \{G(\ell', \nu', \zeta') : (\ell, \nu, \zeta) \xrightarrow{\alpha} (\ell', \nu', \zeta')\} = G(\ell, \nu, [\nu]) = G^{\boxplus}(\ell, \nu).
\end{aligned}$$

The first equality holds since  $(G, B) \models \text{Opt}(\hat{\mathbb{T}})$ . The second equality follows since  $G$  is regionally constant and hence it suffices to consider the delay  $\text{time}(\nu, (b, c))$  that corresponds to either left or right boundary of the region  $\zeta''$ , i.e. for fixed  $\nu, \zeta''$  and  $a \in \text{Act}$  we have that  $\inf_{\substack{t: \\ \nu+t \in \zeta''}} \{G(\ell', (\nu+t)[\rho(a):=0], \zeta')\} = G(\ell', \nu_{\alpha}[C:=0], \zeta')$  where  $\nu_{\alpha} = \nu + \text{time}(\nu, (b_{\alpha}, c_{\alpha}))$ ,  $\zeta''[C:=0] = \zeta'$  with  $\zeta \rightarrow_{b_{\alpha}, c_{\alpha}} \zeta''$  if  $\zeta''$  is thin, and  $\zeta \rightarrow_{b_{\alpha}, c_{\alpha}} \zeta_{\text{inf}} \rightarrow \zeta''$  for some  $\zeta_{\text{inf}} \in \mathcal{R}$  if  $\zeta''$  is thick. Similarly, for the bias equations, we need to show:

$$\begin{aligned}
& \inf_{\substack{t: \\ \nu+t \in \zeta''}} \{\pi((\ell, \nu), (t, a)) - G(\ell, \nu) + B(\ell', (\nu+t)[\rho(a):=0], \zeta')\} \\
= & \pi((\ell, \nu, [\nu]), (\text{time}(\nu, (b_{\alpha}, c_{\alpha})))) - G(\ell, \nu, [\nu]) + B(\ell', \nu_{\alpha}[C:=0], \zeta')
\end{aligned}$$

where  $\nu_{\alpha} = \nu + \text{time}(\nu, (b_{\alpha}, c_{\alpha}))$ ,  $\zeta''[C:=0] = \zeta'$  with  $\zeta \rightarrow_{b_{\alpha}, c_{\alpha}} \zeta''$  if  $\zeta''$  is thin; and  $\zeta \rightarrow_{b_{\alpha}, c_{\alpha}} \zeta_{\text{inf}} \rightarrow \zeta''$  for some  $\zeta_{\text{inf}} \in \mathcal{R}$  or  $\zeta \rightarrow_{b_{\alpha}, c_{\alpha}} \zeta_{\text{sup}} \rightarrow \zeta''$  for some  $\zeta_{\text{sup}} \in \mathcal{R}$  if  $\zeta''$  is thick. Given  $B$  is regionally affine (and hence linear in  $t$ ) and the price function is linear in  $t$ , the whole expression  $\pi((\ell, \nu), (t, a)) - G(\ell, \nu) + B(\ell', (\nu+t)[\rho(a):=0], \zeta')$  is linear in  $t$  and from Lemma 8 it attains its infimum or supremum on either boundary of the region.  $\blacktriangleleft$

## 4 Decidability for One Clock Binary-priced PTGA

Given the undecidability with 3 or more clocks, we focus on one clock PTGA. We provide a strategy improvement algorithm to compute a solution  $G : \hat{S} \rightarrow \mathbb{R}$  and  $B : \hat{S} \rightarrow \mathbb{R}$  of the optimality equations, i.e.  $(G, B) \models \text{Opt}(\hat{\mathbb{T}})$  for the BRA  $\hat{\mathbb{T}} = (\hat{S}, \hat{S}_{\text{Min}}, \hat{S}_{\text{Max}}, \hat{A}, \hat{T}, \hat{\pi})$  of one-clock binary-priced PTGAs with certain “integral payoff” restriction. Further, we show that for one clock binary-priced integral-payoff PTGA, the solution of optimality equations of corresponding BRG is such that the gains are regionally constant and biases are regionally affine. Hence by Theorem 9, the algorithm can be applied to solve mean-payoff games for one-clock binary-priced integral-payoff PTGAs. We also show how to lift the integral-payoff restriction to recover decidability for one-clock binary-priced PTGA.

**Regionally constant positional strategies.** Standard strategy improvement algorithms iterate over a finite set of strategies such that the value of the subgame at each iteration gets strictly improved. However, since there are infinitely many positional strategies in a boundary region abstraction, we focus on “regionally constant” positional strategies (RCPSs). We say that a positional strategy  $\mu : \hat{S} \rightarrow \hat{A}$  of player Min is regionally-constant if for all  $(\ell, \nu, \zeta), (\ell, \nu', \zeta) \in \hat{S}_{\text{Min}}$  we have that  $[\nu]=[\nu']$  implies that  $\mu(\ell, \nu, \zeta) = \mu(\ell, \nu', \zeta)$ . We similarly define RCPSs for player Max. In other words, in an RCPS a player chooses the same boundary action for every valuation of a region – as a side-result we show that optimal strategies for both players have this form. Observe that there are finitely many RCPSs for both players. We write  $\hat{\Pi}_{\text{Min}}$  and  $\hat{\Pi}_{\text{Max}}$  for the set of RCPSs for player Min and player Max, respectively. For a BRA  $\hat{\mathbb{T}}$ ,  $\chi \in \hat{\Pi}_{\text{Max}}$ , and  $\mu \in \hat{\Pi}_{\text{Min}}$  we write  $\hat{\mathbb{T}}(\chi)$  and  $\hat{\mathbb{T}}(\mu)$  for the “one-player” game on the sub-graph of BRAs where the strategies of player Max and Min

**Algorithm 1:** COMPUTEVALUEZEROPLAYER( $\mathbb{T}, \mu, \chi$ )

---

```

1 Consider  $\widehat{\mathbb{T}}(\mu, \chi)$  as a (single successor) weighted graph  $\mathcal{G} = (V, E, w)$  where
  ■  $V = L \times \mathcal{R} \times \mathcal{R}$  (with an order  $\preceq$ ) and  $E \subseteq V \times \widehat{A} \times V$ 
  ■  $(v_1, \alpha, v_2) \in E$  if  $v_1 = (\ell_1, \zeta_1, \zeta'_1)$ ,  $v_2 = (\ell_2, \zeta_2, \zeta'_2)$ , and  $\mu(\ell_1, \nu_1, \zeta'_1) = \alpha$  (or
     $\chi(\ell_1, \nu_1, \zeta'_1) = \alpha$ ) for all  $\nu_1 \in \zeta_1$  and  $(\ell_1, \nu_1, \zeta'_1) \xrightarrow{\alpha} (\ell_2, \nu_2, \zeta'_2)$  for some  $\nu_2 \in \zeta_2$ .
  ■  $w(v_1, \alpha, v_2)$  is the expression  $\nu \mapsto b_\alpha - \nu(c_\alpha)$ ;
for every cycle  $C$  of  $\mathcal{G}$  do
  | Let  $\text{Reach}(C)$  be set of vertices that reach  $C$ ;
  | Let  $\gamma$  be the average weight of the cycle ( $w$  is constant on cycles);
  | For every vertex  $V$  in  $\text{Reach}(C)$  set  $G(V) = \gamma$  and  $B(V) = \perp$ ;
  | For the smallest  $\preceq$ -vertex  $V_*$  in  $C$ . Set  $B(V_*) = 0$ ;
  | while there is  $V' \in \text{Reach}(C)$  with  $B(V') = \perp$  do
  | | Let  $(V', \alpha, V'') \in E$  with  $B(V'') \neq \perp$ ;
  | |  $B(V') := \nu \mapsto (w(V', \alpha, V''))(\nu) - G + B(V'')$ ;
return  $(G, B)$ ;

```

---

have been fixed to RCPSs  $\chi$  and  $\mu$ , respectively. Similarly we define the zero-player game  $\widehat{\mathbb{T}}(\mu, \chi)$  where strategies of both players are fixed to RCPSs  $\mu$  and  $\chi$ .

Let  $\widehat{\mathbb{T}}(\chi, \mu)$  be a zero-player game on the subgraph where strategies of player Max (and Min) are fixed to RCPSs  $\chi$  (and  $\mu$ ). Observe that for  $\widehat{\mathbb{T}}(\mu, \chi)$  the unique runs originating from states  $\hat{s}_0 = (\ell, \nu, \zeta)$  and  $\hat{s}'_0 = (\ell, \nu', \zeta)$  with  $[\nu] = [\nu']$  follow the same “lasso” after one step, i.e. the unique runs  $\hat{s}_0 \xrightarrow{\alpha_1} \hat{s}_1 \cdots \hat{s}_k \xrightarrow{\alpha_{k+1}} \cdots \hat{s}_{k+N-1} \xrightarrow{\alpha_{k+N}} \hat{s}_k^*$  and  $\hat{s}'_0 \xrightarrow{\alpha_1} \hat{s}'_1 \cdots \hat{s}'_k \xrightarrow{\alpha_{k+1}} \cdots \hat{s}'_{k+N-1} \xrightarrow{\alpha_{k+N}} \hat{s}'_k^*$  are such that for  $\hat{s}_i = (\ell_i, \nu_i, \zeta_i)$  and  $\hat{s}'_i = (\ell'_i, \nu'_i, \zeta'_i)$  we have that  $\ell_i = \ell'_i$ ,  $\zeta_i = \zeta'_i$  and  $\nu_i = \nu'_i$  for all  $i \in [1, k+N-1]$ . This is so because for one-clock timed automata the successors of the states  $\hat{s}_0 = (\ell, \nu, \zeta)$  and  $\hat{s}'_0 = (\ell, \nu', \zeta)$  for action  $\alpha_1 = (b, c, a, \zeta')$  is the same  $(\ell'', \nu'', \zeta'')$  where  $\nu''(c) = \nu(c) + (b - \nu(c)) = b = \nu'(c) + (b - \nu'(c))$  if  $c \notin \rho(a)$  and  $\nu''(c) = 0$  otherwise. Consider the optimality equations (See Appendix C.3 in [15]) for the lasso. Observe that the gain for the states  $\hat{s}_0, \dots, \hat{s}_{k+N-1}$  is the same, and let's call it  $g$ . If we add the bias equations side-wise for the cycle, we get  $g = \frac{1}{N} \sum_{i=0}^{N-1} \pi(\hat{s}_{k+i}, \alpha_{k+i+1})$ . It follows from the previous observation that the gains are regionally constant.

**Integral Payoff PTGA.** The gain in a zero-player game,  $\widehat{\mathbb{T}}(\chi, \mu)$ , although regionally-constant, may not be a whole number. We say that a PTGA is integral-payoff if for every pair  $(\mu, \chi) \in \widehat{\Pi}_{\text{Min}} \times \widehat{\Pi}_{\text{Max}}$  of RCPSs the gain as defined above is a whole number. Observe that the denominator in the gains correspond to the number of edges in a simple cycle of the BRA  $\widehat{\mathbb{T}}$ . If there are  $N$  simple cycles in the region graph of length  $n_1, n_2, \dots, n_N$ , then let  $\mathcal{L}$  be the least-common multiple of  $n_1, n_2, \dots, n_N$ . We multiply the constants appearing in the guards and invariants of the original PTGA  $\mathbb{T}$  by  $\mathcal{L}$  to obtain a PTGA  $\Upsilon_{\mathbb{T}}$ . It is easy to observe that mean-payoff of any state in  $\mathbb{T}$  is the mean-payoff in  $\Upsilon_{\mathbb{T}}$  divided by  $\mathcal{L}$ . For notational convenience, we assume that the given PTGA is an integral-payoff PTGA and hence for RCPS strategy profile  $(\mu, \chi)$  the gain is regionally constant and integral.

#### 4.1 Strategy Improvement Algorithm for Binary-Priced PTGA

Let  $\mathbb{T}$  be a one-clock integral-payoff binary-priced PTGA  $\mathbb{T}$  and  $\widehat{\mathbb{T}}$  be its boundary region graph. For a given RCPS profile  $(\mu, \chi) \in \widehat{\Pi}_{\text{Min}} \times \widehat{\Pi}_{\text{Max}}$ , Algorithm 1 computes the solution for the optimality equations  $\text{Opt}(\mathbb{T}(\mu, \chi))$ . This algorithm considers  $\widehat{\mathbb{T}}(\mu, \chi)$  as a graph whose



**Algorithm 2:** COMPUTEVALUETWOPLAYER( $\mathbb{T}$ )

---

```

1 Choose an arbitrary regionally constant positional strategy  $\chi' \in \Pi_{\text{Max}}$ ;
2 repeat
3    $\chi := \chi'$ ;
4   Choose an arbitrary regionally constant positional strategy  $\mu' \in \Pi_{\text{Min}}$ ;
5   repeat
6      $\mu := \mu'$ ;
7      $(G, B) := \text{COMPUTEVALUEZEROPLAYER}(\mathbb{T}, \mu, \chi)$ ;
8      $\mu' := \text{IMPROVEMINSTRATEGY}(\mathbb{T}, \mu, G, B)$ ;
9   until  $\mu = \mu'$ ;
10   $\chi' := \text{IMPROVEMAXSTRATEGY}(\mathbb{T}, \chi, G, B)$ ;
11 until  $\chi = \chi'$ ;
12 return  $(G, B)$ ;

```

---

vertices are “regions”  $(\ell, [\nu], \zeta)$  corresponding to state  $(\ell, \nu, \zeta) \in \widehat{S}$  of the boundary region graph, edges are boundary actions between them determined by the regionally constant strategy profile, and weight of an edge is the time function associated with the boundary action. Observe that every cycle in this graph will have constant weight on the edges since taking boundary actions in a loop will require going from an integral valuation to another integral valuation, and the average cost of such a cycle can be easily computed.

Also observe that, not unlike standard convention [21], our algorithm chooses a vertex in a cycle arbitrarily and fixes the bias of all of the states in that vertex to 0. This is possible since optimality equations over a cycle are underdetermined, and we exploit this flexibility to achieve solution to biases in a particularly “simple” structure. We say that a function  $f : \widehat{S} \rightarrow \mathbb{R}_{\geq 0}$  is regionally simple [3] if for all  $\ell \in L$ ,  $\zeta, \zeta' \in \mathcal{R}$  either i) there exists a  $d \in \mathbb{N}$  such that  $f(\ell, \nu, \zeta') = d$  for all  $\nu \in \zeta$ ; or ii) there exists  $d \in \mathbb{N}$  and  $c \in \mathcal{X}$  such that  $f(\ell, \nu, \zeta') = d - \nu(c)$  for all  $\nu \in \zeta$ . Key properties of regionally simple functions (Lemma 20 in Appendix C.2 in [15]) include that they are also regionally affine, closed under minimum and maximum, and if  $B : \widehat{S} \rightarrow \mathbb{R}$  is a regionally simple function and  $G : \widehat{S} \rightarrow \mathbb{N}$  is a regionally constant function, then  $\hat{s} \mapsto \pi(\hat{s}, \alpha) - G(\hat{s}) + B(\hat{s}')$ , with  $\hat{s} \xrightarrow{\alpha} \hat{s}'$ , is a regionally simple function. Using these properties and induction on the distance to  $\preceq$ -minimal element in the reachable cycle, we prove the correctness and following property of Algorithm 1.

► **Lemma 10.** *Algorithm 1 computes solution of optimality equations  $(G, B) \models \text{Opt}(\widehat{\mathbb{T}}(\mu, \chi))$  for  $\mu \in \widehat{\Pi}_{\text{Min}}$  and  $\chi \in \widehat{\Pi}_{\text{Max}}$ . Moreover,  $G$  is regionally constant and  $B$  is regionally simple.*

The strategy improvement algorithm to solve optimality equations is given as Algorithm 2. It begins by choosing an arbitrary regionally constant positional strategy  $\chi'$  and at every iteration of the loop (2–11) the algorithm computes (5–9) the value  $(G, B)$  of the current RCPS  $\chi$  and based on the value, the function `IMPROVEMAXSTRATEGY` returns an improved strategy by picking boundary action that lexicographically maximizes gain and bias respecting the policy that switches a decision only for a strict improvement. We formally define the function `IMPROVEMAXSTRATEGY` as follows: for  $\chi \in \widehat{\Sigma}_{\text{Max}}$ ,  $G : \widehat{S} \rightarrow \mathbb{R}$ , and  $B : \widehat{S} \rightarrow \mathbb{R}$  we let strategy `IMPROVEMAXSTRATEGY`( $\mathbb{T}, \chi, G, B$ ) be such that for all  $\hat{s} \in \widehat{S}_{\text{Max}}$  we have

$$\text{IMPROVEMAXSTRATEGY}(\mathbb{T}, \chi, G, B)(\hat{s}) = \begin{cases} \chi(\hat{s}) & \text{if } \chi(\hat{s}) \in M^*(\hat{s}, G, B) \\ \text{Choose}(M^*(\hat{s}, G, B)) & \text{Otherwise.} \end{cases}$$

where  $M^*(\hat{s}, G, B) = \text{argmax}_{\alpha \in \widehat{A}}^{\text{lex}} \{(G(\hat{s}'), \pi(\hat{s}, \alpha) - G(\hat{s}) + B(\hat{s}')) : \hat{s} \xrightarrow{\alpha} \hat{s}'\}$  and `Choose` picks an arbitrary element from a set. `IMPROVEMAXSTRATEGY` satisfies the following.



► **Lemma 11.** *If  $\chi \in \widehat{\Pi}_{Max}$ ,  $G$  is regionally constant, and  $B$  is regionally simple, then function  $\text{IMPROVEMAXSTRATEGY}(\mathbb{T}, \chi, G, B)$  returns a regionally constant positional strategy.*

The lines (5–9) compute the value of the strategy  $\chi$  of Player Max via a strategy improvement algorithm. This sub-algorithm works by starting with an arbitrary strategy of Player Min and computing the value  $(G, B)$  of the zero-player PTGA  $\widehat{\mathbb{T}}(\mu, \chi)$ . Based on the value, the function  $\text{IMPROVEMINSTRATEGY}$  returns an improved strategy of Min. The function  $\text{IMPROVEMINSTRATEGY}$  is defined as a dual of the function  $\text{IMPROVEMAXSTRATEGY}$  where  $\chi$  is replaced by  $\mu$  and  $\text{argmax}$  by  $\text{argmin}$ .  $\text{IMPROVEMINSTRATEGY}$  satisfies the following.

► **Lemma 12.** *If  $\mu \in \widehat{\Pi}_{Min}$ ,  $G$  is regionally constant, and  $B$  is regionally simple, then function  $\text{IMPROVEMINSTRATEGY}(\mathbb{T}, \mu, G, B)$  returns a regionally constant positional strategy.*

It follows from Lemma 11 and Lemma 12 that at every iteration of the strategy improvement the strategies  $\mu$  and  $\chi$  are RCPSs. Together with finiteness of the set of RCPSs and strict improvement at every step (see Lemmas 21 and 22 in [15] for the formal statements), we get following result.

► **Theorem 13.** *Algorithm 2 computes solution of optimality equations  $(G, B) \models \text{Opt}(\widehat{\mathbb{T}})$  for integral payoff PTGA  $\mathbb{T}$ . Moreover,  $G$  is regionally constant and  $B$  is regionally affine.*

This theorem – together with Theorem 9 and Theorem 2 – gives a proof of decidability for mean-payoff games for integral-payoff binary-priced one-clock timed automata.

## 5 Undecidability Results

► **Theorem 14.** *The mean-payoff problem  $\text{MPG}(\mathbb{T}, r)$  is undecidable for PTGA  $\mathbb{T}$  with 3 clocks having location-wise price-rates  $\pi(\ell) \in \{0, 1, -1\}$  for all  $\ell \in L$  and  $r = 0$ . Moreover, it is undecidable for binary-priced  $\mathbb{T}$  with 3 clocks and  $r > 0$ .*

**Proof.** We first show the undecidability result of the mean-payoff problem  $\text{MPG}(\mathbb{T}, 0)$  with location prices  $\{1, 0, -1\}$  and no edge prices. We prove the result by reducing the non-halting problem of 2 counter machines. Our reduction uses a PTGA with 3 clocks  $x_1, x_2, x_3$ , location prices  $\{1, 0, -1\}$ , and no edge prices. Each counter machine instruction (increment, decrement, zero check) is specified using a PTGA module. The main invariant in our reduction is that on entry into any module, we have  $x_1 = \frac{1}{5c_1 7^{c_2}}$ ,  $x_2 = 0$  and  $x_3 = 0$ , where  $c_1, c_2$  are the values of counters  $C_1, C_2$ . We outline the construction for the decrement instruction of counter  $C_1$  in Figure 2. For conciseness, we present here modules using arbitrary location prices. However, we can redraw these with extra locations and edges using only the location prices from  $\{1, 0, -1\}$  as shown for  $WD_1^1$  in Figure 1.

The role of the Min player is to faithfully simulate the two counter machine, by choosing appropriate delays to adjust the clocks to reflect changes in counter values. Player Max will have the opportunity to verify that player Min did not cheat while simulating the machine.

We enter location  $\ell_k$  with  $x_1 = \frac{1}{5c_1 7^{c_2}}$ ,  $x_2 = 0$  and  $x_3 = 0$ . Let's denote by  $x_{old}$  the value  $\frac{1}{5c_1 7^{c_2}}$ . To correctly decrement  $C_1$ , player Min should choose a delay of  $4x_{old}$  at location  $\ell_k$ . At location Check, there is no time elapse and player Max has three possibilities : (i) to go to  $\ell_{k+1}$  and continue the simulation, or (ii) to enter the widget  $WD_1^1$ , or (iii) to enter the widget  $WD_2^1$ . If player Min makes an error, and delays  $4x_{old} + \varepsilon$  or  $4x_{old} - \varepsilon$  at  $\ell_k$  ( $\varepsilon > 0$ ), then player Max can enter one of the widgets and punish player Min. Player Max enters widget  $WD_1^1$  if the error made by player Min is of the form  $4x_{old} + \varepsilon$  at  $\ell_k$  and enters widget  $WD_2^1$  if the error made by player Min is of the form  $4x_{old} - \varepsilon$  at  $\ell_k$ .

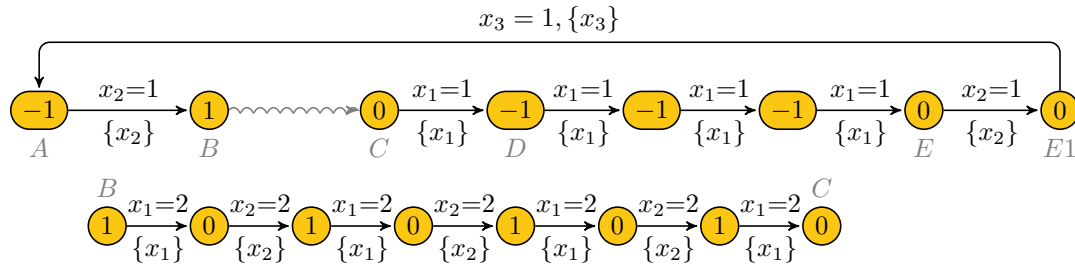


Figure 1  $WD_1^1$  redrawn with location prices from  $\{1, 0, -1\}$ . Every location has a self loop with the guard  $x_2, x_3 = 1$ , reset  $x_2, x_3$ , which is not shown here for conciseness. The curly edge from  $B$  to  $C$  is shown below. The mean-payoff incurred in one transit from  $A$  to  $A$  via  $E$  is  $\frac{\epsilon}{14}$ . If  $Min$  makes no error, this is 0.

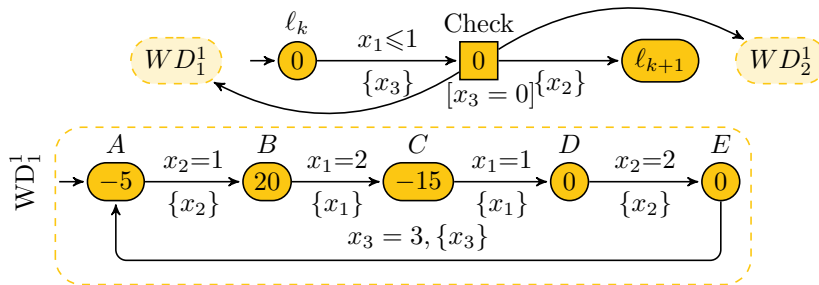


Figure 2 Simulation to decrement counter  $C_1$ , mean cost is  $\epsilon$  for error  $\epsilon$ . The widget  $WD_2^1$  has exactly the same structure and guards on all transitions as  $WD_1^1$ , but the price signs are reversed.

Let us examine the widget  $WD_1^1$ . When we enter  $WD_1^1$  for the first time, we have  $x_1 = x_{old} + 4x_{old} + \epsilon$ ,  $x_2 = 4x_{old} + \epsilon$  and  $x_3 = 0$ . In  $WD_1^1$ , the cost of going once from location  $A$  to  $E$  is  $5\epsilon$ . Also, when we get back to  $A$  after going through the loop once, the clock values with which we entered  $WD_1^1$  are restored; thus, each time, we come back to  $A$ , we restore the starting values with which we enter  $WD_1^1$ . The third clock is really useful for this purpose only. It can be seen that the mean cost of transiting from  $A$  to  $A$  through  $E$  is  $\epsilon$ . In a similar way, it can be checked that the mean cost of transiting from  $A$  to  $A$  through  $E$  in widget  $WD_2^1$  is  $\epsilon$  when player  $Min$  chooses a delay  $4x_{old} - \epsilon$  at  $\ell_k$ . Thus, if player  $Min$  makes a simulation error, player  $Max$  can always choose to goto one of the widgets, and ensure that the mean pay-off is not  $\leq 0$ . Note that when  $\epsilon = 0$ , then player  $Min$  will achieve his objective: the mean pay-off will be 0. Details of other gadgets are in Appendix D in [15].

For the  $MPG(T, r)$  problem ( $r > 0$ ) we reduce the non-halting problem by constructing a PTGA with 3 clocks and location prices in  $\{0, 1\}$  such that the meanpayoff is  $\leq \frac{1}{3}$  iff  $Min$  does a faithful simulation. Again, details can be found in Appendix D in [15]. ◀

In Appendix D.2 in [15], we show how this undecidability results extends (with the same parameters) if one defines mean payoff per time unit instead of per step. This way of averaging across time spent was considered in [10], where the authors show the undecidability of  $MPG(T, 0)$  with 5 clocks. We improve this result to show undecidability already in 3 clocks.

## References

- 1 R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In *Proc. of ICALP*, pages 122–133. Springer, 2004.
- 2 R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- 3 E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In F. W. Vaandrager and J. H. van Schuppen, editors, *Proc. of HSCC*, pages 19–30, 1999.
- 4 H. Björklund, S. Sandberg, and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. In *Proc. of MFCS*, pages 673–685, 2004.
- 5 P. Bouyer. Weighted timed automata: Model-checking and games. In *Proc. of MFPS*, volume 158, pages 3–17, 2006.
- 6 P. Bouyer, T. Brihaye, M. Jurdzinski, R. Lazic, and M. Rutkowski. Average-price and reachability-price games on hybrid automata with strong resets. In *FORMATS*, volume 5215 of *LNCS*, pages 63–77, 2008.
- 7 P. Bouyer, E. Brinksma, and K. G. Larsen. Staying alive as cheaply as possible. In *Proc. of HSCC*, volume 2993 of *LNCS*, pages 203–218. Springer, 2004.
- 8 P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In *Proc. of FSTTCS*, volume 3328 of *LNCS*, pages 148–160. Springer, 2004.
- 9 P. Bouyer, K. G. Larsen, and N. Markey. Lower-bound constrained runs in weighted timed automata. In *Proc. of QEST*, pages 128–137, 2012. doi:10.1109/QEST.2012.28.
- 10 R. Brenguier, F. Cassez, and J. F. Raskin. Energy and mean-payoff timed games. In *Proc. of HSCC*, pages 283–292, 2014. doi:10.1145/2562059.2562116.
- 11 T. Brihaye, G. Geeraerts, S. N. Krishna, L. Manasa, B. Monmege, and A. Trivedi. Adding negative prices to priced timed games. In *Proc. of CONCUR*, pages 560–575, 2014.
- 12 E. Dynkin and A. Yushkevich. *Controlled Markov Processes*. Springer, 1979.
- 13 A. Ehrenfeucht and A. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.
- 14 J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.
- 15 S. Guha, M. Jurdziński, S. N. Krishna, and A. Trivedi. Mean-payoff games on timed automata. *CoRR*, abs/1607.08480, 2016. URL: <http://arxiv.org/abs/1607.08480>.
- 16 V. A. Gurvich, A. V. Karzanov, and L. G. Khachiyan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Computational Mathematics and Mathematical Physics*, 28:85–91, 1988.
- 17 Thomas A. Henzinger. Quantitative reactive modeling and verification. *Computer Science – Research and Development*, 28(4):331–344, 2013.
- 18 R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.
- 19 M. Jurdziński and A. Trivedi. Reachability-time games on timed automata. In *Proc. of ICALP*, pages 838–849. Springer, 2007.
- 20 M. Jurdziński and A. Trivedi. Average-time games. In R. Hariharan, M. Mukund, and V. Vinay, editors, *Proc. of FSTTCS*, Dagstuhl Seminar Proceedings, 2008.
- 21 M. L. Puterman. *Markov Decision Processes: Disc. Stoc. Dynamic Prog.* Wiley, 1994.
- 22 P. J. Ramadge and W. M. Wonham. The control of discrete event systems. In *IEEE*, volume 77, pages 81–98, 1989.
- 23 A. Trivedi. *Competitive Optimisation on Timed Automata*. PhD thesis, Department of Computer Science, The University of Warwick, 2009.
- 24 U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.

# The Power and Limitations of Uniform Samples in Testing Properties of Figures\*

Piotr Berman<sup>1</sup>, Meiram Murzabulatov<sup>2</sup>, and Sofya Raskhodnikova<sup>3</sup>

1 Pennsylvania State University, University Park, USA

berman@cse.psu.edu

2 Pennsylvania State University, University Park, USA

mzm269@psu.edu

3 Pennsylvania State University, University Park, USA

sofya@cse.psu.edu

---

## Abstract

We investigate testing of properties of 2-dimensional figures that consist of a black object on a white background. Given a parameter  $\epsilon \in (0, 1/2)$ , a tester for a specified property has to accept with probability at least  $2/3$  if the input figure satisfies the property and reject with probability at least  $2/3$  if it does not. In general, property testers can query the color of any point in the input figure.

We study the power of testers that get access only to uniform samples from the input figure. We show that for the property of being a half-plane, the uniform testers are as powerful as general testers: they require only  $O(1/\epsilon)$  samples. In contrast, we prove that convexity can be tested with  $O(1/\epsilon)$  queries by testers that can make queries of their choice while uniform testers for this property require  $\Omega(1/\epsilon^{5/4})$  samples. Previously, the fastest known tester for convexity needed  $\Theta(1/\epsilon^{4/3})$  queries.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Property testing, randomized algorithms, being a half-plane, convexity

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.45

## 1 Introduction

We investigate testing of properties of 2-dimensional figures that consist of a black object and a white background. Sometimes the correctness of an algorithm depends on whether its input satisfies a certain property, e.g., it is a half-plane or a convex set. However, for a very large set, it is infeasible to determine whether it is indeed a half-plane or convex. How quickly is it possible to determine whether the input approximately satisfies the desired property? What access to the input is sufficient for this task?

Property testing [24, 14] studies algorithms that quickly determine whether the input has the desired property or it is *far* from having it. Many types of objects have been investigated in the property testing framework, including graphs [14, 12, 1], functions [7, 13, 10], distributions [3, 28], and geometric objects [9, 8]. In this work, we study properties of 2-dimensional figures.

A figure  $(U, C)$  consists of a compact convex universe  $U \subseteq \mathbb{R}^2$  and a measurable subset  $C \subseteq U$ . The set  $C$  can be thought of as a black object on a white background  $U \setminus C$ . A figure  $(U, C)$  is a *half-plane* if there is a line separating  $C$  from  $U \setminus C$ . A figure  $(U, C)$  is

---

\* M. M. and S. R. were supported by NSF award CCF-1422975.



© Piotr Berman, Meiram Murzabulatov, and Sofya Raskhodnikova;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 45; pp. 45:1–45:14



Leibniz International Proceedings in Informatics

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

*convex* iff  $C$  is convex. The relative distance between two figures  $(U, C)$  and  $(U, C')$  over the same universe is the probability of the symmetric difference between them under the uniform distribution on  $U$ . A figure  $(U, C)$  is  $\epsilon$ -far from a property (e.g., being a half-plane) if the relative distance from  $(U, C)$  to every figure  $(U, C')$  with the property over the same universe is at least  $\epsilon$ .

► **Definition 1.1.** Given a proximity parameter  $\epsilon \in (0, 1/2)$  and error probability  $\delta \in (0, 1)$ , an  $\epsilon$ -tester for a given property accepts with probability at least  $1 - \delta$  if the figure has the desired property and rejects with probability at least  $1 - \delta$  if the figure is  $\epsilon$ -far from the desired property<sup>1</sup>. A tester has *1-sided error* if it always accepts inputs with the property. (Otherwise, it has 2-sided error). A tester is *nonadaptive* if it makes all of its queries in advance, before seeing any of the input. A tester is *uniform* if it accesses its input only via uniform and independent samples from  $U$ , each labeled with a bit indicating whether it belongs to  $C$ .

In particular, a uniform tester is nonadaptive. In general, a tester can query the input at an arbitrary location. Such a strong assumption about the access model is not always realistic. Uniform testers, in contrast, rely only on uniform samples from the input. One advantage of using uniform testers is that they are *universal* in the following sense: we can collect uniform samples from the data in advance, before we know what property of the data needs to be tested.

Uniform testers were first considered by Goldreich, Goldwasser, and Ron [14] and systematically studied by Goldreich and Ron [15]. In particular, [15] shows that certain types of query-based testers yield uniform testers with sublinear (but dependent on size of the input) sample complexity.

In the context of property testing and sublinear algorithms, visual properties of 2-dimensional figures and discretized images have been studied in [21, 20, 23, 16, 17, 18, 6, 4, 5]. In [21], *adaptive*  $\epsilon$ -testers for the half-plane property and convexity were obtained. For the half-plane property, the query complexity<sup>2</sup> is  $O(1/\epsilon)$  and for convexity the query complexity is  $O(1/\epsilon^2)$ . Currently, the best  $\epsilon$ -tester known for convexity takes  $O(\epsilon^{-4/3})$  samples and is uniform [4]. This tester has 1-sided error, and every uniform 1-sided error tester for convexity needs  $\Omega(\epsilon^{-4/3})$  samples [4].

This motivates the following question: What is the power of uniform samples? Specifically, can we test the half-plane property with  $O(1/\epsilon)$  uniform samples? Can the best complexity for testing convexity be achieved by a uniform tester?

**Our results.** We show that for the property of being a half-plane, the uniform testers are as powerful as general testers: they require only  $O(1/\epsilon)$  samples. This is not the case for convexity. We prove that convexity can be tested with  $O(1/\epsilon)$  queries by testers that can make queries of their choice, improving the bound of  $O(\epsilon^{-4/3})$  in [4]. We also show that uniform testers for convexity, even with 2-sided error, require  $\Omega(\epsilon^{-5/4})$  samples.

**Connection to learning.** An upper bound  $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$  on the number of uniform samples for testing the half-plane property can be obtained from a connection between (proper)

<sup>1</sup> If  $\delta$  is not specified, it is assumed to be  $1/3$ . By standard arguments, the error probability can be reduced from  $1/3$  to an arbitrarily small  $\delta$  by running the tester  $O(\log 1/\delta)$  times.

<sup>2</sup> For any nontrivial property, including being a half-plane,  $\Omega(1/\epsilon)$  is an easy lower bound on the complexity of an  $\epsilon$ -tester.

PAC-learning and property testing, described in [14]. This bound follows from the fact that the VC dimension of the half-plane property is constant. Even though our tester has only slightly better sample complexity, its complexity is tight. Moreover, the running time of our tester is also optimal. (The running time cannot be obtained from the VC-dimension bound.) For convexity, PAC-learning under the uniform distribution requires  $\Theta(\epsilon^{-3/2})$  samples, as shown by Schmeltz [25]. (VC dimension of convexity is unbounded, so this result is specific to the uniform distribution.) For this property, however, as shown in [4], testing requires significantly fewer samples than learning when the object is accessed via uniform samples. Our tester for convexity can be viewed as an adaptive learner for the property, followed by a check that the learned convex object corresponds to the input.

**Our techniques.** Our tester for the half-plane property is the natural one: it checks whether the convex hull of sampled black points intersects the convex hull of sampled white points and rejects if it is the case. In other words, it rejects only if it finds a violation of the half-plane property. To analyze the tester, we use the notion of *black-central* and *white-central* points defined in terms of the Ham Sandwich cut of black (respectively, white) points. (These central points are related to the well studied centerpoints [11] and Tukey medians [27]. The guarantee for a centerpoint is that every line that passes through it creates a relatively balanced cut.) Such cuts have been studied extensively (see, e.g., [11, p. 356] and [19]), for example, in the context of range queries. Specifically, a *black-central* (respectively, *white-central*) point is the intersection of two lines that partition the figure into four regions, each with black (respectively, white) area<sup>3</sup> at least  $\epsilon/4$ . Black-central points were defined in [4] in order to analyze a tester of convexity of figures. A black-central (respectively, white-central) point is overwhelmingly likely to end up in the convex hull of sampled black (respectively, white) points. We show that if the figure is  $\epsilon$ -far from being a half-plane, the convex hull of its black-central points intersects the convex hull of its white-central points. A point in the intersection, even though is not likely to be sampled, is likely to be in the intersection of the convex hull of the black samples and the convex hull of the white samples. Thus, there is likely to be the intersection, and the tester is likely to reject.

Our tester for convexity samples points uniformly at random and constructs a rectangle  $R$  that with high probability contains nearly the entire black area and whose sides include sampled black points. Then it adaptively queries points of  $R$  in order to partition it into the candidate black and white regions, leaving only a small region unclassified. After completing this learning stage, it samples points in the classified regions and rejects iff it finds a mistake.

To prove our lower bound, we construct hard instances, for which every uniform tester needs to get a 2-point witness, with points coming from different specified regions, in order to distinguish between our hard instances that are convex from hard instances that are far from convex. The challenge here is to construct a figure with regions that can be manipulated independently to either keep convexity or to violate it.

## 2 The Uniform Tester for the Half-Plane Property

In this section, we give a uniform tester for the half-plane property.

---

<sup>3</sup> For the two properties we consider (being a half-plane and convexity), we assume w.l.o.g. that the input figure  $U$  has unit area. If it is not the case,  $U$  can be rescaled. Thus, the area of a region corresponds to the probability of sampling from it under the uniform distribution.



---

**Algorithm 1:** Uniform tester for the half-plane property.

---

**input** : parameter  $\epsilon \in (0, 1/2)$ ;  
access to uniform and independent samples from  $(U, C)$ .

- 1 Set  $s \leftarrow \frac{18}{\epsilon}$ . Sample  $s$  points from  $U$  uniformly and independently at random.
- 2 Set  $U$  is contained in a rectangle  $R$  whose area is at most twice the area of  $U$ . Orient  $U$ , so that  $R$  is axis-aligned.
- 3 Bucket sort sampled black pixels by the  $x$ -coordinate into  $s$  bins to obtain list  $S_B$ . Similarly, compute  $S_W$  for the sampled white pixels.

// Check if the convex hull of  $S_B$  contains a pixel from  $S_W$ .

- 4 Use Andrew's monotone chain convex hull algorithm [2] to compute  $\text{UH}(S_B)$  and  $\text{LH}(S_B)$ , the upper and the lower hulls of  $S_B$ , respectively, sorted by the  $x$ -coordinate.
- 5 Merge sorted lists  $S_W$ ,  $\text{UH}(S_B)$  and  $\text{LH}(S_B)$  to determine for each point  $w$  in  $S_W$  its left and right neighbors in  $\text{UH}(S_B)$  and  $\text{LH}(S_B)$ . If  $w$  lies between the corresponding line segments of the upper and lower hulls, **reject**.

// Check if the convex hull of  $S_W$  contains a point from  $S_B$ .

- 6 Repeat Steps 4–5 with the roles of  $S_B$  and  $S_W$  reversed.
- 7 **Accept**.

---

► **Theorem 2.1.** *There is a uniform (1-sided error)  $\epsilon$ -tester for the half-plane property of figures with sample and time complexity  $O(1/\epsilon)$ .*

**Proof.** Our uniform tester for the half-plane property is Algorithm 1. It takes  $O(1/\epsilon)$  uniform samples and checks if the sampled black and white points are linearly separable. We will show that the expected running time of Algorithm 1 is  $O(1/\epsilon)$  and its error probability is 0.3. A tester with worst case running time  $O(1/\epsilon)$  and error probability 1/3 can be obtained from Algorithm 1 by standard arguments.

Consider a half-plane figure  $(U, C)$ . Let  $S_B$  and  $S_W$  be the two lists obtained by Algorithm 1 in Step 3. It is easy to see that  $\text{Hull}(S_B)$  and  $\text{Hull}(S_W)$  do not intersect, i.e., they are linearly separable. Thus, the algorithm accepts the figure.

Now assume that  $(U, C)$  is  $\epsilon$ -far from being a half-plane. We prove that the algorithm rejects the figure with probability at least 2/3. We consider two sets of points in  $U$ : *black-central* and *white-central*. We show that if the figure is  $\epsilon$ -far from being a half-plane, then the convex hulls of the two sets intersect. In this case, the tester will detect this intersection, with probability at least 2/3, by only looking at the convex hull of sampled black points and the convex hull of sampled white points.

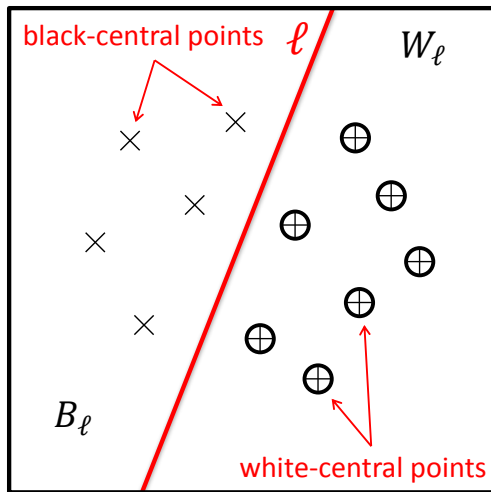
Next, we define white-central and black-central points. Black-central points were used in [4] to analyze a tester for convexity. In that work, they were called *central* points.

► **Definition 2.2** (White-central and black-central points). A point in the figure is *white-central* (respectively, *black-central*) if it is the intersection of two lines such that each of the quadrants formed by these lines has white (respectively, black) area at least  $\epsilon/4$ .

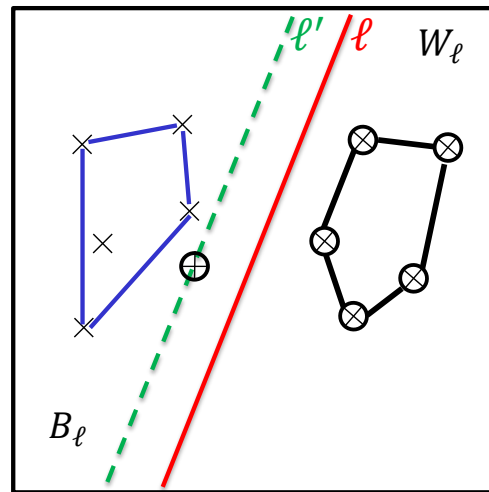
► **Lemma 2.3.** *There is no line that separates white-central points from black-central points in a figure that is  $\epsilon$ -far from being a half-plane.*

**Proof.** Let  $(U, C)$  be a figure that is  $\epsilon$ -far from being a half-plane. For the sake of contradiction, suppose there is a line  $\ell$  that separates white-central and black-central points in  $(U, C)$ , i.e., it partitions the figure into two regions,  $W_\ell$  and  $B_\ell$ , such that  $W_\ell$  contains only





■ **Figure 1** An illustration of black-central and white-central points separated by a line.

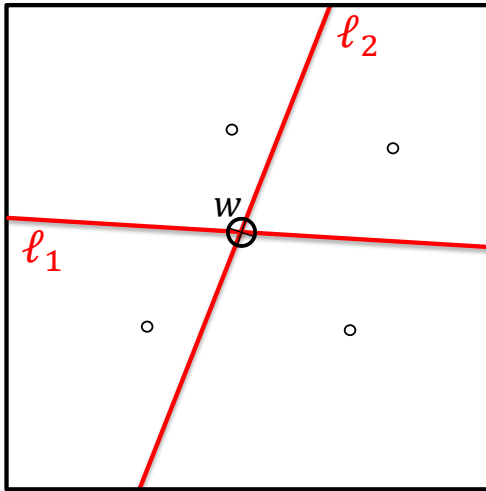


■ **Figure 2** An illustration of the line  $\ell'$  and a white-central point on it.

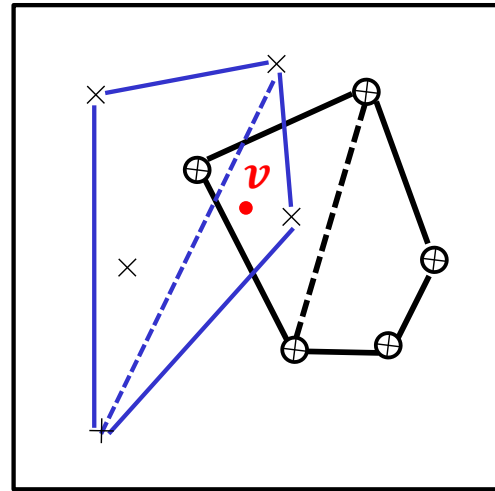
white-central points and  $B_\ell$  contains only black-central points (see Figure 1). The sum of the black area in  $W_\ell$  and the white area in  $B_\ell$  is at least  $\epsilon$  since the figure is  $\epsilon$ -far from being a half-plane. W.l.o.g. assume that the black area in  $W_\ell$  is at least  $\epsilon/2$ . Consider the line  $\ell'$  that is parallel to  $\ell$  and such that the black area in one of the two half-planes defined by  $\ell'$  is equal to  $\epsilon/2$ . (See Figure 2. Note that the black area in the other half-plane is at least  $\epsilon/2$ .) Clearly,  $\ell'$  lies in  $W_\ell$ . Next, we show that there is a black-central point on  $\ell'$ , i.e., in  $W_\ell$ , thus arriving at a contradiction. Consider the two sets of black points, on either side of  $\ell'$ . We have argued that each of them has area at least  $\epsilon/2$ . By the Ham Sandwich Theorem, applied to the two sets, there is a line  $\ell''$  that bisects the two sets simultaneously, forming four sets black points of area at least  $\epsilon/4$  each. The intersection point of  $\ell'$  and  $\ell''$  is black-central and lies in  $W_\ell$ . This is a contradiction, since  $\ell$  is a line that separates white-central and black-central points. ◀

Consider a white-central point  $w$  which is the intersection of two lines  $\ell_1$  and  $\ell_2$ , as shown in Figure 3. If four white pixels from four different quadrants determined by  $\ell_1$  and  $\ell_2$  are sampled by Algorithm 1, we say that the tester *captures*  $w$ . (The tester captures a black-central point analogously.) By Lemma 2.3, the convex hull of all white-central points and the convex hull of all black-central points intersect. Thus, there is a point  $v$  that lies in both convex hulls (see Figure 4). Moreover, there exists a set  $P_W$  of at most three white-central points such that point  $v$  lies in the convex hull of the points in  $P_W$ . Analogously, there exists a set  $P_B$  of at most three black-central points such that point  $v$  lies in the convex hull of the points in  $P_B$ . If all points in  $P_W \cup P_B$  are captured then  $v$  simultaneously lies in the convex hull of black samples and in the convex hull of white samples, i.e., the convex hull of black samples and the convex hull of white samples intersect, and the tester will reject the figure. The probability that the tester fails to capture a specific point in  $P_W \cup P_B$  is, by the union bound, at most  $4 \cdot (1 - \epsilon/4)^{18/\epsilon} \leq 4 \cdot e^{-18/4}$ . The probability that the tester fails to capture at least one point in  $P_W \cup P_B$  is at most  $6 \cdot 4 \cdot e^{-18/4} < 0.3$ . Therefore, the failure probability of the tester is at most 0.3.

**Sample and time complexity.** Algorithm 1 samples  $s = O(\epsilon^{-1})$  points.



■ **Figure 3** An illustration of a captured white central point.



■ **Figure 4** An illustration of the point  $v$  in the intersection of two convex hulls.

Next, we analyze its running time. Conduct the following mental experiment: Suppose we sample points from the rectangle  $R$  (defined in Algorithm 1) uniformly and independently at random until we collect  $s$  points from  $U$ ; then we bucket sort sampled points by their  $x$ -coordinate into  $s$  bins. Let  $q$  be the number of points we sample. Then  $\mathbb{E}[q] \leq 2s$ . Since the  $x$ -coordinates of the sampled  $q$  points are distributed uniformly in the interval corresponding to the length of the rectangle  $R$ , they can be sorted in expected time  $O(q)$  by subdividing this interval into  $s$  subintervals of equal length, and using them as buckets in the bucket sort. Thus, the expected running time of this algorithm is  $O(s)$ .

Observe that Algorithm 1 has the same distribution on the  $s$  points sampled from  $U$  as the algorithm in the mental experiment. It sorts two (disjoint) subsets of the points sampled in the mental experiment. Thus, the expected running time of Step 3 of Algorithm 1 is  $O(s)$ . Andrew's monotone chain algorithm finds the convex hull of a set of  $s$  sorted points in time  $O(s)$ . Merging also takes  $O(s)$  time. Overall, Algorithm 1 runs in expected time  $O(s) = O(\epsilon^{-1})$ . By standard arguments, we get a uniform algorithm with the worst case running time  $O(\epsilon^{-1})$  and with a slightly larger error probability  $\delta$  than in Algorithm 1, specifically, with  $\delta = 1/3$ . ◀

### 3 The Adaptive Tester for Convexity

► **Theorem 3.1.** *Given  $\epsilon \in (0, 1/2)$ , convexity of figure  $(U, C)$  can be  $\epsilon$ -tested (adaptively) with 1-sided error in time  $O(\epsilon^{-1})$ .*

**Proof.** In [4], it was shown that testing convexity of figures  $(U, C)$  can be reduced to the special case when the universe  $U$  is an axis-aligned rectangle of unit area. Therefore, we can assume w.l.o.g. that  $U$  is an axis-aligned rectangle of unit area.

Our  $\epsilon$ -tester for convexity (Algorithm 2) samples points uniformly at random and constructs a rectangle  $R$  that with high probability contains nearly the entire black area and whose sides include sampled black points. (See Figure 5.) Then it adaptively queries points of  $R$  in order to partition it into regions  $B$ ,  $W$  and  $F$ . (See Figure 6.) The “fence” region  $F$  has a small area. If the image is convex,  $B$  is entirely black and  $W$  is entirely white.

---

**Algorithm 2:**  $\epsilon$ -tester for convexity.
 

---

**input** : parameter  $\epsilon \in (0, 1/2)$ ; access to a figure  $(U, C)$ .

- 1 Query  $\frac{64}{\epsilon}$  points uniformly at random. If all sampled points are white, **accept**.
- 2 Let  $R$  be the minimum axis-parallel rectangle that contains all sampled black points. Let  $p_0$  (respectively,  $p_1, p_2, p_3$ ) be a sampled black point on the top (respectively, left, bottom, right) side of  $R$ .
- 3 **for**  $i \leftarrow 0$  **to** 3 **do**
- 4     Let  $(x, y) \leftarrow p_i$  and  $P_i \leftarrow \emptyset$ . // Investigate the upper right corner of  $R$ .
- 5     **while**  $(x, y)$  is in  $R$  **do**
- 6         **if**  $(x, y)$  is black or below the line through  $p_i$  and  $p_{(i+3) \bmod 4}$  **then**  
             $x \leftarrow x + \epsilon/12$ . // Move right.
- 7         **else**  
             $P_i \leftarrow P_i \cup \{(x, y)\}$ ;  $y \leftarrow y - \epsilon/12$ . // Move down.
- 8     Let  $W_i \leftarrow \{(u, v) \text{ inside } R \mid \exists (x, y) \in P_i \text{ such that } u \geq x, v \geq y \text{ with respect to the rotated coordinates}\}$ . Rotate  $R$  clockwise by 90 degrees.  
 // We rotate  $R$  to reuse lines 4-8 of the pseudocode for investigating all four corners.
- 9 Let  $B$  be the convex hull of all black points discovered after Step 3, and  $W \leftarrow \cup_{i=0}^3 W_i$ .
- 10 Query  $\frac{8}{\epsilon}$  points in  $B \cup W$  uniformly and independently. If a white point in  $B$  or a black point in  $W$  is detected, **reject**; otherwise, **accept**.

---

The algorithm queries a small number of random points in  $B \cup W$  and rejects if it finds a misclassified point (i.e., a white point in  $B$  or a black point in  $W$ ); otherwise, it accepts.

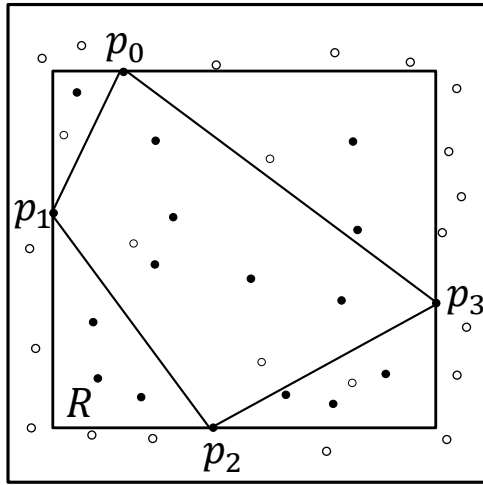
Since the black area outside  $R$  and the area of  $F$  are small, if the figure is  $\epsilon$ -far from convexity then there will be enough misclassified points in  $B \cup W$ , and the algorithm will detect at least one of them with high probability.

We prove that Algorithm 2 satisfies Theorem 3.1. First, we argue that Algorithm 2 always accepts if its input is a convex figure. If  $(U, C)$  has no black points (i.e.,  $C = \emptyset$ ), Step 1 always accepts. Otherwise, all points in  $B$  are black, by convexity of  $(U, C)$ . We will show that all points in  $W$  are white. For the sake of contradiction, suppose there is a black point  $b = (u, v)$  in  $W_0$ . By definition of  $W_0$ , there is a white point  $w = (x, y)$  in  $P_0$  such that  $u \geq x$  and  $v \geq y$ . Thus, white point  $w$  is inside the triangle  $p_0bp_3$ , formed by three black points, contradicting convexity of  $(U, C)$ . Thus, there are no black points in  $W_0$ . Analogously, there are no black points in  $W_1, W_2$  and  $W_3$ . Since there are no white points in  $B$  and no black points in  $W = \cup_{i=0}^3 W_i$ , Step 10 of Algorithm 2 always accepts  $(U, C)$ .

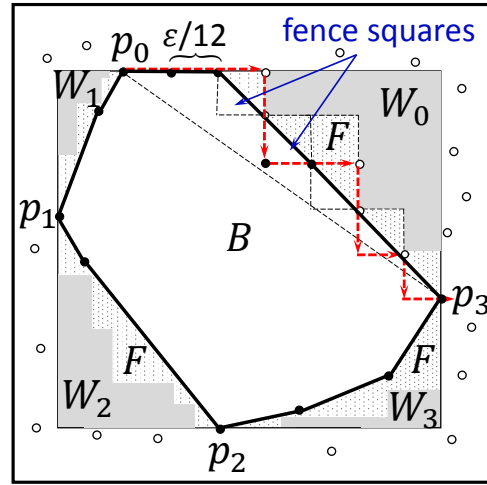
Now assume that  $(U, C)$  is  $\epsilon$ -far from convexity.

► **Lemma 3.2.** *The probability that the black area outside  $R$  is greater than  $\frac{\epsilon}{4}$  after Step 2 of Algorithm 2 is at most  $1/9$ .*

**Proof.** Let  $L$  be a horizontal line with the largest  $y$ -coordinate such that the black area of the figure above  $L$  is at least  $\frac{\epsilon}{16}$ . The probability that no black points above  $L$  are sampled in Step 1 of Algorithm 2 (and, consequently  $R$  lies below  $L$ ) is at most  $(1 - \frac{\epsilon}{16})^{64/\epsilon} \leq e^{-4} < 1/36$ . Thus, with probability at most  $1/36$ , the black area in the half-plane above  $R$  is greater than  $\frac{\epsilon}{16}$ . The same bound holds for the half-planes to the left, to the right and below  $R$ . By a union bound, the probability that the black area outside  $R$  is greater than  $\frac{\epsilon}{4}$  is at most  $(1/36) \cdot 4 = 1/9$ . ◀



■ **Figure 5** An illustration to Step 2 of Algorithm 2.



■ **Figure 6** An illustration to Steps 3–9 of Algorithm 2.

► **Lemma 3.3.** *Let  $F = R - (B \cup W)$ . Then the area of  $F$  is at most  $\frac{\epsilon}{2}$ .*

**Proof.** Let  $m = \epsilon/12$  and  $(x_i, y_i)$  be  $p_i$  (as defined in Step 2 of Algorithm 2) for  $i \in \{0, 1, 2, 3\}$ . Call every region that consists of points  $(x, y) + [0, m]^2$  a *square*, where  $\frac{x-x_i}{m}, \frac{y-y_i}{m} \in \mathbb{N}$ . Call squares that contain points from  $F$  *fence squares*. Let  $r = (x_3, y_0)$  and let  $T = \Delta p_0 p_3 r$ . We will find an upper bound on the number of fence squares inside  $T$ . Each point that Algorithm 2 queries in Step 5 results in at most one (new) fence square in  $T$ . The algorithm queries at most  $\frac{x_3-x_0+y_0-y_3}{\epsilon/12} + 2$  points in the triangle (thus, it discovers at most that many fence squares), since, in every iteration, it either increases the  $x$ -coordinate or decreases the  $y$ -coordinate of the queried point. Therefore, there are at most  $\frac{x_3-x_0+y_0-y_3}{\epsilon/12} + 2$  fence squares in this triangle. Similarly, we can find an upper bound on the number of discovered fence squares in the remaining triangles. Since the perimeter of  $R$  is at most 4, the sum of the upper bounds is at most  $\frac{4}{\epsilon/12} + 8 = \frac{48}{\epsilon} + 8 \leq \frac{56}{\epsilon}$ . The area of a single fence square is  $(\frac{\epsilon}{12})^2 = \frac{\epsilon^2}{144}$  and thus the total area of  $F$  is at most  $\frac{\epsilon^2}{144} \cdot \frac{56}{\epsilon} \leq \frac{\epsilon}{2}$ . ◀

We call a point *misclassified* if it is black and is in  $W$  or if it is white and in  $B$ . (The area that the set of misclassified points cover is called a *misclassified area*.) If we make all area in  $B$  black and all area outside of  $B$  white, we obtain a convex figure. Thus, by Lemma 3.3, the misclassified area in  $B \cup W$  is at least  $\frac{\epsilon}{4}$  if the black area outside of  $R$  is at most  $\frac{\epsilon}{4}$ . If the latter is the case, the probability that the algorithm will not detect a misclassified point is at most  $(1 - \frac{\epsilon}{4})^{\frac{8}{\epsilon}} < e^{-2} < 2/9$ . By Lemma 3.2, the probability that the misclassified area in  $B \cup W$  is less than  $\frac{\epsilon}{4}$  is at most  $1/9$ . Therefore, the probability that Algorithm 2 accepts is at most  $2/9 + 1/9 = 1/3$ , as desired.

**Query complexity.** The algorithm queries points in Steps 1, 6 and 10. In Steps 1 and 10, the algorithm makes  $O(\frac{1}{\epsilon})$  queries. In Step 6, over all iterations, the algorithm also queries  $O(\frac{1}{\epsilon})$  points. Thus, the overall query complexity of the algorithm is  $O(\frac{1}{\epsilon})$ .

**Running time.** The running time of the algorithm in Steps 1 through 9 is  $O(\frac{1}{\epsilon})$ . Starting from the uppermost horizontal side of  $R$  consider a partition of  $R$  into horizontal strips with

width  $\epsilon/12$ . Note that there are  $O(\frac{1}{\epsilon})$  such strips. Moreover, for each strip, there are at most 2 vertical lines that define the boundary of  $W$  (not the lines that define the sides of  $R$ ) and at most 2 lines that define the boundary of  $B$  (see Figure 6). Thus, for each horizontal strip in the partition of  $R$ , we can store at most 4 lines that define the boundary of  $B$  or the boundary of  $W$ . Given a sampled point  $p$  from Step 10, in  $O(1)$  time we identify the horizontal strip that point  $p$  belongs to. For each line  $\ell$  stored for this strip, in time  $O(1)$  we identify the half-plane (defined by  $\ell$ ) that contains point  $p$ . Thus, in time  $O(1)$  we determine whether  $p$  is in  $B \cup W$ . Since we sample  $O(\frac{1}{\epsilon})$  points in Step 10 its running time is  $O(\frac{1}{\epsilon})$ . Therefore, the running time of the algorithm is  $O(\frac{1}{\epsilon})$ , as claimed.  $\blacktriangleleft$

## 4 The Lower Bound for Nonadaptive Convexity Testers

### 4.1 Preliminaries on Poissonization

The proof of our lower bound uses a technique called *Poissonization* [26], in which one modifies a probabilistic experiment to replace a fixed quantity (e.g., the number of samples) with a variable one that follows a Poisson distribution. This breaks up dependencies between different events and makes the analysis tractable. The Poisson distribution with parameter  $\lambda \geq 0$ , denoted  $\text{Po}(\lambda)$ , takes each value  $x \in \mathbb{N}$  with probability  $\frac{e^{-\lambda} \lambda^x}{x!}$ . The expectation and variance of a random variable distributed according to  $\text{Po}(\lambda)$  are both  $\lambda$ .

► **Definition 4.1.** A *Poisson- $s$  tester* is a uniform tester that takes a random number of samples distributed as  $\text{Po}(s)$ .

► **Lemma 4.2** (Poissonization Lemma [22, Lemma 5.3] and [4]).

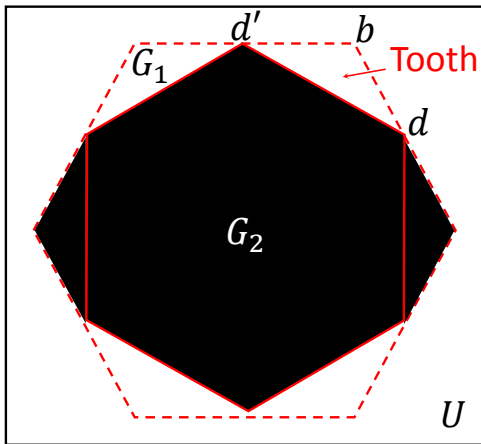
- (a) *Poisson algorithms can simulate uniform algorithms. Specifically, for every uniform tester  $\mathcal{A}$  for property  $\mathcal{P}$  that uses at most  $s$  samples and has error probability  $\delta$ , there is a Poisson- $2s$  tester  $\mathcal{A}'$  for  $\mathcal{P}$  with error probability at most  $\delta + 4/s$ . Moreover,*
- (b) *Let  $\Omega$  be a sample space from which a Poisson- $s$  algorithm makes uniform draws. Suppose we partition  $\Omega$  into sets  $\Omega_1, \dots, \Omega_k$  (e.g., these sets can correspond to disjoint areas of the figure from which points are sampled), where each outcome is in set  $\Omega_i$  with probability  $p_i$  for  $i \in [k]$ . Let  $X_i$  be the total number of samples in  $\Omega_i$  seen by the algorithm. Then  $X_i$  is distributed as  $\text{Po}(p_i \cdot s)$ . Moreover, random variables  $X_i$  are mutually independent for all  $i \in [k]$ .*

### 4.2 The Lower Bound

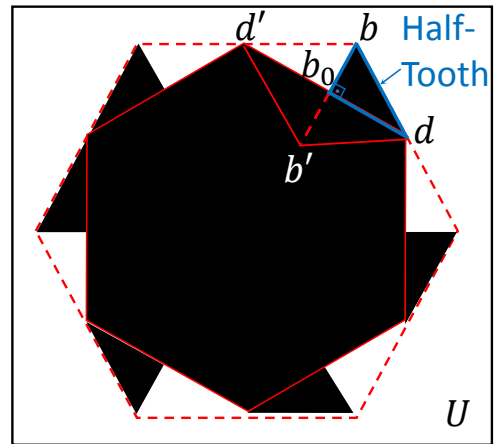
► **Theorem 4.3.** *Every 2-sided error uniform  $\epsilon$ -tester for convexity needs  $\Omega(\epsilon^{-5/4})$  samples.*

**Proof.** By the Poissonization Lemma (Lemma 4.2), it is enough to prove the lower bound for Poisson algorithms. For sufficiently small  $\epsilon$ , we define distributions  $\mathcal{P}$  and  $\mathcal{N}$  on figures, where  $\mathcal{P}$  is supported only on convex figures whereas  $\mathcal{N}$  is supported only on figures which are  $\epsilon$ -far from convexity. We show that every uniform Poisson- $s$  tester, where  $s = o(\epsilon^{-5/4})$ , fails to distinguish  $\mathcal{P}$  from  $\mathcal{N}$  with sufficient probability.

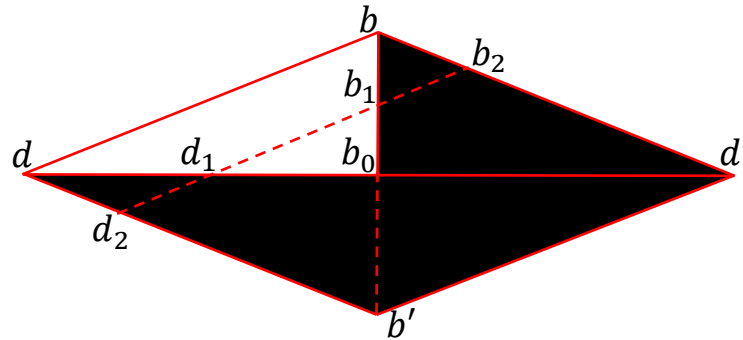
Let  $k = \lceil \frac{1}{2} \cdot \epsilon^{-1/2} \rceil$  and the universe  $U = [0, 1]^2$ . Consider two regular convex  $k$ -gons  $G_1$  and  $G_2$ , centered at  $(1/2, 1/2)$ , such that  $G_1$  has side length  $\sin(\frac{\pi}{k})$  and the vertices of  $G_2$  are the midpoints of the sides of  $G_1$  (see Figure 7). Call triangular regions inside  $G_1$  but outside  $G_2$  *teeth* (one such triangular region is a *tooth*). Let  $T$  be a tooth and  $b$  be its vertex which is also a vertex of  $G_1$ . Let the other two vertices of  $T$  be  $d$  and  $d'$  and let  $b_0$  be a point on  $dd'$  such that  $bb_0$  is the height of  $T$  from  $b$  to its base  $dd'$ . Call  $\triangle bb_0d$  and  $\triangle bb_0d'$  *half-teeth* (see Figure 8). Distributions  $\mathcal{P}$  and  $\mathcal{N}$  are defined next.



■ **Figure 7** A figure from  $\mathcal{P}$  for  $k = 6$ .



■ **Figure 8** A figure from  $\mathcal{N}$  for  $k = 6$ .



■ **Figure 9** An illustration of a block.

1. For all figures from both distributions, points outside  $G_1$  are white and points in  $G_2$  are black.
2. For a figure in  $\mathcal{P}$ , every tooth is independently colored white or black, each with probability  $1/2$ , as shown in Figure 7.
3. For a figure in  $\mathcal{N}$ , every tooth is independently colored as follows: one half-tooth is colored black or white, each with probability  $1/2$ , and the other half-tooth gets the opposite color, as shown in Figure 8.

Note that every figure in the support of  $\mathcal{P}$  is convex.

► **Lemma 4.4.** For all  $\epsilon \leq 3 \cdot 10^{-3}$ , every figure in the support of  $\mathcal{N}$  is  $\epsilon$ -far from convexity.

**Proof.** Let  $A_\Delta$  denote the area of a tooth. Consider a figure  $(U, C)$  in the support of  $\mathcal{N}$ . Let  $\triangle bdd'$  be a tooth of  $(U, C)$ . Consider point  $b'$  that is symmetric to  $b$  with respect to the line  $dd'$ , as shown in Figure 8. Call the quadrilateral  $bb'dd'$  a *block*. Observe that there are  $k$  disjoint blocks. Let  $(U, C')$  be a convex figure that is closest to  $(U, C)$ .

► **Claim 4.5.** In every block of  $C$ , area at least  $\frac{A_\Delta}{16}$  must be modified to obtain  $C'$  from  $C$ .

**Proof.** For a region  $R$ , let  $A(R)$  denote the area of  $R$ .

Consider the block  $bb'dd'$  illustrated in Figure 9. Let  $b_1$  and  $d_1$  be the midpoints of  $bb'$  and  $dd'$ , respectively. Let the line  $b_1d_1$  intersect  $bb'$  and  $dd'$  at  $b_2$  and  $d_2$ , respectively.

Consider the white triangle  $\triangle b_1 b_0 d_1$  and the three black triangles  $\triangle d d_1 d_2$ ,  $\triangle b b_1 b_2$ , and  $\triangle b_0 b' d'$ . If there is a point in each of these four triangles that has not changed color, then we have a white point in the convex hull of three black points, i.e., the figure is not convex. Therefore, in at least one of these four triangles, all points must change color in order to make the figure convex. Since the areas of the triangles are

$$A(\triangle b_0 b' d') = \frac{A_\Delta}{2}, A(\triangle b_1 b_0 d_1) = \frac{A_\Delta}{8}, A(\triangle d d_1 d_2) = A(\triangle b b_1 b_2) = \frac{A_\Delta}{16},$$

the claim holds. ◀

► **Claim 4.6.**  $5.6 \cdot \frac{1}{k^3} < A_\Delta \leq 8 \cdot \frac{1}{k^3}$ .

**Proof.** By simple geometric reasoning,

$$A_\Delta = \frac{1}{2} \cdot \frac{1}{4} \cdot \sin^2\left(\frac{\pi}{k}\right) \cdot \sin\left(\frac{2\pi}{k}\right).$$

Since  $0.9x \leq \sin x \leq x$  for  $x \in [0, 0.78]$ , we obtain that, for sufficiently large  $k$  (i.e., for  $\epsilon \leq 3 \cdot 10^{-3}$ ),

$$A_\Delta \geq \frac{1}{8} \cdot \left(0.9 \cdot \frac{\pi}{k}\right)^2 \cdot \left(0.9 \cdot \frac{2\pi}{k}\right) > 5.6 \cdot \frac{1}{k^3};$$

$$A_\Delta \leq \left(\frac{1}{8}\right) \cdot \left(\frac{\pi}{k}\right)^2 \cdot \left(\frac{2\pi}{k}\right) \leq \frac{8}{k^3}. \quad \blacktriangleleft$$

There are  $k$  blocks and by Claim 4.5 at least

$$k \cdot \frac{A_\Delta}{16} > k \cdot \frac{5.6}{16} \cdot \frac{1}{k^3} = \frac{7}{20} \cdot \frac{1}{k^2} \geq \epsilon$$

area needs to be modified to make  $C$  convex. (Recall that  $k = \lceil \frac{1}{2} \cdot \epsilon^{-1/2} \rceil$ .) ◀

Consider a Poisson- $s$  algorithm  $\mathcal{A}$  with  $s = c_0 \cdot \epsilon^{-5/4}$ . We will show that when  $c_0$  is sufficiently small then  $\mathcal{A}$  fails on  $\mathcal{P}$  or  $\mathcal{N}$  with probability greater than  $1/3$ .

► **Definition 4.7.** A pair of points  $(p_1, p_2)$  is called a *red-flag pair* if  $p_1$  and  $p_2$  belong to different half-teeth of the same tooth.

Let  $BAD$  denote the event that no red-flag pair is sampled by the algorithm  $\mathcal{A}$ .

► **Claim 4.8.** If  $c_0$  is sufficiently small,  $\Pr[\overline{BAD}] < 1/10$ .

**Proof.** Let  $L_T$  and  $R_T$  be the random variables that count the number of points sampled by the tester in the left half-tooth and in the right half-tooth of a tooth  $T$ , respectively. Let  $X_T$  and  $X$  be the random variables that count the number of sampled red-flag pairs in a tooth  $T$  and in all teeth, respectively. By the Poissonization Lemma (Lemma 4.2),  $L_T$  and  $R_T$  are independent Poisson random variables with expectation  $(A_\Delta/2) \cdot s$ . Note that  $X_T = L_T \cdot R_T$  and, therefore,

$$\mathbb{E}[X_T] = \mathbb{E}[L_T] \cdot \mathbb{E}[R_T] = (A_\Delta/2)^2 \cdot s^2 \leq \frac{16s^2}{k^6},$$

by Claim 4.6. Since all teeth are disjoint, then for sufficiently small  $c_0$ ,

$$\mathbb{E}[X] = k \cdot \mathbb{E}[X_T] \leq k \cdot \frac{16s^2}{k^6} \leq 512 \cdot c_0^2 < 1/10.$$

By Markov's inequality,  $\Pr[\overline{BAD}] = \Pr[X \geq 1] \leq \mathbb{E}[X] < 1/10$ . ◀



Conditioned on  $BAD$ , the distribution on the answers to the queries made by  $\mathcal{A}$  is the same whether the input is sampled from  $\mathcal{P}$  or  $\mathcal{N}$ . Therefore,

$$\Pr_{x \sim \mathcal{P}}[\mathcal{A} \text{ accepts } x \mid BAD] = \Pr_{x \sim \mathcal{N}}[\mathcal{A} \text{ accepts } x \mid BAD] = 1 - \Pr_{x \sim \mathcal{N}}[\mathcal{A} \text{ rejects } x \mid BAD].$$

Consequently,

$$\min\left(\Pr_{x \sim \mathcal{P}}[\mathcal{A} \text{ accepts } x \mid BAD], \Pr_{x \sim \mathcal{N}}[\mathcal{A} \text{ rejects } x \mid BAD]\right) \leq 1/2.$$

Assume w.l.o.g. that  $\Pr_{x \sim \mathcal{P}}[\mathcal{A} \text{ accepts } x \mid BAD] \leq 1/2$ . Then,

$$\begin{aligned} & \Pr_{x \sim \mathcal{P}}[\mathcal{A} \text{ accepts } x] \\ &= \Pr_{x \sim \mathcal{P}}[\mathcal{A} \text{ accepts } x \mid \overline{BAD}] \cdot \Pr[\overline{BAD}] + \Pr_{x \sim \mathcal{P}}[\mathcal{A} \text{ accepts } x \mid BAD] \cdot \Pr[BAD] \\ &< 1 \cdot \frac{1}{10} + \Pr_{x \sim \mathcal{P}}[\mathcal{A} \text{ accepts } x \mid BAD] \cdot 1 \\ &\leq \frac{1}{10} + \frac{1}{2} < \frac{2}{3}. \end{aligned}$$

Thus, a uniform algorithm needs  $\Omega(\epsilon^{-5/4})$  samples to test convexity with error probability at most  $1/3$ .  $\blacktriangleleft$

## 5 Conclusion and Open Problems

We showed that uniform testers are as powerful as adaptive testers in the case of the half-plane property. Specifically, our uniform half-plane tester has 1-sided error and optimal running time. For convexity, the best previously known tester was uniform. However, we designed an adaptive tester with better (optimal) query complexity and showed that every uniform tester must have a significantly larger query complexity than our adaptive tester.

One remaining open problem is to resolve the sample complexity of an optimal (2-sided error) uniform tester for convexity. Our lower bound on this quantity is  $\Omega(\epsilon^{-5/4})$ , while the best upper bound is  $O(\epsilon^{-4/3})$  [4]. Another direction for research is to investigate the power of uniform samples in the context of tolerant property testing. Tolerant testing of 2-dimensional figures was investigated in [5]. The tolerant testers for half-plane and convexity in that work are uniform and have nearly optimal query complexity (as compared to any, even adaptive testers). However, it is open whether uniform samples are sufficient for achieving the optimal running time for tolerantly testing these properties. It is interesting to investigate the power of other restricted classes of testers, such as nonadaptive testers, in the context of testing of properties of geometric figures. Finally, this work only looks at 2-dimensional figures. Generalizing this study to higher dimensions is an intriguing open question.

---

### References

- 1 Noga Alon, Eldar Fischer, Ilan Newman, and Asaf Shapira. A combinatorial characterization of the testable graph properties: It's all about regularity. *SIAM J. Comput.*, 39(1):143–167, 2009.
- 2 A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Inf. Process. Lett.*, 9(5):216–219, 1979. doi:10.1016/0020-0190(79)90072-3.
- 3 Tugkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. Testing closeness of discrete distributions. *J. ACM*, 60(1):4, 2013.

- 4 Piotr Berman, Meiram Murzabulatov, and Sofya Raskhodnikova. Testing convexity of figures under the uniform distribution. In *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA*, volume 51 of *LIPIcs*, pages 17:1–17:15, 2016. doi:10.4230/LIPIcs.SoCG.2016.17.
- 5 Piotr Berman, Meiram Murzabulatov, and Sofya Raskhodnikova. Tolerant Testers of Image Properties. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 90:1–90:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.90.
- 6 Piotr Berman, Sofya Raskhodnikova, and Grigory Yaroslavtsev.  $L_p$ -testing. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 164–173, 2014. doi:10.1145/2591796.2591887.
- 7 Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.
- 8 Artur Czumaj and Christian Sohler. Property testing with geometric queries. In *Algorithms – ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, pages 266–277, 2001.
- 9 Artur Czumaj, Christian Sohler, and Martin Ziegler. Property testing in computational geometry. In *Algorithms – ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, pages 155–166, 2000.
- 10 Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. Improved testing algorithms for monotonicity. In *Randomization, Approximation, and Combinatorial Algorithms and Techniques, Third International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM'99, Berkeley, CA, USA*, pages 97–108, 1999.
- 11 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1987.
- 12 O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.
- 13 Oded Goldreich, Shafi Goldwasser, Eric Lehman, Dana Ron, and Alex Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.
- 14 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.
- 15 Oded Goldreich and Dana Ron. On sample-based testers. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 337–345, 2015.
- 16 Igor Kleiner, Daniel Keren, Ilan Newman, and Oren Ben-Zwi. Applying property testing to an image partitioning problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(2):256–265, 2011. doi:10.1109/TPAMI.2010.165.
- 17 Simon Korman, Daniel Reichman, and Gilad Tsur. Tight approximation of image matching. *CoRR*, abs/1111.1713, 2011.
- 18 Simon Korman, Daniel Reichman, Gilad Tsur, and Shai Avidan. Fast-match: Fast affine template matching. In *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*, pages 2331–2338, 2013. doi:10.1109/CVPR.2013.302.
- 19 Nimrod Megiddo. Partitioning with two lines in the plane. *J. Algorithms*, 6(3):430–433, 1985.
- 20 Luis Rademacher and Santosh Vempala. Testing geometric convexity. In *FSTTCS*, pages 469–480, 2004.
- 21 Sofya Raskhodnikova. Approximate testing of visual properties. In *RANDOM-APPROX*, pages 370–381, 2003. doi:10.1007/978-3-540-45198-3\_31.

- 22 Sofya Raskhodnikova, Dana Ron, Amir Shpilka, and Adam Smith. Strong lower bounds for approximating distribution support size and the distinct elements problem. *SIAM J. Comput.*, 39(3):813–842, 2009.
- 23 Dana Ron and Gilad Tsur. Testing properties of sparse images. *ACM Trans. Algorithms*, 10(4):17:1–17:52, 2014.
- 24 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.
- 25 Bernd Schmeltz. Learning convex sets under uniform distribution. In *Data Structures and Efficient Algorithms, Final Report on the DFG Special Joint Initiative*, pages 204–213, 1992.
- 26 Wojciech Szpankowski. *Average Case Analysis of Algorithms on Sequences*. John Wiley & Sons, Inc., New York, 2001.
- 27 John W. Tukey. Mathematics and the picturing of data. In *Proceedings of the international congress of mathematicians*, volume 2, pages 523–531, 1975.
- 28 Paul Valiant. Testing symmetric properties of distributions. *SIAM J. Comput.*, 40(6):1927–1968, 2011. doi:10.1137/080734066.

# Local Testing for Membership in Lattices

Karthekeyan Chandrasekaran<sup>1</sup>, Mahdi Cheraghchi<sup>2</sup>,  
Venkata Gandikota<sup>3</sup>, and Elena Grigorescu<sup>4</sup>

- 1 University of Illinois, Urbana-Champaign, IL, USA  
karthe@illinois.edu
- 2 Imperial College London, UK  
cheraghchi@berkeley.edu
- 3 Purdue University, West Lafayette, IN, USA  
vgandiko@purdue.edu
- 4 Purdue University, West Lafayette, IN, USA  
elena-g@purdue.edu

---

## Abstract

Testing membership in lattices is of practical relevance, with applications to integer programming, error detection in lattice-based communication and cryptography. In this work, we initiate a systematic study of *local testing* for membership in lattices, complementing and building upon the extensive body of work on locally testable codes. In particular, we formally define the notion of local tests for lattices and present the following:

1. We show that in order to achieve low query complexity, it is sufficient to design one-sided non-adaptive *canonical* tests. This result is akin to, and based on an analogous result for error-correcting codes due to Ben-Sasson *et al.* (SIAM J. Computing, 35(1):1–21).
2. We demonstrate upper and lower bounds on the query complexity of local testing for membership in *code formula* lattices. We instantiate our results for code formula lattices constructed from Reed-Muller codes to obtain nearly-matching upper and lower bounds on the query complexity of testing such lattices.
3. We contrast lattice testing from code testing by showing lower bounds on the query complexity of testing low-dimensional lattices. This illustrates large lower bounds on the query complexity of testing membership in *knapsack lattices*. On the other hand, we show that knapsack lattices with bounded coefficients have low-query testers if the inputs are promised to lie in the span of the lattice.

**1998 ACM Subject Classification** F.2 Analysis of algorithms and problem complexity

**Keywords and phrases** Lattices, Property Testing, Locally Testable Codes, Complexity Theory

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.46

## 1 Introduction

Local testing for properties of combinatorial and algebraic objects have widespread applications and have been intensely investigated in the past few decades. The main underlying goal in Local Property Testing is to distinguish objects that satisfy a given property from objects that are far from satisfying the property, using a small number of observations of the input object. Starting with the seminal works of [7, 13, 33], significant focus in the area has been devoted to locally testable error-correcting codes, called Locally Testable Codes (LTCs) [15]. LTCs are the key ingredients in several fundamental results in complexity theory, most notably in the PCP theorem [2, 3].



© Karthekeyan Chandrasekaran, Mahdi Cheraghchi, Venkata Gandikota, and Elena Grigorescu; licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 46; pp. 46:1–46:14



Leibniz International Proceedings in Informatics

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

In this work we initiate the study of local testability for membership in *point lattices*, a class of infinite algebraic objects that form discrete subgroups of  $\mathbb{R}^n$ . Lattices are well-studied in mathematics, physics and computer science due to their rich algebraic structure [9]. Algorithms for various lattice problems have directly influenced the ability to solve integer programs [10, 23, 17]. Recently, lattices have found applications in modern cryptography due to attractive properties that enable efficient computations and security guarantees [28, 26, 31, 32]. Lattices are also used in practical communication settings to encode data in a redundant manner in order to protect it from channel noise during transmission [12].

A point lattice  $L \subset \mathbb{R}^n$  of *rank*  $k$  and *dimension*  $n$  is specified by a set of linearly independent vectors  $b_1, \dots, b_k \in \mathbb{R}^n$  known as a basis, for some  $k \leq n$ . If  $k = n$  the lattice is said to have full rank. The set  $L$  is defined to be the set of all vectors in  $\mathbb{R}^n$  that are integer linear combinations of the basis vectors, i.e.,  $L := \{\sum_{i=1}^k \alpha_i b_i \mid \alpha_i \in \mathbb{Z} \forall i \in [k]\}$ . Lattices are the analogues over  $\mathbb{Z}$  of linear error-correcting codes over a finite field  $\mathbb{F}$ , which are generated as  $\mathbb{F}$ -linear combinations of a linearly independent set of basis vectors  $b_1, \dots, b_k \in \mathbb{F}^n$ .

Given a basis for a lattice  $L$ , we are interested in testing if a given input  $t \in \mathbb{R}^n$  belongs to  $L$ , or is far from all points in  $L$  by querying a small number of coordinates of  $t$ . We emphasize that this setting does not limit the computational space or time in pre-processing the lattice as well as the queried coordinates. The main goal is to design a tester that queries only a small number of coordinates of the input.

## 1.1 Motivation

**Integer Programming.** Lattices are the fundamental structures underlying integer programming problems. An integer programming problem (IP) is specified by a constraint matrix  $A \in \mathbb{R}^{n \times m}$ , a vector  $b \in \mathbb{R}^n$ . The goal is to verify if there exists an integer solution to the system  $Ax = b, x \geq 0$ . Although IP is NP-complete [18], its instances are solved routinely in practice using cutting planes and branch-and-cut techniques [35]. The relaxed problem of verifying integer feasibility of the system  $Ax = b$  is equivalent to verifying whether  $b$  lies in the lattice generated by the columns of  $A$ . Thus, the relaxation problem is the membership testing problem in a lattice. It is solvable efficiently and is a natural pre-processing step to solving IPs. Furthermore, if the number of constraints  $n$  in the problem is very large, then it would be helpful to run a tester that reads only a partial set of coordinates of the input  $b$  to verify if  $b$  could lie in the lattice generated by the columns of  $A$  or is far from it. If the test rejects, then this saves on the computational effort to search for a non-negative solution.

**Cryptography.** In cryptographic applications, it is imperative to understand which lattices are difficult to test in order to ensure security of lattice-based cryptosystems. In some cryptanalytic attacks on lattice-based cryptosystems, one needs to distinguish target vectors that are close to lattice vectors from those that are far from all lattice vectors, a problem commonly known as the gap version of the Closest Vector Problem (GapCVP). An approach to address GapCVP is to use expensive distance estimation algorithms inspired by Aharonov and Regev [1] and Liu *et al.* [24]. Local testing of lattices is closely related to both distance estimation [30] and GapCVP, and hence progress in the proposed testing model could lead to new insights in cryptanalytic attacks.

**Complexity theory.** Lattices can be seen as coding theoretic objects naturally bringing features of error-correcting codes from the finite field domain to the real domain. As such, a study of local testing (and correction) procedures for lattices naturally extends the classical notions of Locally Testable Codes (LTCs) and Locally Decodable Codes (LDCs), which

are in turn of significance to computational complexity theory (for example in constructing probabilistically checkable proofs and hardness amplification, among numerous other applications). Characterizing local testability, explicitly initiated by Kaufman and Sudan [19], has been an intensely investigated direction in the study of LTCs. We believe that an analogous investigation of lattices is likely to bring new insights and new connections in property testing.

**Lattice-based communication.** Lattices are a major technical tool in communication systems as the analogue of error-correcting codes over reals, for applications such as wireless communication and transmission over analog lines. In lattice-coding, the message  $m$  is mapped to a point  $c$  in a chosen lattice  $L$ . The codeword  $c$  is transmitted over an analog channel. If the encoded message gets corrupted by the channel, then the channel output may not be a lattice point, thus enabling transmission error detection. In order to correct errors, computationally expensive decoding algorithms are employed. Instead, the receiver may perform a local test for membership in the lattice beforehand, allowing the costly decoding computation to run only when there is a reasonably high chance of correct decoding.

We now give an informal description of our testing model motivated by its application in lattice-coding. The transmission of each coordinate of a lattice-codeword over the analog channel consumes power that is proportional to the square of the transmitted value. Thus the power consumption for transmitting the lattice-codeword  $c \in L \subset \mathbb{R}^n$  is proportional to its squared  $\ell_2$  norm. The power consumption for transmitting a codeword over the channel is usually constrained by a power budget. The noise vector is also subject to a bound on its power. The power budget for transmission is typically formulated by considering the lattice-code  $C(L)$  defined by the set of lattice points  $c \in L$  that satisfy  $\sum_{i=1}^n c_i^2 \leq \sigma n$  for some constant power budget  $\sigma > 0$ . In order to ensure that the receiver can tolerate adversarial noise budget  $\delta$  per channel use, the shortest nonzero vector  $v \in L$  should be such that  $\sum_{i=1}^n v_i^2 \geq \delta n$ . Thus, the *relative distance* of the lattice-code  $C(L)$  is defined to be  $\sum_{i=1}^n v_i^2/n$ , where  $v \in L$  is a shortest nonzero lattice vector. The *rate* of a lattice-code  $C(L)$  is defined to be  $(1/n) \log |C(L)|$  (note that this quantity could be larger than 1). An *asymptotically good family of lattices*, in this work, is one that achieves rate and relative distance that are both lower bounded by a positive constant. Such families are ideal for use in noisy communication channels.

We define a notion of a tester that will be useful as a pre-processor for decoding, and is similar to the established notion of code testing: An  $\ell_2$ -tester of a lattice  $L$  for a given distance parameter  $\epsilon > 0$  is a probabilistic procedure that given an input  $t \in \mathbb{R}^n$ , queries at most  $q$  coordinates of  $t$ , accepts with probability at least  $2/3$  if  $t \in L$ , and rejects with probability at least  $2/3$  if  $\sum_{i=1}^n (t_i - w_i)^2 \geq \epsilon n$  for every  $w \in L$ .

For the purposes of lattice-coding, the central lattice testing problem is whether there exists an asymptotically good family of lattices that can be tested for membership with query complexity  $q = O(1)$ .

## 1.2 Testing model

In the above application, we focused on  $\ell_2$  distances. We now formalize the notion of testing lattices for  $\ell_p$  distances. We consider  $\ell_p$  distances since these are natural notions for real-valued inputs [5]. The  $\ell_p$  distance between  $x, y \in \mathbb{R}^n$  is defined as  $d_p(x, y) := \|x - y\|_p = (\sum_{i \in [n]} |x_i - y_i|^p)^{1/p}$ . The distance from  $v \in \mathbb{R}^n$  to  $L$  is  $d_p(v, L) := \min_{u \in L} d_p(v, u)$ . Denote the  $\ell_p$  norm of the real vector  $1^n$  by  $\|1^n\|_p$ . For a lattice  $L$ , we denote the subspace of the

## 46:4 Local Testing for Membership in Lattices

lattice by  $\text{span}(L)$ . We focus on integral lattices, which are sub-lattices of  $\mathbb{Z}^n$ , as these are the most commonly encountered lattices in applications<sup>1</sup>.

► **Definition 1** (Local test for lattices). An  $\ell_p$ -tester  $T(\epsilon, c, s, q)$  for a lattice  $L \subseteq \mathbb{Z}^n$  is a probabilistic algorithm that queries  $q$  coordinates of the input  $t \in \mathbb{R}^n$ , and

- (completeness) *accepts* with probability at least  $1 - c$  if  $t \in L$ ,
- (soundness) *rejects* with probability at least  $1 - s$  if  $d_p(t, L) \geq \epsilon \cdot \|1^n\|_p$  (we call such a vector  $t$  to be  $\epsilon$ -far from  $L$ ).

If  $T$  always accepts inputs  $t$  that are in the lattice  $L$  then it is called 1-sided, otherwise it is 2-sided. If the queries performed by  $T$  depend on the answers to the previous queries, then  $T$  is called adaptive, otherwise it is called non-adaptive.

A test  $T(\epsilon, 0, 0, q)$  is a test with perfect completeness and perfect soundness. 1-sided testers (i.e., testers with perfect completeness) are useful as a pre-processing step, as mentioned earlier. An asymptotically good family of lattices  $L(n)$  for  $\ell_p$  distances is one that has  $\ell_p$ -relative distance lower bounded by a constant (i.e.,  $\min_{v \in L(n)} \|v\|_p^n / n = \Omega(1)$ ) and has  $2^{\Omega(n)}$  lattice points in the origin-centered  $\ell_p$ -ball of radius  $n^{1/p}$ . Similar to the application in lattice-coding and locally testable codes, a main question in  $\ell_p$ -testing of lattices is the following:

► **Question 2.** *Is there an asymptotically good family of lattices that can be tested for membership with constant number of queries?*

Motivated by the applications in IP and cryptography, we identify another fundamental question in  $\ell_p$ -testing of lattices:

► **Question 3.** *What properties of a given lattice enable the design of  $\ell_p$ -testers with constant query complexity?*

**Tolerant Testing.** Many applications can tolerate a small amount of noise in the input. Parnas *et al.* [30] introduced the notion of tolerant testing to account for a small amount of noise in the input. Tolerant testing has been studied in the context of codes (e.g. [16, 20]), and in the context of properties of real-valued data in the  $\ell_p$  norm (e.g. [5]). We extend the tolerant testing model to lattices as follows.

► **Definition 4** (Tolerant local test for lattices). An  $\ell_p$ -tolerant-tester  $T(\epsilon_1, \epsilon_2, c, s, q)$  for a lattice  $L \subseteq \mathbb{Z}^n$  is a probabilistic algorithm that queries  $q$  coordinates of the input  $t \in \mathbb{R}^n$ , and

- (completeness) *accepts* with probability at least  $1 - c$  if  $d_p(t, L) \leq \epsilon_1 \cdot \|1^n\|_p$ ,
- (soundness) *rejects* with probability at least  $1 - s$  if  $d_p(t, L) \geq \epsilon_2 \cdot \|1^n\|_p$ .

Tolerant testing with parameter  $\epsilon_1 = 0$  corresponds to the notion of testing given in Definition 1. Tolerant testing and distance approximation are closely related notions. In fact, in the Hamming space, the ability to perform tolerant testing for *every* choice of  $\epsilon_1 < \epsilon_2$  can be exploited to approximate distances (using a binary search) [30].

---

<sup>1</sup> Arbitrary lattices can be approximated by rational lattices and rational lattices can be scaled to integral lattices.



**Analogy with code testers.** A common notion of testing for membership in *error-correcting codes* requires that inputs at *Hamming distance* at least  $\epsilon n$  from the code be rejected. (This notion is only relevant when the covering radius of the code is larger than  $\epsilon n$ .) We include the common definition here, and note that stronger versions of testing have also been considered in the literature [15, 16].

- **Definition 5** (Local test for codes). A tester  $T(\epsilon, c, s, q)$  for an error-correcting code  $C \subseteq \mathbb{F}^n$  is a probabilistic algorithm that makes  $q$  queries to the input  $t \in \mathbb{F}^n$ , and
- (completeness) *accepts* with probability at least  $1 - c$  if  $t \in C$ , and
  - (soundness) *rejects* with probability at least  $1 - s$  if  $d_H(t, C) \geq \epsilon \cdot n$ , where  $d_H(u, v) := |\{i \in [n] : u(i) \neq v(i)\}|$  denotes the Hamming distance between  $u$  and  $v$ , and  $d_H(t, C) := \min_{c \in C} d_H(t, c)$  (we call such a vector  $t$  to be  $\epsilon$ -far from  $C$ ).

### 1.3 Our contributions

We initiate the study of membership testing in point lattices from the perspective of sublinear algorithms aiming to lay the ground work for further advances towards resolving Question 2 and Question 3. Our contributions draw on connections between lattices and codes, and on well-known techniques in property testing.

#### 1.3.1 Upper and lower bounds for testing specific lattice families

Motivated by applications in lattice-based communication, we focus on an asymptotically good family of sets constructed from linear codes, via the so-called “code formula” [12]. We show upper and lower bounds on the query complexity of  $\ell_1$ -testers for code formulas, as a function of the query complexity of the constituent code testers.

**Code formula lattices.** For simplicity, in what follows we will slightly abuse notation and use binary code  $C \subseteq \{0, 1\}^n$  to denote both the code viewed over the field  $\mathbb{F}_2 = \{0, 1\}$  and the code embedded into  $\mathbb{R}^n$  via the trivial embedding  $0 \mapsto 0$  and  $1 \mapsto 1$ . All the arithmetic operations in the code formula refer to operations in  $\mathbb{R}^n$ . For two sets  $A$  and  $B$  of vectors we define  $A + B := \{a + b \mid a \in A, b \in B\}$ .

- **Definition 6** (Code Formula). Let  $C_0 \subseteq C_1 \subseteq \dots \subseteq C_{m-1} \subseteq C_m = \mathbb{F}_2^n$  be a family of nested binary linear codes. Then the code formula constructed from the family is defined as

$$C_0 + 2C_1 + \dots + 2^{m-1}C_{m-1} + 2^m\mathbb{Z}^n.$$

Here,  $m$  is the *height* of the code-formula.

If the family satisfies the *Schur product condition*, namely,  $c_1 * c_2 \in C_{i+1}$  for all codewords  $c_1, c_2 \in C_i$ , where the ‘\*’ operator is the coordinate-wise (Schur) product  $c_1 * c_2 = \langle (c_1)_i \cdot (c_2)_i \rangle_{i \in [n]}$ , then the code-formula forms a *lattice* (see [21]) and we denote it by  $L(\langle C_i \rangle_{i=0}^{m-1})$ .

**Significance of code formula lattices.** Code formula lattices with height one already have constant rate if the constituent code  $C_0$  has minimum Hamming distance  $\Omega(n)$ . Unfortunately, these lattices have tiny relative minimum distance (since  $2\mathbb{Z}^n$  has constant length vectors). However, code formulas of larger height achieve much better relative distance. In particular, it is easy to see that code formula lattices of height  $m \geq \log n$  in which each of the constituent codes  $C_i$  has minimum Hamming distance  $\Omega(n)$  give asymptotically good families of lattices [14, 9]. The code formula lattice constructed from a family of codes that satisfies the Schur-product condition is equivalent to the lattice constructed from the same family of codes by

Construction D [22, 9, 21]. Construction-D lattices are primarily used in communication settings, e.g. see Forney [12].

In this work we design a tester for code formula lattices using testers for the constituent codes.

► **Theorem 7.** *Let  $0 < \epsilon, s < 1$  and  $C_0 \subseteq C_1 \subseteq \dots \subseteq C_{m-1} \subseteq \{0, 1\}^n$  be a family of binary linear codes satisfying the Schur product condition. Suppose every  $C_i$  has a 1-sided tester  $T_i(\epsilon/m2^{i+1}, 0, s, q_i)$ . Then, there exists an  $\ell_1$ -tester  $T(\epsilon, 0, s, q)$  for the lattice  $L(\langle C_i \rangle_{i=0}^{m-1})$  with query complexity*

$$q = O\left(\frac{1}{\epsilon} \log \frac{1}{s}\right) + \sum_{i=1}^{m-1} q_i.$$

Next, we show a lower bound on the query complexity for testing membership in code formula lattices, using lower bounds for testing membership in the constituent codes.

► **Theorem 8.** *Let  $0 < \epsilon, c, s < 1$  and  $C_0 \subseteq C_1 \subseteq \dots \subseteq C_{m-1} \subseteq \{0, 1\}^n$  be a family of binary linear codes satisfying the Schur product condition. Let  $q_i = q_i(\epsilon, c, s)$  be such that any (possibly adaptive, 2-sided)  $\ell_1$ -tester  $T_i(\epsilon, c, s, q')$  for  $C_i$  satisfies  $q' = \Omega(q_i)$ , for every  $i = 0, 1, \dots, m-1$ . Then every (possibly adaptive, 2-sided)  $\ell_1$ -tester  $T(\epsilon, c, s, q)$  for the lattice  $L(\langle C_i \rangle_{i=0}^{m-1})$  has query complexity*

$$q = \Omega\left(\max\left\{\frac{1}{\epsilon} \log \frac{1}{s}, \max_{i=0,1,\dots,m-1} q_i\right\}\right).$$

*Code formula lattices from Reed-Muller codes.* We instantiate the upper and lower bounds on the query complexity for a common family of code formula lattices constructed using Reed-Muller codes [12] to obtain nearly matching upper and lower bounds. We recall Reed-Muller codes below.

► **Definition 9 (Reed Muller Codes).** Each codeword of a binary Reed-Muller code  $RM(k, r) \subseteq \mathbb{F}_2^{2^r}$  corresponds to a polynomial  $p(x) \in \mathbb{F}_2[x]$  in  $r$  variables of degree at most  $k$  evaluated at all  $2^r$  possible inputs  $x \in \mathbb{F}_2^r$ .

For the family of Reed-Muller codes in  $\mathbb{F}_2^{2^r}$ , it is well-known that  $RM(0, r) \subseteq RM(1, r) \subseteq RM(2, r) \subseteq RM(3, r) \subseteq \dots \subseteq RM(r-1, r) \subseteq RM(r, r) = \mathbb{F}_2^{2^r}$ . A particular family of RM codes that leads to code formula lattices is  $\langle RM(k_i, r) \rangle_{i=0}^{\log r}$ , with  $k_i = 2^i$ . Indeed, it can be easily verified that this family satisfies the Schur product condition since Reed-Muller codewords are evaluation tables of multivariate polynomials over the binary field and product of two degree  $k$  polynomials is a degree  $2k$  polynomial. Hence for height  $m \leq \log r$  the construction  $\langle RM(2^i, r) \rangle_{i=0}^{m-1}$  gives rise to a lattice. We note these lattices have small relative minimum distance and are not asymptotically good families of lattices.

► **Corollary 10.** *Let  $0 \leq k_0 < k_1 < \dots < k_{m-1} < r$  be integers such that the family of Reed-Muller codes  $RM(k_0, r) \subseteq RM(k_1, r) \subseteq \dots \subseteq RM(k_{m-1}, r)$  satisfies the Schur product condition. Let  $0 < \epsilon, s < 1$  and  $L$  be the lattice obtained from this family of codes using the code formula construction:*

$$L = RM(k_0, r) + 2RM(k_1, r) + \dots + 2^{m-1}RM(k_{m-1}, r) + 2^m\mathbb{Z}^{2^r}.$$

*Then, there exists an  $\ell_1$ -tester  $T(\epsilon, 0, s, q)$  for  $L$  with query complexity*

$$q(\epsilon, s) = O\left(2^{k_{m-1}} \cdot \frac{1}{\epsilon} \log \frac{1}{s}\right).$$

In particular, when the height  $m$  and the degrees are constant, the query complexity of the tester is a constant.

For the lower bound, we obtain the following corollary using known lower bounds for testing Reed-Muller codes.

► **Corollary 11.** *Let  $0 \leq k_0 < k_1 < \dots < k_{m-1} < r$  be integers such that the family of Reed-Muller codes  $RM(k_0, r) \subseteq RM(k_1, r) \subseteq \dots \subseteq RM(k_{m-1}, r)$  satisfies the Schur product condition. Let  $0 < \epsilon, c, s < 1$  be constants and  $L$  be the lattice obtained from this family of codes using the code formula construction:*

$$L = RM(k_0, r) + 2RM(k_1, r) + \dots + 2^{m-1}RM(k_{m-1}, r) + 2^m\mathbb{Z}^{2^r}.$$

Then, every (possibly 2-sided, adaptive)  $\ell_1$ -tester  $T(\epsilon, c, s, q)$  for  $L$  has query complexity

$$q = \Omega(2^{k_{m-1}}).$$

We note that for code formula lattices obtained from Reed-Muller codes, Corollaries 10 and 11 show matching bounds (up to a constant factor depending on  $\epsilon, s$ ).

**Random lattices.** There exists a distribution of random lattices which are impossible to test with small number of queries. This follows from Theorem 8 and considering random codes, which typically need at least a linear number of queries to test. We illustrate a concrete example by considering the following distribution of random lattices [11, 4]: For constants  $b < a$ , let  $m = nb/a$  and let  $H \in \mathbb{F}_2^{m \times n}$  be a random matrix such that each row and column has exactly  $a$  and  $b$  non-zeroes respectively. Consider the linear code  $C_{a,b} := \{x \in \mathbb{F}_2^n : Hx = 0 \pmod{2}\}$  and the code formula lattice  $L(C_{a,b})$  associated with the linear code  $C_{a,b}$ .

► **Theorem 12.** *There exist constants  $a, b, \epsilon, c, s$  such that every (possibly 2-sided, adaptive)  $\ell_1$ -tester  $T(\epsilon, c, s, q)$  for  $L(C_{a,b})$  has query complexity  $q = \Omega(n)$ .*

The above theorem follows as an immediate corollary of Theorem 8 and Theorem 3.7 of [4].

### 1.3.2 Tolerant testing code formulas

We also obtain upper bounds for tolerantly testing code formula lattices.

► **Theorem 13.** *Let  $0 < \epsilon_1, \epsilon_2, c, s < 1$  and  $C_0 \subseteq C_1 \subseteq \dots \subseteq C_{m-1} \subseteq \{0, 1\}^n$  be a family of binary linear codes satisfying the Schur product condition. Suppose every  $C_i$  has a tolerant tester  $T_i(2\epsilon_1, \frac{\epsilon_2}{m^{2^{i+1}}}, \frac{c}{m+1}, s, q_i)$ . Let  $\gamma = \min\{c/(m+1), s\}$ ,  $\epsilon_2 > m2^{m+1}\epsilon_1$ . Then there exists an  $\ell_1$ -tolerant-tester  $T(\epsilon_1, \epsilon_2, c, s, q)$  for the lattice  $L(\langle C_i \rangle_{i=0}^{m-1})$  with query complexity*

$$q = O\left(\frac{1}{(\epsilon_2 - 2\epsilon_1)^2} \log\left(\frac{1}{\gamma}\right)\right) + \sum_{i=0}^{m-1} q_i.$$

► **Theorem 14.** *Let  $0 \leq k_0 < k_1 < \dots < k_{m-1} < r$  be integers such that the family of Reed-Muller codes  $RM(k_0, r) \subseteq RM(k_1, r) \subseteq \dots \subseteq RM(k_{m-1}, r)$  satisfies the Schur product condition. Let  $L$  be the lattice obtained from this family of codes using the code formula construction:*

$$L = RM(k_0, r) + 2RM(k_1, r) + \dots + 2^{m-1}RM(k_{m-1}, r) + 2^m\mathbb{Z}^{2^r}.$$

Then there exists a  $\ell_1$ -tolerant-tester  $T(\epsilon_1, \epsilon_2, 1/3, 1/3, q)$  for  $L$  for all  $\epsilon_1 \leq \frac{c'_1}{2^{k_{m-1}}}$ ,  $\epsilon_2 \geq \frac{c'_2 m}{2^{k_0-1}}$  (for some constants  $c'_1$  and  $c'_2$ ) with query complexity  $q = O(2^{k_{m-1}} \cdot \log m)$ .

### 1.3.3 A canonical/linear test for lattices

We show a reduction from any given arbitrary test to a *canonical linear test*, thus suggesting that it is sufficient to design *canonical linear tests* for achieving low query complexity. In order to describe the intuition behind a canonical linear test, we first illustrate how to solve the membership testing problem when all coordinates of the input are known. For a given lattice  $L$ , its *dual lattice* is defined as

$$L^\perp := \{u \in \text{span}(L) \mid \langle u, v \rangle \in \mathbb{Z}, \text{ for all } v \in L\}.$$

It is easy to verify that  $(L^\perp)^\perp = L$ . Furthermore, a vector  $v \in L$  if and only if for all  $u \in L^\perp$ , we have  $\langle u, v \rangle \in \mathbb{Z}$ . Thus, to test membership of  $t$  in  $L$  in the classical decision sense, it is sufficient to verify whether  $t$  has integer inner products with a set of basis vectors of the dual lattice  $L^\perp$ . Inspired by this observation, we define a canonical *linear test* for lattices as follows. For a lattice  $L \subseteq \mathbb{R}^n$  and  $J \subseteq [n]$ , let  $L_J^\perp := \{x \in L^\perp \mid \text{supp}(x) \subseteq J\}$ , where  $\text{supp}(x)$  is the set of non-zero indices of the vector  $x$ .

► **Definition 15** (Linear Tester). A *linear tester* for a lattice  $L \subseteq \mathbb{Z}^n$  is a probabilistic algorithm which queries a subset  $J = \{j_1, \dots, j_q\} \subseteq [n]$  of coordinates of the input  $t \in \mathbb{R}^n$  and accepts  $t$  if and only if  $\langle t, x \rangle \in \mathbb{Z}$  for all  $x \in L_J^\perp$ .<sup>2</sup>

► **Remark.** By definition, the probabilistic choices of a linear tester are only over the set of coordinates to be queried: upon fixing the coordinate queries, the choice of the algorithm to accept or reject is fully determined. Furthermore, a linear tester is 1-sided since if the input  $t$  is a lattice vector, then for every dual vector  $u \in L^\perp$ , the inner product  $\langle u, t \rangle$  is integral, and so it will be accepted with probability 1.

We show that non-adaptive linear tests are nearly as powerful as 2-sided adaptive tests for a full-rank lattice. We reduce any (possibly 2-sided, and adaptive) test for a full-rank lattice to a non-adaptive linear test for the same distance parameter  $\epsilon$ , with a small increase in the query complexity and the soundness error.

► **Theorem 16.** *Let  $L \subseteq \mathbb{Z}^n$  be a lattice with  $\text{rank}(L) = n$ . If there exists an adaptive 2-sided  $\ell_p$ -tester  $T(\epsilon, c, s, q)$  with query complexity  $q = q_T(\epsilon, c, s)$ , then there exists a non-adaptive linear  $\ell_p$ -tester  $T'(\epsilon, 0, c + s, q')$  with query complexity  $q' = q_T(\epsilon/2, c, s) + O((1/\epsilon^p) \log(1/s))$ .*

Furthermore, if we are guaranteed that the inputs are in  $\mathbb{Z}^n$ , then the query complexity of the test  $T'$  above can be improved to be identical to that of  $T$  (up to a constant factor in the  $\epsilon$  parameter). The increase in the query complexity comes from an extra step used to verify the integrality of the input.

Theorem 16 suggests that, for the purposes of designing a tester with small query complexity, it is sufficient to design a non-adaptive linear tester, i.e., it suffices to only identify the probability distribution for the coordinates that are queried. Moreover, this theorem makes progress towards Question 3, since it shows that a lower bound on the query complexity of non-adaptive linear tests for a particular lattice implies a lower bound on the query complexity of all tests for that lattice. Thus in order to understand the existence of low query complexity tester for a particular lattice, it is sufficient to examine the existence of low query complexity *non-adaptive linear* tester for that lattice.

We note that Theorem 16 is the analogue of the result of [4] for linear error-correcting codes. In section 2, we comment on the comparison between our proof and that in [4].

<sup>2</sup> Verifying whether  $\langle t, x \rangle \in \mathbb{Z}$  for all  $x \in L_J^\perp$  can be performed efficiently by checking inner products with a set of basis vectors of the lattice  $L_J^\perp$ .

### 1.3.4 Testing membership of inputs outside the span of the lattice

We also observe a stark difference between the membership testing problem for a linear code, and the membership testing problem for a lattice. In the membership testing problem for a linear code  $C \subseteq \mathbb{F}^n$  defined over a finite field that is specified by a basis, the input is assumed to be a vector in  $\mathbb{F}^n$  and the goal is to verify whether the input lies in the span of the basis (see definition 5). As opposed to codes, for a lattice  $L \subseteq \mathbb{R}^n$ , the input is an arbitrary real vector, and the goal is to verify whether the input is a member of  $L$ , and not to verify whether the input is a member of the span of the lattice. Thus, the inputs to the lattice membership testing problem could lie either in  $\text{span}(L)$ , or outside  $\text{span}(L)$ . Interestingly, for some lattices it is easy to show strong lower bounds on the query complexity if the inputs are allowed to lie outside  $\text{span}(L)$ , thus suggesting that such inputs are hard to test.

► **Theorem 17.** *Let  $L \subseteq \mathbb{Z}^n$  be a lattice of rank  $k$ . Let  $P \subseteq [n]$  be the support of the vectors in  $\text{span}(L)^\perp$ . Let  $0 < \epsilon, c, s < 1$ . Every non-adaptive  $\ell_p$ -tester  $T(\epsilon, c, s, q)$  for  $L$  for inputs in  $\mathbb{R}^n$  has query complexity*

$$q = \Omega(|P|).$$

On the other hand, testers for inputs in the  $\text{span}(L)$  can be lifted to obtain testers for all inputs (including inputs that could possibly lie outside  $\text{span}(L)$ ).

► **Theorem 18.** *Let  $L \subseteq \mathbb{Z}^n$  be a lattice of rank  $k$ . Let  $P \subseteq [n]$  be the support of the vectors in  $\text{span}(L)^\perp$ . Let  $0 < \epsilon, c, s < 1$ , and suppose  $L$  has an  $\ell_p$ -tester  $T(\epsilon, c, s, q)$  for inputs  $t \in \text{span}(L)$ . Then  $L$  has a tester  $T'(2\epsilon, c, s, q')$  for inputs in  $\mathbb{R}^n$  with query complexity*

$$q' \leq q + |P|.$$

Theorem 18 implies that for lattices  $L$  of rank at most  $n - 1$ , if the membership testing problem for inputs that lie in  $\text{span}(L)$  is solvable using a small number of queries and if  $\text{span}(L)^\perp$  is supported on few coordinates, then the membership testing problem for all inputs (including those that do not lie in  $\text{span}(L)$ ) is solvable using a small number of queries.

**Knapsack Lattices.** Theorem 17 implies a linear lower bound for non-adaptively testing a well-known family of lattices, known as *knapsack lattices*, which have been investigated in the quest towards lattice-based cryptosystems [25, 34, 29]. We recall that a knapsack lattice is generated by a set of basis vectors  $B = \{b_1, \dots, b_{n-1}\}$ ,  $b_i \in \mathbb{R}^n$  that are of the form

$$\begin{aligned} b_1 &= (1, 0, \dots, 0, a_1) \\ b_2 &= (0, 1, \dots, 0, a_2) \\ &\vdots \\ b_{n-1} &= (0, 0, \dots, 1, a_{n-1}) \end{aligned}$$

where  $a_1, \dots, a_n$  are integers. We denote such a knapsack lattice by  $L_{a_1, \dots, a_{n-1}}$ .

► **Corollary 19.** *Let  $a_1, \dots, a_n$  be integers and  $0 < \epsilon, c, s < 1$ . Every non-adaptive  $\ell_p$ -tester  $T(\epsilon, c, s, q)$  for  $L_{a_1, \dots, a_n}$  has query complexity*

$$q = \Omega(n).$$

However, knapsack lattices with bounded coefficients are testable with a constant number of queries if the inputs are promised to lie in  $\text{span}(L)$ .

► **Theorem 20.** *Let  $a_1, \dots, a_n$  be integers with  $M = \max_{i \in [n]} |a_i|^p$  and  $0 < \epsilon, s < 1$ . There exists a non-adaptive  $\ell_p$ -tester  $T(\epsilon, 0, s, q)$  for  $L_{a_1, \dots, a_n}$  with query complexity  $q = O\left(\frac{M}{\epsilon^p} \cdot \log \frac{1}{s}\right)$ , if the inputs are guaranteed to lie in  $\text{span}(L)$ .*

Theorem 20 indicates that the large lower bound suggested by Theorem 17 could be circumvented for certain lattices if we are promised that the inputs lie in  $\text{span}(L)$ . The assumption that the input lies in  $\text{span}(L)$  is natural in decoding problems for lattices.

## 2 Overview of the proofs

### 2.1 Upper and lower bounds for testing general code formula lattices

The constructions of a tester for Theorem 7 and a tolerant tester for Theorem 13 follow the natural intuition that in order to test the lattice one can test the underlying codes individually. The proof relies on a triangle inequality that can be derived for such lattices. The application to code-formula lattices constructed from Reed-Muller codes follows from the tight analysis of Reed-Muller code testing from [6], which guarantees constant rejection probability of inputs that are at distance proportional to the minimum distance of the code. We note that the time complexity of the code-formula tester is given by the sum of the run-times of the component code testers. Since the component code testers can be assumed to be linear, and hence efficient, the code-formula lattice tester is also efficient.

While the tester that we construct from code testers for the purposes of proving Theorem 7 is an adaptive linear test, there is a simple variant that is a non-adaptive linear test with at least as good correctness and soundness. (see Remark 5.16 in full version [8] for a formal description).

The lower bound (Theorem 8) relies on the fact that if an input  $t$  is far from the code  $C_k$  in the code formula construction, then the vector  $2^k t$  is far from the lattice. Moreover, if  $t \in C_k$  then  $2^k t$  belongs to the lattice. Therefore a test for the lattice can be turned into a test for the constituent codes.

### 2.2 From general tests to canonical tests

We briefly outline our reduction for Theorem 16. Suppose  $T(\epsilon, c, s, q)$  is a 2-sided, adaptive tester with query complexity  $q = q_T(\epsilon, c, s)$  for a full rank integral lattice  $L$ . Such a tester handles all real-valued inputs. We first restrict  $T$  to a test that processes only integral inputs in the bounded set  $\mathcal{Z}_d = \{0, 1, \dots, d-1\}$  (for some carefully chosen  $d$ ), and so the restricted test inherits all the parameters of  $T$ . We remark that  $\mathcal{Z}_d \subset \mathbb{Z}$  is a subset of integers, and it should not be confused with  $\mathbb{Z}_d$ , the ring of integers modulo  $d$ .

A key ingredient in our reduction is choosing the appropriate value of  $d$  in order to enable the same guarantees as that of codes. We choose  $d$  such that  $d\mathbb{Z}^n \subseteq L$ . Such a  $d$  always exists [27]. This choice of  $d$  allows us to add any vector in  $V = L \bmod d$  (embedded in  $\mathbb{R}^n$ ) to any vector  $x \in \mathbb{R}^n$  without changing the distance of  $x$  to  $L$  in any  $\ell_p$ -norm (see Proposition 22).

Since our inputs are now integral and bounded, any adaptive test can be viewed as a distribution over deterministic tests, which themselves can be viewed as decision trees. This allows us to proceed along the same lines as in the reduction for codes over finite fields of [4].

We exploit the property that adding any vector in  $V$  to any vector  $x \in \mathbb{R}^n$  does not change the distance to  $L$ . In the first step of our reduction we add a random vector in  $V$  to the input and perform a probabilistic *linear* test. The idea is that one can relabel the decision tree of any test according to the decision tree of a linear test, such that the error shifts from the positive (yes) instances to the negative (no) instances (see Lemma 23). A

simple property of lattices used in this reduction is that if the set of queries  $I$  and answers  $a_I$  do not have a local witness for non-membership in the lattice (in the form of a dual lattice vector  $v$  supported on  $I$  such that  $\langle w_I, v_I \rangle \notin \mathbb{Z}$ ), then there exists  $w \in L$  that extends  $a_I$  to the remaining set of coordinates (i.e.,  $a_I = w_I$ ).

In the next step we remove the adaptive aspect of the test to obtain a non-adaptive linear test for inputs in  $\mathcal{Z}_d^n$  (see Lemma 24). We obtain this tester by performing the adaptive queries on a randomly chosen vector in  $V$  (and not on the input itself) and rejecting/accepting according to whether there exists a local witness for the non-membership of the input queried on the same coordinates.

We then lift this test to a non-adaptive linear test for inputs in  $\mathbb{Z}^n$ , by simulating the test over  $\mathcal{Z}_d^n$  on the same queried coordinates but using the answers obtained after taking modulo  $d$ . Owing to the choice of  $d$ , this does not change the distance of the input to the lattice (see Lemma 25).

Finally, we extend this test to a non-adaptive linear test for inputs in  $\mathbb{R}^n$  by performing some additional queries to rule out inputs that are not in  $\mathbb{Z}^n$ . For this, we design a tester for the integer lattice  $\mathbb{Z}^n$  with query complexity  $O((1/\epsilon^p) \log(1/s))$ . This final step of testing integrality increases the overall query complexity to  $q_T(\epsilon/2, c, s) + O((1/\epsilon^p) \log(1/s))$  (see Lemma 26).

**Organization.** We present the formal lemmas needed to prove Theorem 16 in Section 3. We refer the reader to the full version [8] for all the missing proofs.

### 3 Reducing an arbitrary test to a non-adaptive linear test

In this section we sketch the proof of Theorem 16. Throughout this section, we focus on full-rank integral lattices. Given a 2-sided adaptive  $\ell_p$ -tester  $T(\epsilon, c, s, q)$ , with  $q = q_T(\epsilon, c, s)$  for an integral lattice  $L$ , we construct a non-adaptive linear  $\ell_p$ -tester  $T'(\epsilon, 0, c + s, q)$  with query complexity  $q' = q_T(\epsilon/2, c, s) + O((1/\epsilon^p) \log(1/s))$ . We reduce the inputs to a bounded set using the following property of integral lattices.

► **Fact 21** ([27]). *Given any full rank integral lattice  $L$ , there exists  $d \in \mathbb{Z}$  such that  $d \cdot \mathbb{Z}^n \subseteq L$ . In particular  $|\det(L)| \cdot \mathbb{Z}^n \subseteq L$  for any lattice (where  $\det(L)$  denotes the determinant of a lattice, a parameter that can be computed given a basis of the lattice). For instance, we can take  $d = 2^m$  for the lattices of height  $m$  obtained using the code formula construction.*

Let  $V = L \bmod d$  embedded in  $\mathbb{Z}^n$  (i.e., we treat  $V$  as a set of vectors in  $\mathbb{Z}^n$  each of which is obtained by taking coordinate-wise modulo  $d$  of some lattice vector). Thus,  $V \subseteq \mathcal{Z}_d^n$ . We will need the following properties of  $V$ .

► **Proposition 22.** *Let  $L \subseteq \mathbb{Z}^n$  be a full-rank lattice,  $d \in \mathbb{Z}_+$  such that  $d\mathbb{Z}^n \subseteq L$ , and let  $V = L \bmod d \subseteq \mathbb{Z}^n$ . Then  $V$  satisfies the following properties:*

1.  $v \in L$  if and only if  $v \bmod d \in V$ .
2.  $V = L \cap \mathcal{Z}_d^n$ .
3.  $(v + V) \bmod d \subseteq V$  if and only if  $v \in L$ .
4. For any  $v \in \mathbb{Z}^n$ ,  $d_p(v, L) = d_p(v \bmod d, L)$ .

Theorem 16 will immediately follow by combining Lemmas 23, 24, 25, and 26.

► **Lemma 23.** *Suppose a full-rank lattice  $L \subseteq \mathbb{Z}^n$  with  $d\mathbb{Z}^n \subseteq L$  for  $d \in \mathbb{Z}_+$  has an adaptive 2-sided  $\ell_p$ -tester  $T(\epsilon, c, s, q)$  for inputs from the domain  $\mathcal{Z}_d^n$ . Then  $L$  has an adaptive linear  $\ell_p$ -tester  $T'(\epsilon, 0, c + s, q)$  for inputs from the domain  $\mathcal{Z}_d^n$ .*



► **Lemma 24.** *Suppose a full-rank lattice  $L \subseteq \mathbb{Z}^n$  with  $d\mathbb{Z}^n \subseteq L$  for  $d \in \mathbb{Z}_+$  has an adaptive linear  $\ell_p$ -tester  $T(\epsilon, 0, s, q)$  for inputs from the domain  $\mathcal{Z}_d^n$ . Then  $L$  has a non-adaptive linear  $\ell_p$ -tester  $T'(\epsilon, 0, s, q)$  for inputs from the domain  $\mathcal{Z}_d^n$ .*

► **Lemma 25.** *Let  $L \subseteq \mathbb{Z}^n$  be a full-rank lattice with  $d\mathbb{Z}^n \subseteq L$  for  $d \in \mathbb{Z}_+$ . Then,  $L$  has a non-adaptive linear  $\ell_p$ -tester  $T(\epsilon, 0, s, q)$  for inputs from the domain  $\mathcal{Z}_d^n$  if and only if  $L$  has a non-adaptive linear  $\ell_p$ -tester  $T'(\epsilon, 0, s, q)$  for inputs from the domain  $\mathbb{Z}^n$ .*

► **Lemma 26.** *Suppose a full-rank lattice  $L \subseteq \mathbb{Z}^n$  has a non-adaptive  $\ell_p$ -tester  $T(\epsilon, c, s, q)$  for inputs from the domain  $\mathbb{Z}^n$ . Then there exists a non-adaptive  $\ell_p$ -tester  $T'(\epsilon, c, s, q')$  for inputs in  $\mathbb{R}^n$  with query complexity  $q' = q(\epsilon/2, c, s) + O((1/\epsilon^p) \log(1/s))$ . Moreover, if  $T$  is a linear tester, then so is  $T'$ .*

The proof of Lemma 26 uses the following tester for integer lattices which is based on querying a random collection of coordinates and verifying whether all of them are integral.

► **Lemma 27.** *For every  $0 < \epsilon \leq 1$  and every  $0 < s \leq 1$ , there exists a non-adaptive linear  $\ell_p$ -tester  $T_p(\epsilon, 0, s, q_Z)$  for  $\mathbb{Z}^n$  with query complexity*

$$q_Z = O\left(\frac{1}{\epsilon^p} \log \frac{1}{s}\right).$$

#### 4 Discussion

In this paper we defined a notion of local testing for a new family of objects: point lattices. Our results demonstrate connections between lattice testing and the ripe theory of locally testable codes, and brings up numerous avenues for further research (particularly, Questions 2 and 3).

We remark that the notion of being ‘ $\epsilon$ -far’ from the lattice may be defined differently than in Definition 1, depending on the application of interest. In particular, in applications like IP and cryptography, it is natural to ask for a notion of tester that ensures that scaling the lattice does not change the query complexity. An alternate definition of  $\epsilon$ -far based on the *covering radius* of the lattice could be helpful to achieve this property. The covering radius of a lattice  $L \subseteq \mathbb{R}^n$  (similar to codes) is the largest distance of any vector in  $\mathbb{R}^n$  to the lattice. It is trivial to design a tester to verify if a point is in the lattice or at distance more than the covering radius from the lattice (simply accept all inputs). In order to have a tester notion where scaling preserves query complexity, we may define a vector as being  $\epsilon$ -far from the lattice, if the distance of the vector to every lattice point is at least  $\epsilon$  times the covering radius of the lattice. We note that the covering radius of any *integral lattice* is  $\Omega(\|1^n\|_p)$ . Indeed, the densest possible integral lattice, namely the integer lattice  $\mathbb{Z}^n$ , has covering radius  $(1/2)\|1^n\|_p$ , as exhibited by the point  $v = (1/2, \dots, 1/2) \in \mathbb{R}^n$ . Thus, by asking the tester to reject points at distance more than  $\epsilon\|1^n\|_p$  in Definition 1, we have settled upon a strong notion of being  $\epsilon$ -far from the lattice (i.e., the definition would in particular imply that vectors that are farther than  $\epsilon$  times the covering radius would be rejected by the tester). This definition is essentially equivalent to the current Definition 1 if the covering radius of the lattice is  $\Theta(n)$ . With the modified definition of local testers using covering radius as described above, the equivalent Question 1 is to identify a family of lattices that can be tested using a constant number of queries, achieves constant rate and whose ratio of minimum distance to covering radius is also at least a constant.

**Acknowledgments.** We thank Chris Peikert for mentioning to us about the potential application to cryptanalysis, and anonymous reviewers for helpful comments and pointers.

## References

- 1 Dorit Aharonov and Oded Regev. Lattice problems in  $NP \cap coNP$ . *J. ACM*, 52(5):749–765, 2005.
- 2 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
- 3 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.
- 4 E. Ben-Sasson, P. Harsha, and S. Raskhodnikova. Some 3CNF properties are hard to test. *SIAM Journal on Computing*, 35(1):1–21, 2005. Earlier version in STOC’03.
- 5 Piotr Berman, Sofya Raskhodnikova, and Grigory Yaroslavtsev.  $L_p$ -testing. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 164–173, 2014.
- 6 Arnab Bhattacharyya, Swastik Kopparty, Grant Schoenebeck, Madhu Sudan, and David Zuckerman. Optimal testing of Reed-Muller codes. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23–26, 2010, Las Vegas, Nevada, USA*, pages 488–497, 2010.
- 7 M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47:549–595, 1993.
- 8 Karthekeyan Chandrasekaran, Mahdi Cheraghchi, Venkata Gandikota, and Elena Grigorescu. Local testing for membership in lattices. *arXiv preprint arXiv:1608.00180*, 2016.
- 9 J. Conway, N. J. A. Sloane, and E. Bannai. *Sphere Packings, Lattices and Groups*. A series of comprehensive studies in mathematics. Springer, 1999.
- 10 Friedrich Eisenbrand. Fast integer programming in fixed dimension. In *Algorithms – ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16–19, 2003, Proceedings*, pages 196–207, 2003.
- 11 Uri Erez, Simon Litsyn, and Ram Zamir. Lattices which are good for (almost) everything. *IEEE Transactions on Information Theory*, 51(10):3401–3416, 2005.
- 12 G. D. Forney. Coset codes-I: Introduction and geometrical classification. *IEEE Transactions on Information Theory*, 34(5):1123–1151, 1988.
- 13 K. Friedl and M. Sudan. Some improvements to low-degree tests. In *Proceedings of the 3rd Annual Israel Symposium on Theory and Computing Systems*, 1995.
- 14 Philippe Gaborit and Gilles Zémor. On the construction of dense lattices with a given automorphisms group. *Annales de l’institut Fourier*, 57(4):1051–1062, 2007.
- 15 Oded Goldreich. Short locally testable codes and proofs: A survey in two parts. In *Property Testing – Current Research and Surveys*, pages 65–104, 2010.
- 16 Venkatesan Guruswami and Atri Rudra. Tolerant locally testable codes. In *Proceedings of RANDOM/APPROX 2005*, pages 306–317, 2005.
- 17 Ravi Kannan. Minkowski’s convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, August 1987.
- 18 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103, 1972.
- 19 T. Kaufman and M. Sudan. Algebraic property testing: The role of invariance. In *STOC*, pages 403–412, 2008.
- 20 Swastik Kopparty and Shubhangi Saraf. Tolerant linearity testing and locally testable codes. In *Proceedings of RANDOM*, pages 601–614, 2009.
- 21 Wittawat Kositwattanarerk and Frédérique E. Oggier. Connections between construction D and related constructions of lattices. *Des. Codes Cryptography*, 73(2):441–455, 2014.
- 22 John Leech and N. J. A. Sloane. Sphere packings and error-correcting codes. *Canad. J. Math*, 23(4):718–745, 1971.

- 23 H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.
- 24 Yi-Kai Liu, Vadim Lyubashevsky, and Daniele Micciancio. On bounded distance decoding for general lattices. In *Proceedings of RANDOM*, pages 450–461, 2006.
- 25 Ralph C. Merkle and Martin E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978.
- 26 Daniele Micciancio. *The LLL Algorithm: Survey and Applications*, chapter Cryptographic functions from worst-case complexity assumptions, pages 427–452. Information Security and Cryptography. Springer, December 2009. Prelim. version in Proc. of LLL25, 2007.
- 27 Daniele Micciancio. Lecture notes on lattice algorithms and applications, Winter 2012, Lecture 2, 2012.
- 28 Daniele Micciancio and Shafi Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, March 2002.
- 29 Andrew M. Odlyzko. The rise and fall of knapsack cryptosystems. *Cryptology and computational number theory*, 42:75–88, 1990.
- 30 M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6):1012–1042, 2006.
- 31 Oded Regev. Lattice-based cryptography. In *Advances in Cryptology – CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20–24, 2006, Proceedings*, pages 131–141, 2006.
- 32 Oded Regev. The learning with errors problem (invited survey). In *IEEE Conference on Computational Complexity*, pages 191–204, 2010.
- 33 R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25:252–271, 1996.
- 34 Adi Shamir. A polynomial time algorithm for breaking the basic merkle-hellman cryptosystem. In *Advances in Cryptology*, pages 279–288. Springer, 1983.
- 35 Laurence A. Wolsey and George L. Nemhauser. *Integer and combinatorial optimization*. John Wiley & Sons, 2014.

# Super-Fast MST Algorithms in the Congested Clique Using $o(m)$ Messages\*

Sriram V. Pemmaraju<sup>1</sup> and Vivek B. Sardeshmukh<sup>2</sup>

1 Department of Computer Science, University of Iowa, Iowa City, USA  
sriram-pemmaraju@uiowa.edu

2 Department of Computer Science, University of Iowa, Iowa City, USA  
vivek-sardeshmukh@uiowa.edu

---

## Abstract

In a sequence of recent results (PODC 2015 and PODC 2016), the running time of the fastest algorithm for the *minimum spanning tree (MST)* problem in the *Congested Clique* model was first improved to  $O(\log \log \log n)$  from  $O(\log \log n)$  (Hegeman et al., PODC 2015) and then to  $O(\log^* n)$  (Ghaffari and Parter, PODC 2016). All of these algorithms use  $\Theta(n^2)$  messages independent of the number of edges in the input graph.

This paper positively answers a question raised in Hegeman et al., and presents the first “super-fast” MST algorithm with  $o(m)$  message complexity for input graphs with  $m$  edges. Specifically, we present an algorithm running in  $O(\log^* n)$  rounds, with message complexity  $\tilde{O}(\sqrt{m \cdot n})$  and then build on this algorithm to derive a family of algorithms, containing for any  $\varepsilon$ ,  $0 < \varepsilon \leq 1$ , an algorithm running in  $O(\log^* n/\varepsilon)$  rounds, using  $\tilde{O}(n^{1+\varepsilon}/\varepsilon)$  messages. Setting  $\varepsilon = \log \log n / \log n$  leads to the first sub-logarithmic round Congested Clique MST algorithm that uses only  $\tilde{O}(n)$  messages.

Our primary tools in achieving these results are

- (i) a component-wise bound on the number of candidates for MST edges, extending the sampling lemma of Karger, Klein, and Tarjan (Karger, Klein, and Tarjan, JACM 1995) and
- (ii)  $\Theta(\log n)$ -wise-independent linear graph sketches (Cormode and Firmani, Dist. Par. Databases, 2014) for generating MST candidate edges.

**1998 ACM Subject Classification** F.2.0 [Analysis of Algorithms and Problem Complexity] General, C.2.4 [Computer-Communication Networks] Distributed Systems

**Keywords and phrases** Congested Clique, Minimum Spanning Tree, Linear Graph Sketches, Message Complexity, Sampling

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.47

## 1 Introduction

The *Congested Clique* is a synchronous, message-passing model of distributed computing in which the underlying network is a clique and in each round, a message of size  $O(\log n)$  bits can be sent in each direction across each communication link. The Congested Clique is a simple, clean model for studying the obstacles imposed by congestion – all relevant information is nearby in the network (at most 1 hop away), but may not be able to travel to an intended node due to the  $O(\log n)$ -bit bandwidth restriction on the communication links. There has been a lot of recent work in studying various fundamental problems in the Congested Clique model, including facility location [9, 3], *minimum spanning tree (MST)*

---

\* This work is supported in part by National Science Foundation grant CCF 1318166.



© Sriram V. Pemmaraju and Vivek B. Sardeshmukh;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 47; pp. 47:1–47:15



Leibniz International Proceedings in Informatics

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

[23, 13, 11, 10], shortest paths and distances [4, 14, 26], triangle finding [7, 6], subgraph detection [7], ruling sets [3, 13], sorting [28, 22], and routing [22]. The modeling assumption in solving these problems is that the input graph  $G = (V, E)$  is “embedded” in the Congested Clique – each node of  $G$  is uniquely mapped to a machine and the edges of  $G$  are naturally mapped to the links between the corresponding machines (see Section 1.1).

The earliest non-trivial example of a Congested Clique algorithm is the *deterministic* MST algorithm that runs in  $O(\log \log n)$  rounds due to Lotker et al. [23]. Using *linear sketching* [1, 2, 15, 24, 5] and the *sampling* technique due to Karger, Klein, and Tarjan [16], Hegeman et al. [11] were able to design a substantially faster, *randomized* Congested Clique MST algorithm, running in  $O(\log \log \log n)$  rounds. Soon afterwards, Ghaffari and Parter [10] designed an  $O(\log^* n)$ -round algorithm, using the techniques in Hegeman et al., but supplemented with the use of *sparsity-sensitive sketching*, which is useful for sparse graphs and *random edge sampling*, which is useful for dense graphs.

**Our Contributions.** All of the MST algorithms mentioned above, essentially use the entire bandwidth of the Congested Clique model, i.e., they use  $\Theta(n^2)$  messages. From these examples, one might (incorrectly!) conclude that “super-fast” Congested Clique algorithms are only possible when the entire bandwidth of the model is used. In this paper, we focus on the design of MST algorithms in the Congested Clique model that have low *message complexity*, while still remaining “super-fast.” Message complexity refers to the number of messages sent and received by all machines over the course of an algorithm; in many applications, this is the dominant cost as it plays a major role in determining the running time and auxiliary resources (e.g., energy) consumed by the algorithm. In our main result, we present an  $O(\log^* n)$ -round algorithm that uses  $\tilde{O}(\sqrt{m \cdot n})$ <sup>1</sup> messages for an  $n$ -node,  $m$ -edge input graph. Two points are worth noting about this message complexity upper bound: (i) it is bounded above by  $\tilde{O}(n^{1.5})$  for all values of  $m$  and is thus substantially sub-quadratic, independent of  $m$  and (ii) it is bounded above by  $o(m)$  for all values of  $m$  that are super-linear in  $n$ , i.e., when  $m = \omega(n \text{ poly}(\log n))$ . We then extend this result to design a family of algorithms parameterized by  $\varepsilon$ ,  $0 < \varepsilon \leq 1$ , and running in  $O(\log^* n/\varepsilon)$  rounds and using  $\tilde{O}(n^{1+\varepsilon}/\varepsilon)$  messages. If we set  $\varepsilon = \log \log n / \log n$ , we get an algorithm running in  $O(\log^* n \cdot \log n / \log \log n)$  rounds and using  $\tilde{O}(n)$  messages. Thus we demonstrate the existence of a sub-logarithmic round MST algorithm using only  $O(n \cdot \text{poly}(\log n))$  messages, positively answering a question posed in Hegeman et al. [11]. We note that Hegeman et al. present an algorithm using  $\tilde{O}(n)$  messages that runs in  $O(\log^5 n)$  rounds. All of the round and message complexity bounds mentioned above hold with high probability (w.h.p.), i.e., with probability at least  $1 - \frac{1}{n}$ . Our results indicate that the power of the Congested Clique model lies not so much in its  $\Theta(n^2)$  bandwidth as in the flexibility it provides – any communication link that is needed is present in the network, though most communication links may eventually not be needed.

**Applications.** Optimizing message complexity as well as time complexity for Congested Clique algorithms has direct applications to the performance of distributed algorithms in other models such as the Big Data ( $k$ -machine) model [18], which was recently introduced to study distributed computation on large-scale graphs. Via a Conversion Theorem in [18] one can obtain fast algorithms in the Big Data model from Congested Clique algorithms that

---

<sup>1</sup> The notation  $\tilde{O}$  hides  $\text{poly}(\log n)$  factors.

have low time complexity *and* message complexity. Another related motivation comes from the connection between the Congested Clique model and the MapReduce model. In [12] it is shown that if a Congested Clique algorithm runs in  $T$  rounds and, in addition, has moderate message complexity then it can be simulated in the MapReduce model in  $O(T)$  rounds.

## 1.1 Technical Preliminaries

**Congested Clique model.** The *Congested Clique* is a set of  $n$  computing entities (nodes) connected through a complete network that provides point-to-point communication. Each node in the network has a distinct identifier of  $O(\log n)$  bits. At the beginning of the computation, each node knows the identities of all  $n$  nodes in the network and the part of the input assigned to it. The computation proceeds in synchronous rounds. In each round each node can perform some local computation and send a (*possibly different*) message of  $O(\log n)$  bits to each of its  $n - 1$  neighbors. It is assumed that both the computing entities and the communication links are fault-free. The Congested Clique model is therefore specifically geared towards understanding the role of the limited bandwidth as a fundamental obstacle in distributed computing, in contrast to other classical models for distributed computing that instead focus, e.g., on the effects of latency (the LOCAL model) or on the effects of both latency and limited bandwidth (the CONGEST model).

The input graph is assumed to be a spanning subgraph of the underlying communication network. Before the algorithm starts, each node knows the edges of the input graph incident on it and their (respective) weights. We assume that every edge weight can be represented with  $O(\log n)$  bits. For ease of exposition, we assume that edge weights are distinct; otherwise, without loss of generality (WLOG) we can “pad” each edge weight with the IDs of the two end points of the edge so as to distinguish the edges by weight while respecting their weight-based ordering. We require that when the algorithm ends, each node knows which of its incident edges belong to the output MST.

**Linear Sketches.** A key tool used by our algorithm is *linear sketches* [1, 2, 24]. Let  $\mathbf{a}_v$  denote a vector whose non-zero entries represent edges incident on  $v$ . A *linear sketch* of  $\mathbf{a}_v$  is a low-dimensional random vector  $\mathbf{s}_v$ , typically of size  $O(\text{poly}(\log n))$ , with two properties:

- (i) sampling from the sketch  $\mathbf{s}_v$  returns a non-zero entry of  $\mathbf{a}_v$  with uniform probability (over all non-zero entries in  $\mathbf{a}_v$ ) and
- (ii) when nodes in a connected component are merged, the sketch of the new “super node” is obtained by coordination-wise addition of the sketches of the nodes in the component.

The first property is referred to as  $\ell_0$ -sampling in the streaming literature [5, 24, 15] and the second property is *linearity*. The graph sketches used in [1, 2, 24] rely on the  $\ell_0$ -sampling algorithm by Jowhari et al. [15]. Sketches constructed using the Jowhari et al. [15] approach use  $\Theta(\log^2 n)$  bits per sketch, but require polynomially many mutually independent random bits to be shared among all nodes in the network. Sharing this volume of information is not feasible; it takes too many rounds and too many messages. So instead, we appeal to the  $\ell_0$ -sampling algorithm of Cormode and Firmani [5] which requires a family of  $\Theta(\log n)$ -wise independent hash functions, as opposed to hash functions with full-independence. Hegeman et al. [11] provide details of how the Cormode-Firmani approach can be used in the Congested Clique model to construct graph sketches. We summarize their result in the following theorem.

► **Theorem 1.1** (Hegeman et al. [11]). *Given an input graph  $G = (V, E)$ ,  $n = |V|$ , there is a Congested Clique algorithm running in  $O(1)$  rounds and using  $O(n \cdot \text{poly}(\log n))$  messages,*

at the end of which every node  $v \in V$  has computed a linear sketch  $\mathbf{s}_v$  of  $\mathbf{a}_v$ . The size of the computed sketch of a node is  $O(\log^4 n)$  bits. The  $\ell_0$ -sampling algorithm on sketch  $\mathbf{s}_v$  succeeds with probability at least  $1 - n^{-2}$  and, conditioned on success, returns an edge in  $\mathbf{a}_v$  with probability in the range  $[1/L_v - n^{-2}, 1/L_v + n^{-2}]$ , where  $L_v$  is the number of non-zero entries in  $\mathbf{a}_v$ .

**Concentration Bounds for sums of  $k$ -wise-independent random variables.** The use of  $k$ -wise-independent random variables, for  $k = \Theta(\log n)$ , plays a key role in keeping the time and message complexity of our algorithms low. The use of  $\Theta(\log n)$ -wise independent hash functions in the construction of linear sketches has been mentioned above. In the next subsection, we discuss the use of  $\Theta(\log n)$ -wise-independent edge sampling as a substitute for the fully-independent edge sampling of Karger, Klein, and Tarjan. For our analysis we use the following concentration bound on the sum of  $k$ -wise independent random variables, due to Schmidt et al. [33] and slightly simplified by Pettie and Ramachandran [31].

► **Theorem 1.2** (Schmidt et al. [33]). *Let  $X_1, X_2, \dots, X_n$  be a sequence of random  $k$ -wise independent 0-1 random variables with  $X = \sum_{i=1}^n X_i$ . If  $k \geq 2$  is even and  $C \geq \mathbf{E}[X]$  then:*

$$\Pr(|X - \mathbf{E}[X]| \geq T) \leq \left[ \sqrt{2} \cosh \left( \sqrt{k^3/36C} \right) \right] \cdot \left( \frac{kC}{eT^2} \right)^{k/2}.$$

We use the above theorem for  $k = \Theta(\log n)$  and  $C = T = \mathbf{E}[X]$ . Furthermore, in all instances in which we use this bound,  $\mathbf{E}[X] > k^3$  and therefore the contribution of the  $\cosh(\cdot)$  term is  $O(1)$ , whereas the contribution of the second term on the right hand side is smaller than  $1/n^c$  for any constant  $c$ .

**MST with Linear Message Complexity.** The “super-fast” MST algorithms mentioned so far [23, 11, 10] use  $\Theta(n^2)$  messages, independent of the number of edges in the input graph. One reason for this is that these algorithms rely on deterministic constant-round Congested Clique algorithms for routing and sorting due to Lenzen [22]. Lenzen’s algorithms do not attempt to explicitly conserve messages and need  $\Omega(n^{1.5})$  messages independent of the number of messages being routed or the number of keys being sorted. However, the above-mentioned MST algorithms do not need the full power of Lenzen’s algorithms. We design sorting and routing protocols that work in slightly restricted settings, but use only a linear number of messages (i.e., linear in the total number messages to be routed or keys to be sorted). Details of these protocols appear in the full version of the paper [30]. We use these protocols (instead of Lenzen’s protocols) as subroutines in the Ghaffari-Parter MST algorithm [10] to derive a version that uses only linear (up to a polylogarithmic factor) number of messages.

► **Theorem 1.3** (LINEARMESSAGES-MST). *There exists a Congested Clique MST algorithm running in  $O(\log^* n)$  rounds using  $\tilde{O}(m)$  messages w.h.p. on an input graph with  $n$  nodes and  $m$  edges.*

## 1.2 Algorithmic Overview

The high-level structure of our algorithm is simple. Suppose that the input is an  $n$ -node,  $m$ -edge graph  $G = (V, E)$ . We start by sparsifying  $G$  by sampling each edge with probability  $p$  and compute a *maximal minimum weight spanning forest*  $F$  of the resulting sparse subgraph  $H$ . Thus  $H$  contains  $O(m \cdot p)$  edges w.h.p. Now consider an edge  $\{u, v\}$  in  $G$  and add it to  $F$ ; if  $F + \{u, v\}$  contains a cycle and  $\{u, v\}$  is the heaviest edge in this cycle, then by



Tarjan’s “red rule” [34] the MST of  $G$  does not contain edge  $\{u, v\}$ . Ignoring all such edges leaves a set of edges that are candidates for being in the MST. We appeal to the well-known sampling lemma due to Karger, Klein, and Tarjan [16] (KKT sampling) that provides an estimate of the size of this set of candidates.

► **Definition 1.4** (*F*-light edge [16]). Let  $F$  be a forest in a graph  $G$  and let  $F(u, v)$  denote the path (if any) connecting  $u$  and  $v$  in  $F$ . Let  $w_F(u, v)$  denote the maximum weight of an edge on  $F(u, v)$  (if there is no path then  $w_F(u, v) = \infty$ ). We call an edge  $\{u, v\}$  *F-heavy* if  $w(u, v) > w_F(u, v)$ , and *F-light* otherwise.

► **Lemma 1.5** (KKT Sampling Lemma [16]). *Let  $H$  be a subgraph obtained from  $G$  by including each edge independently<sup>2</sup> with probability  $p$  and let  $F$  be the maximal minimum weight spanning forest of  $H$ . The number of *F*-light edges in  $G$  is at most  $n/p$ , w.h.p.*

As our next step we compute the set of *F*-light edges and in our final step, we compute an MST of the subgraph induced by the *F*-light edges. Thus, at a high level, our algorithm consists of two calls to an MST subroutine on sparse graphs, one with  $O(m \cdot p)$  edges and the other with  $O(n/p)$  edges. In between, these two calls is the computation of *F*-light edges. This overall algorithmic structure is clearly visible in Lines 5–7 in the pseudocode in Algorithm 1 MST-v1.

There are several obstacles to realizing this high-level idea in the Congested Clique model in order to obtain an algorithm that is “super-fast” and yet has low message complexity. The reason for sparsifying  $G$  and appealing to the KKT Sampling Lemma is the expectation that we would need to use fewer messages to compute an MST on a sparser input graph. However, as mentioned earlier, all of the existing “super-fast” MST algorithms use  $\Theta(n^2)$  messages and are insensitive to the number of edges in the input graph. In our first contribution, we develop a collection of simple, low-message-complexity distributed routing and sorting subroutines that we can use in the Ghaffari-Parter MST algorithm, allowing us to complete the two calls to the MST subroutine in  $O(\log^* n)$  rounds using  $\max\{O(m \cdot p), O(n/p)\}$  messages. Setting the sampling probability  $p$  in our algorithm to  $\sqrt{\frac{n}{m}}$  balances the two terms in the  $\max(\cdot, \cdot)$  and yields a message complexity of  $O(\sqrt{m \cdot n})$ . Due to space restrictions, this contribution is briefly mentioned in Section 1.1 and is described in detail in the full version of our paper [30].

Our second and *main* contribution (Section 3) is to show that the computation of *F*-light can be completed in  $O(1)$  rounds, while still using  $\tilde{O}(\sqrt{m \cdot n})$  messages. To explain the challenge of this computation we present two simple algorithmic scenarios:

- Suppose that we want each node  $u$  to perform a local computation to determine which of its incident edges from  $G$  are *F*-light. To do this, node  $u$  needs to know  $w_F(u, v)$  for all neighbors  $v$ . Thus  $u$  needs  $\text{degree}_G(u)$  pieces of information and overall this approach seems to require the movement of  $\Omega(m)$  pieces of information, i.e.,  $\Omega(m)$  messages.
- Alternately, we might want each node that knows  $F$  to be responsible for determining which edges in  $G$  are *F*-light. In this case, the obvious approach is to send queries of the type “Is edge  $\{u, v\}$  *F*-light?” to nodes that know  $F$ . This approach also requires  $\Omega(m)$  messages.

Various combinations of and more sophisticated versions of these ideas also require  $\Omega(m)$  messages. So the fundamental question is how do we determine the status (i.e., *F*-light or

<sup>2</sup> For reasons that will become clear later, our goal of keeping the message complexity low, does not allow us to assume full independence in this sampling. Instead we use  $\Theta(\log n)$ -wise independent sampling and show that a slightly weaker version of the KKT Sampling Lemma holds even with limited independence sampling.

$F$ -heavy) of  $m$  edges while exchanging far fewer than  $m$  messages? Below we outline two techniques we have developed in order to answer this question.

**Component-wise bound on number of  $F$ -light edges.** As mentioned above, the KKT Sampling Lemma upper bounds the total number of  $F$ -light edges by  $O(n/p)$ , which is  $O(\sqrt{m \cdot n})$  for  $p = \sqrt{n/m}$ . We show (in Corollary 3.5) that a slightly weaker bound (weaker by a logarithmic factor) holds even if the edge-sampling is done using an  $\Theta(\log n)$ -wise-independent sampler. If we could ensure that the total volume of communication is proportional to the number of  $F$ -light edges, we would achieve our goal of  $o(m)$  message complexity. To achieve this goal we show that the set of  $F$ -light edges has additional structure; they are “evenly distributed” over the components of  $F$ . To understand this imagine that  $F$  is constructed from  $H$  using Borůvka’s algorithm. Let  $\mathcal{C}^i = \{C_1^i, C_2^i, \dots\}$  be the set of components at the beginning of a phase  $i$  of the algorithm. For each component  $C_j^i \in \mathcal{C}^i$ , the algorithm picks a *minimum weight outgoing edge* (MWOE)  $e_j^i$  from  $F$ . Components are merged using edges  $e_j^i, j = 1, 2, \dots$  and we get a new set of components  $\mathcal{C}^{i+1}$ . Let  $L_j^i$  be the set of edges in  $G$  leaving component  $C_j^i$  with weight at most  $w(e_j^i)$ . We show in Lemma 3.4 that the set of all  $F$ -light edges is just the union of the  $L_j^i$ ’s, over all phases  $i$  and components  $j$  within Phase  $i$ . Furthermore, we show in Lemma 3.2 that the size of  $L_j^i$  for any  $i, j$  is bounded by  $\tilde{O}(1/p)$  w.h.p. This “even distribution” of  $F$ -light edges suggests that we could make each component  $C_j^i$  responsible for identifying the  $L_j^i$ -edges. Note that we don’t use distributed Borůvka’s algorithm to compute  $F$  because that would take  $\Theta(\log n)$  rounds. We compute  $F$  in  $O(\log^* n)$  rounds using LINEARMESSAGES-MST, the modified Ghaffari-Parter MST algorithm (see Theorem 1.3.).  $F$  is then gathered at each of a small number of nodes and each node who knows  $F$  completely simulates Borůvka’s algorithm *locally* on  $F$ , thus identifying the components  $C_j^i$  and their MWOE’s  $e_j^i$ .)

**Component-wise generation of  $F$ -light edges using linear sketches.** Linear sketches play a key role in helping nodes in each component  $C_j^i$  collectively compute all edges in  $L_j^i$ . For any node  $v$  and number  $x$ , let  $N_x(v)$  denote the set of neighbors of  $v$  that are connected to  $v$  via edges of weight less than  $x$ . Each node  $v \in C_j^i$  computes a  $w(e_j^i)$ -restricted sketch  $\mathbf{s}_v$ , i.e., a sketch of its neighborhood  $N_{w(e_j^i)}(v)$ , and sends it to the component leader of  $C_j^i$  who aggregates these sketches to compute a single component sketch. Sampling this sketch yields a single edge in  $L_j^i$ . Since  $L_j^i$  has  $\tilde{O}(1/p)$  edges, each node  $v \in C_j^i$  can send  $\tilde{O}(1/p)$  separate  $w(e_j^i)$ -restricted sketches to the component leader of  $C_j^i$  and the Coupon Collector argument ensures that this volume of sketches is enough to generate *all* edges incident in  $L_j^i$  w.h.p.

► **Remark.** The sampling approach of Karger, Klein, and Tarjan is used in a somewhat minor way in earlier Congested Clique MST algorithms [10, 11] and in fact in [19] it is shown that this sampling approach can be replaced by a simple, deterministic sparsification. However, KKT sampling and specifically its  $\Theta(\log n)$ -wise independent version that we use in the current algorithm seems crucial for ensuring low message complexity, while keeping the algorithms fast.

### 1.3 Related Work

It is important to point out that our algorithms are designed for the so-called KT1 [29] model, where every node initially knows the IDs of all its neighbors, in addition to its own ID. (In the Congested Clique model, this means that each node knows the IDs of all  $n$  nodes in the network.) If we drop this assumption and work in the so-called KT0 model [29], in which

nodes are unaware of IDs of neighbors, then it has been shown in [11] that  $\Omega(m)$  messages are needed by any Congested Clique MST algorithm (including randomized Monte Carlo algorithms, and regardless of the number of rounds) on an  $m$ -edge input graph. In fact, this lower bound is shown for the simpler graph connectivity problem.

There have also been some recent developments on simultaneously optimizing message complexity and round complexity for the MST problem in the CONGEST model. For example, in [27] it is shown that there exists a randomized (Las Vegas) algorithm that runs in  $\tilde{O}(\sqrt{n} + \text{diameter}(G))$  rounds and uses  $\tilde{O}(m)$  messages (both w.h.p.). This improves the message complexity of the well-known Kutten-Peleg algorithm [21], without sacrificing round complexity (up to polylogarithmic factors). The Kutten-Peleg algorithm runs in  $O(\sqrt{n} \log^* n + \text{diameter}(G))$  rounds, while using  $O(m + n^{1.5})$  messages. Note that the algorithm in [27] simultaneously matches the round complexity lower bound [8, 32] and the message complexity lower bound [20] for the MST problem.

The above-mentioned upper and lower bound results assume the KT0 model. In the KT1 model, the message complexity lower bound of Kutten et al. [20] does not hold and King et al. [17] were able to design an MST algorithm in the KT1 CONGEST model that uses  $\tilde{O}(n)$  messages, though this algorithm has significantly higher round complexity than  $\tilde{O}(\sqrt{n} + \text{diameter}(G))$  rounds.

As mentioned earlier, Hegeman et al. [11] present a Congested Clique MST algorithm using  $\tilde{O}(n)$  messages, but running in  $O(\log^5 n)$  rounds. One can make a few changes to the King et al. [17] CONGEST-model algorithm to implement it in the Congested Clique model, requiring  $\tilde{O}(n)$  messages, but running in  $O(\log^2 n / \log \log n)$  rounds.

## 2 MST Algorithms

In this section we describe two “super-fast” MST algorithms, the first runs in  $O(\log^* n)$  rounds, using  $\tilde{O}(\sqrt{m \cdot n})$  messages and the second algorithm running in  $O(\log^* n / \varepsilon)$  rounds, using  $\tilde{O}(n^{1+\varepsilon} / \varepsilon)$  messages, for any  $0 < \varepsilon \leq 1$ .

### 2.1 A super-fast algorithm using $\tilde{O}(\sqrt{mn})$ messages

Our first algorithm MST-v1, shown in Algorithm 1 has already been outlined in Section 1.2. The correctness, time complexity, and message complexity of this algorithm depends mainly on two subroutines: LINEARMESSAGES-MST( $\cdot$ ) and COMPUTE-F-LIGHT( $\cdot$ ). Recall that LINEARMESSAGES-MST( $H$ ) computes an MST on an  $n$ -node  $m$ -edge input graph  $H$  in  $O(\log^* n)$  rounds using  $\tilde{O}(m)$  messages (Theorem 1.3). We also show that COMPUTE-F-LIGHT( $G, F, p$ ) terminates in  $O(1)$  rounds using  $\tilde{O}(n/p)$  messages w.h.p. This is the main result in our paper and is shown in Section 3.

► **Lemma 2.1.** *For some constants  $c_1, c_2 > 1$ , (i)  $\Pr(|E(H)| > c_1 \cdot \sqrt{mn}) < \frac{1}{n}$  and (ii)  $\Pr(|E_\ell| > c_2 \cdot \sqrt{mn} \text{ poly}(\log n)) < \frac{1}{n}$ .*

**Proof.** For  $0 < i \leq m$ , let  $X_i = 1$  if edge  $i$  is sampled. Hence  $|E(H)| = \sum_i X_i$  and  $\mathbf{E}[|E(H)|] = \sqrt{mn}$ . Note that  $X_i$ 's are  $\Theta(\log n)$ -wise independent. Therefore, by Theorem 1.2 we have,  $\Pr(|E(H)| > c_1 \sqrt{mn}) < \frac{1}{n}$  for some suitable constant  $c_1 > 1$ . Claim (ii) follows from Corollary 3.5. ◀

The following theorem summarizes the properties of Algorithm MST-v1. The running time and message complexity bounds follow from Table 1.

**Algorithm 1** MST-v1

**Input:** An edge-weighted  $n$ -node,  $m$ -edge graph  $G = (V, E, w)$ .

▷ Each node knows weights and end-points of incident edges. Every weight can be represented using  $O(\log n)$  bits.

**Output:** An MST  $\mathcal{T}$  of  $G$ .

▷ Each node in  $V$  knows which of its incident edges are part of  $\mathcal{T}$ .

▷ Let  $v^*$  denote the node with lowest ID in  $V$ , known to all nodes.

- 1:  $v^*$  generates a sequence  $\pi$  of  $\Theta(\log^2 n)$  bits independently and uniformly at random and shares with all nodes in  $V$ .
- 2:  $p \leftarrow \sqrt{\frac{n}{m}}$
- 3: Each node constructs an  $\Theta(\log n)$ -wise-independent sampler from  $\pi$  and uses this to sample each incident edge in  $G$  with probability  $p$
- 4:  $H \leftarrow$  the spanning subgraph of  $G$  induced by the sampled edges
- 5:  $F \leftarrow$  LINEARMESSAGES-MST( $H$ )
- 6:  $E_\ell \leftarrow$  COMPUTE-F-LIGHT( $G, F, p$ )
- 7:  $\mathcal{T} \leftarrow$  LINEARMESSAGES-MST( $(V, E_\ell, w)$ )
- 8: **return**  $\mathcal{T}$

■ **Table 1** Time and message complexity for steps in Algorithm 1 MST-v1.

Step	Time	Messages	Analysis
1	$O(1)$	$\tilde{O}(n)$	Full paper
2–4	–	–	Local computation
5	$O(\log^* n)$	$\tilde{O}( E(H) )$	Theorem 1.3
6	$O(1)$	$\tilde{O}(\sqrt{mn})$	Theorem 3.6 with $p = \sqrt{\frac{n}{m}}$
7	$O(\log^* n)$	$\tilde{O}( E_\ell )$	Theorem 1.3

► **Theorem 2.2.** *Algorithm MST-v1 computes an MST of an edge-weighted  $n$ -node,  $m$ -edge graph  $G$  when it terminates. Moreover, it terminates in  $O(\log^* n)$  rounds and requires  $\tilde{O}(\sqrt{mn})$  messages w.h.p.*

## 2.2 Trading messages and time

The MST-v2 algorithm (shown in Algorithm 2) is a recursive version of MST-v1 algorithm yielding a time-message trade-off. The algorithm recurses until the number of edges in the subproblem becomes “low” enough to solve it via a call to the LINEARMESSAGES-MST subroutine. Specifically, we treat a  $n$ -node graph with  $m = O(n^{1+\varepsilon})$  edges as a base case. For graphs with more edges we use a sampling probability of  $p = 1/n^\varepsilon$ , leading to a sparse graph  $H$  with  $O(m/n^\varepsilon)$  edges w.h.p., which is recursively processed. The use of limited independence sampling is critical here. One simple approach to sampling an edge would be to let the endpoint with higher ID sample the edge and inform the other endpoint *if the outcome is positive*. Unfortunately, this would lead to the use of  $\tilde{O}(m/n^\varepsilon)$  messages w.h.p., exceeding our target of  $\tilde{O}(n^{1+\varepsilon})$  messages when  $m$  is large<sup>3</sup>. Using  $\Theta(\log n)$ -wise-independent sampling allows us to complete the sampling step using  $\tilde{O}(n)$  messages.

<sup>3</sup> This approach would have worked fine for MST-v1, but to keep the two algorithms consistent to the extent possible, we use the  $\Theta(\log n)$ -wise independent sampler there as well.

**Algorithm 2** MST-v2

---

**Input:** An edge-weighted  $n$ -node,  $m$ -edge graph  $G = (V, E, w)$   
 ▷ Each node knows weights and end-points of incident edges in  $G$ . Every weight can be represented using  $O(\log n)$  bits. There is a parameter  $0 < \varepsilon \leq 1$ , known to all nodes.  
**Output:** An MST  $\mathcal{T}$  of  $G$ .  
 ▷ Each node in  $V$  knows which of its incident edges are part of  $T$ .

---

▷ Let  $v^*$  denote the node with lowest ID in  $V$  and  $c \geq 1$  is a constant.

- 1: **if**  $m < c \cdot n^{1+\varepsilon}$  **then**
- 2:    $\mathcal{T} \leftarrow \text{LINEARMESSAGES-MST}(G)$
- 3:   **return**  $\mathcal{T}$
- 4: **else**
- 5:    $v^*$  generates a sequence  $\pi$  of  $\Theta(\log^2 n)$  bits independently and uniformly at random and shares with all nodes in  $V$
- 6:    $p \leftarrow 1/n^\varepsilon$
- 7:   Each node constructs an  $\Theta(\log n)$ -wise-independent sampler from  $\pi$  and uses this to sample each incident edge in  $G$  with probability  $p$
- 8:    $H \leftarrow$  the spanning subgraph of  $G$  induced by the sampled edges
- 9:    $F \leftarrow \text{MST-V2}(H)$
- 10:    $E_\ell \leftarrow \text{COMPUTE-F-LIGHT}(G, F, p)$
- 11:    $\mathcal{T} \leftarrow \text{LINEARMESSAGES-MST}((V, E_\ell, w))$
- 12:   **return**  $\mathcal{T}$

---

► **Theorem 2.3.** *Algorithm MST-v2 outputs an MST of an edge-weighted  $n$ -node,  $m$ -edge graph when terminates. Moreover, for any  $\varepsilon > 0$ , it terminates after  $O(\log^* n/\varepsilon)$  rounds and uses  $\tilde{O}(n^{1+\varepsilon}/\varepsilon)$  messages, w.h.p.*

**Proof.** If  $m = O(n^{1+\varepsilon})$  then the claim follows from Theorem 1.3. Let  $T(m)$  denote the time required for Algorithm 2 to compute an MST of a  $n$ -node,  $m$ -edge graph. Since COMPUTE-F-LIGHT( $\cdot$ ) runs in  $O(1)$  time and LINEARMESSAGES-MST( $\cdot$ ) runs in  $O(\log^* n)$  time, we see that,  $T(m) = T(m/n^\varepsilon) + O(\log^* n)$ , for all large  $m$ . The first quantity is the result of a recursive call on the sampled graph  $H$ , where each edge is sampled with probability  $p = 1/n^\varepsilon$ . Solving this recursion with base case  $m = O(n^{1+\varepsilon})$ , we get  $T(m) = O(\log^* n/\varepsilon)$ . The message complexity bound is obtained by similar arguments. ◀

Setting  $\varepsilon = \log \log n / \log n$ , we get the following result.

► **Corollary 2.4.** *There exists an algorithm that computes an MST of an  $n$ -node,  $m$ -edge input graph and w.h.p. terminates in  $O(\log n \cdot \log^* n / \log \log n)$  rounds and  $\tilde{O}(n)$  messages.*

### 3 Efficient Computation of $F$ -light Edges

In this section we describe the COMPUTE-F-LIGHT algorithm and prove its correctness and analyze its time and message complexity. The inputs to this algorithm are the graph  $G$ , a spanning forest  $F$  of  $G$ , and a probability  $p$ . Recall that  $F$  is the maximal minimum weight spanning forest of the subgraph  $H$  obtained by sampling edges in  $G$  with probability  $p$ , using a  $\Theta(\log n)$ -wise-independent sampler. The main ideas in COMPUTE-F-LIGHT have been informally described in Section 1.2. The COMPUTE-F-LIGHT algorithm is described below in Algorithm 3.

**Algorithm 3** COMPUTE-F-LIGHT

**Input:** (i) An edge-weighted  $n$ -node,  $m$ -edge graph  $G = (V, E, w)$ , (ii) A spanning forest  $F$  of  $G$ , and (iii) a number  $p$ ,  $0 < p < 1$ .

▷  $F$  is a maximal minimum weight spanning forest of a subgraph  $H$  of  $G$ , where  $H$  is a spanning subgraph of  $G$  obtained by sampling each edge in  $G$  with probability  $p$  using a  $\Theta(\log n)$ -wise-independent sampler. Each node knows weights and end-points of incident edges from  $G$  and  $F$ . Every weight can be represented using  $O(\log n)$  bits.

**Output:**  $F$ -light edges of  $G$ .

▷ Each node in  $V$  knows which of its incident edges from  $G$  are  $F$ -light.

- 
- 1: Let  $\{v_1, v_2, \dots, v_c\}$  be set of *commander nodes* (or in short, *commanders*) where  $c = \Theta(\log n)$ . Gather  $F$  at each of these commanders.
  - 2: Each commander simulates Borůvka's algorithm locally on input graph  $F$ . Let  $\mathcal{C}^i = \{C_1^i, C_2^i, \dots\}$  be the set of components at the beginning of Phase  $i$ . The node with smallest ID in a component  $C_j^i$  is the *leader* of component  $C_j^i$  and the ID of the leader serves as the label of each component. For each component  $C_j^i \in \mathcal{C}^i$ , the algorithm picks a MWOE  $e_j^i$  from  $F$ . Components are merged and we get a new set of components  $\mathcal{C}^{i+1}$ . If there is no incident edge on a component  $C_j^i$  in  $F$  then commander sets  $e_j^i = \perp$  with the understanding that  $w(\perp) = \infty$ .
  - 3: For each component  $C_j^i$ , commander  $v_i$  sends the following 3-tuple to each node in  $C_j^i$ :  
(a) Phase number  $i$ , (b) label of  $C_j^i$ , and (c)  $w(e_j^i)$ .
  - 4: A node  $v$  having received a 3-tuple  $(i, \ell, w')$  associated with component  $C_j^i$  for some  $i$  and  $j$  computes  $\Theta\left(\frac{\log^5 n}{p}\right)$  different graph sketches with respect to its  $w'$ -restricted neighborhood  $N_{w'}(v)$ .
  - 5: The component leader of  $C_j^i$  for each  $i$  and  $j$ , gathers  $\Theta\left(\frac{\log^5 n}{p}\right)$   $w(e_j^i)$ -restricted sketches from all the nodes in  $C_j^i$  and computes  $w(e_j^i)$ -restricted sketches of  $C_j^i$ . Then it samples an edge from each sketch computed and notifies the end-points of all sampled edges.
  - 6: **return** Union of sampled edges over all  $i$  over all  $j$ .
- 

**3.1 Analysis**

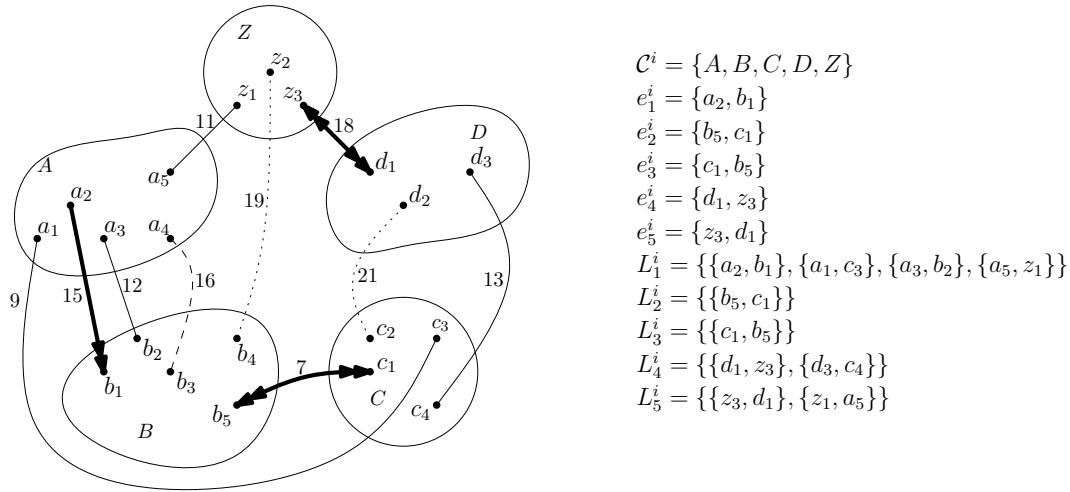
Let  $\mathcal{C}^i = \{C_1^i, C_2^i, \dots\}$  be the set of components at the beginning of Phase  $i$  of Borůvka's algorithm being locally simulated on  $F$ . Consider the set of edges from  $G$  with exactly one endpoint in  $C_j^i$  with weight at most  $w(e_j^i)$ :  $L_j^i = \{e = \{u, v\} \in E \mid u \in C_j^i, v \notin C_j^i \text{ and } w(e) \leq w(e_j^i)\}$ . For example, see Figure 1.

Our first task is to bound the size of  $L_j^i$  and for this we appeal to the following lemma from Pettie and Ramachandran [31] on sampling from an ordered set.

► **Lemma 3.1** (Pettie & Ramachandran [31]). *Let  $\chi$  be a set of  $n$  totally ordered elements and  $\chi_p$  be a subset of  $\chi$ , derived by sampling each element with probability  $p$  using a  $k$ -wise-independent sampler. Let  $Z$  be the number of unsampled elements less than the smallest element in  $\chi_p$ . Then  $\mathbf{E}[Z] \leq p^{-1}(8(\pi/e)^2 + 1)$  for  $k \geq 4$ .*

Observe that a straight-forward application of the above lemma gives us  $\mathbf{E}[|L_j^i|] = O(1/p)$ . In the next lemma, we modify the proof of Lemma 3.1 in Pettie & Ramachandran [31] to obtain a bound on size of  $L_j^i$  that holds w.h.p.

► **Lemma 3.2.** *Pr (There exist  $i$  and  $j$ :  $|L_j^i| > c \cdot \log^3 n/p$ )  $< \frac{1}{n}$  for some constant  $c > 1$ .*



■ **Figure 1** Illustration of notation and terminology used in Algorithm 3 COMPUTE-F-LIGHT. At the beginning of Phase  $i$  of Borůvka's algorithm, there are 5 components  $\{A, B, C, D, Z\}$ . Each component's MWOE in  $F$  is shown as thick directed arc. Solid arcs show edges in  $G$  that are in respective  $L_j^i$ 's and hence identified as being  $F$ -light. Dashed arcs (e.g.,  $a_4b_3$ ) represent edges that the algorithm ignores; these edge are not  $F$ -light. Dotted arcs (e.g.,  $b_4z_2, c_2d_2$ ) represent edges in  $G$  whose status has not yet been resolved by the algorithm. After the merging of components is completed, we end up with two components  $\{ABC, DZ\}$ .

**Proof.** Fix a Phase  $i$  and a component  $C_j^i$  in that phase. Let  $X$  be the set of all edges from  $G$  having exactly one endpoint in  $C_j^i$ . Let  $X_t$  be an indicator random variable defined as  $X_t = 1$  if the  $t^{\text{th}}$  smallest edge in  $X$  is sampled, and 0 otherwise. For any integer  $\ell$ ,  $1 \leq \ell \leq |X|$ , let  $S_\ell = \sum_{t=1}^{\ell} X_t$  count the number of ones in  $X_1, \dots, X_\ell$ . Note that  $L_j^i \subseteq X$  is a set of all edges with weight at most  $e_j^i$ , the MWOE from  $C_j^i$  in  $F$ . This implies that the lightest edge in  $X$  that is sampled is  $e_j^i$ , otherwise Borůvka's algorithm would have chosen a different MWOE. In other words,  $X_k = 0$  for all  $k \leq \ell$  if the rank of  $e_j^i$  in the ordered set  $X$  is  $\ell + 1$  or more. Therefore,  $\Pr(|L_j^i| > \ell) = \Pr(S_\ell = 0)$ .

Observe that,  $S_\ell$  is a sum of 0-1 random variables which are  $\Theta(\log n)$ -wise-independent and  $\mathbf{E}[S_\ell] = p\ell$ . By Theorem 1.2, we have  $\Pr(S_\ell = 0) < \frac{1}{n^3}$  for  $\ell > c \cdot \log^3 n/p$  for some constant  $c > 1$ . The lemma follows by applying union bound over all phases and components. ◀

► **Lemma 3.3.** For any Phase  $i$  and any component-MWOE pair  $(C_j^i, e_j^i)$ , w.h.p.  $O(\log^5 n/p)$   $w(e_j^i)$ -restricted sketches of  $C_j^i$  are sufficient to find all edges in  $L_j^i$ .

**Proof.** Consider an oracle which when queried returns an edge in  $L_j^i$  independently and uniformly at random. Let  $T_s$  denote the number of the oracle queries required to obtain  $s = |L_j^i|$  distinct edges (i.e., all edges in  $L_j^i$ ). Then by the Coupon Collector argument [25],  $\Pr(T_s > \beta s \log s) < s^{-\beta+1}$  for any  $\beta > 1$ . Also, if the oracle is not uniform, but is "almost uniform," returning an edge in  $L_j^i$  with probability  $\frac{1}{s} \pm s^{-\alpha}$  for a constant  $\alpha > 2$ , then we get  $\Pr(T_s > \beta s \log s + o(1)) < s^{-\beta+1}$ .

Now, to simulate a  $t^{\text{th}}$  oracle query ( $t \in [1, T_s]$ ) mentioned above, we sample an unused sketch of  $C_j^i$  until we get an edge. Since sampling from a sketch fails with probability at most  $n^{-2}$ , w.h.p.,  $O(1)$  sketches are sufficient to simulate one oracle query. Hence w.h.p.,  $O(T_s)$  sketches are sufficient to simulate  $T_s$  oracle queries. Therefore, with probability at least  $1 - s^{-\beta+1}$ ,  $O(\beta s \log s)$  sketches are sufficient to get  $s$  distinct edges from  $L_j^i$ .



By Lemma 3.2, we have w.h.p.,  $s = |L_j^i| = O(\log^3 n/p)$ . Therefore by letting  $s = \Theta(\log^3 n/p)$  and  $\beta = O(\log n)$  in the above argument, w.h.p.,  $O(\log^5 n/p)$  sketches are sufficient to find all edges in  $L_j^i$ .  $\blacktriangleleft$

► **Lemma 3.4.** *Let  $E_\ell$  be the set of  $F$ -light edges in  $G$ . Let  $L = \cup_i \cup_j L_j^i$ . Then,  $E_\ell = L$ .*

**Proof.** We first show that  $L \subseteq E_\ell$ . Consider a Phase  $i$  and a component-MWOE pair  $(C_j^i, e_j^i)$ . Consider any edge  $e = \{u, v\} \in L_j^i$  with  $u \in C_j^i, v \notin C_j^i$ . Since  $e_j^i$  is the MWOE from  $C_j^i$  and  $u \in C_j^i$ , any path in  $F$  connecting  $u$  to any node  $x \notin C_j^i$  has to go through edge  $e_j^i$ . Therefore, for any  $x \notin C_j^i, w_F(u, x) \geq w(e_j^i)$ . Since  $v \notin C_j^i$  we have  $w_F(u, v) \geq w(e_j^i)$ . Moreover, since  $e \in L_j^i$ , we have  $w(e) \leq w(e_j^i)$  implies  $w(e) \leq w_F(u, v)$ . Hence,  $e$  is  $F$ -light. Since this is true for any  $e \in L_j^i$ , we have  $L_j^i \subseteq E_\ell$ . Hence,  $L \subseteq E_\ell$ .

Now, we show that  $E_\ell \subseteq L$ . For any node  $u \in V$ , let  $C^q(u)$  denote the component containing  $u$  just before Phase  $q$  of Borůvka's algorithm (Step 2 in Algorithm COMPUTE-F-LIGHT). For the sake of contradiction, let there be an edge  $e = \{u, v\} \in E_\ell \setminus L$ . Let  $i$  be the index of the phase in which component of  $u$  and component of  $v$  is merged together<sup>4</sup> (that is, for any  $q < i + 1, C^q(u) \neq C^q(v)$  and  $C^{i+1}(u) = C^{i+1}(v)$ ). Consider the path  $F(u, v)$  and note that since  $C^{i+1}(u) = C^{i+1}(v)$ , the entire path  $F(u, v)$  is in  $C^{i+1}(u)$ . Now consider the Phase  $i$  components  $C_1^i, \dots, C_t^i, t \geq 2$  along this path  $F(u, v)$  (see Figure 2). WLOG, let  $u \in C_1^i$  and  $v \in C_t^i$  and suppose that the path  $F(u, v)$  visits the components in the order  $u \in C_1^i, C_2^i, \dots, C_{t-1}^i, v \in C_t^i$ . For example, in Figure 2 the path  $F(u, v)$  starts in  $C_1^i$  then goes through  $C_2^i$ , then to  $C_3^i$ , and finally to  $C_4^i$ . Let  $F'(u, v)$  denote the subset of edges in  $F(u, v)$  that have endpoints in two distinct Phase  $i$  components.

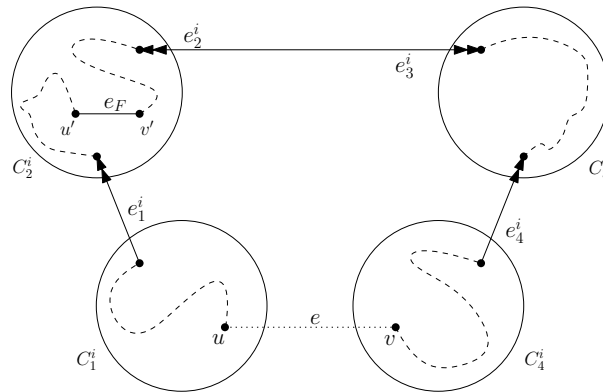
Now consider the MWOE's of these components:  $e_j^i$  is the MWOE for  $C_j^i$  for  $j = 1, 2, \dots, t$ . There are three cases depending on how the MWOEs  $e_j^i$  relate to the path  $F(u, v)$ .

- $e_j^i$  connects  $C_j^i$  to  $C_{j+1}^i$  for  $j = 1, 2, \dots, t - 1$ . Since  $e$  has exactly one endpoint in  $C_1^i$  and  $e \notin L_1^i$  (since  $e \notin L$ ), we have  $w(e) > w(e_1^i)$ . Furthermore, due to the structure of the MWOEs:  $w(e_1^i) > w(e_2^i) > \dots > w(e_{t-1}^i)$ . This implies that  $w(e)$  is larger than the weights of all edges in  $F'(u, v)$ .
- $e_j^i$  connects  $C_j^i$  to  $C_{j-1}^i$  for  $j = 2, \dots, t$ . Since  $e$  has exactly one endpoint in  $C_t^i$  and  $e \notin L_t^i$  (since  $e \notin L$ ), we have  $w(e) > w(e_t^i)$ . Furthermore, due to the structure of the MWOEs:  $w(e_t^i) > w(e_{t-1}^i) > \dots > w(e_2^i)$ . This implies that  $w(e)$  is larger than the weights of all edges in  $F'(u, v)$ .
- There is some  $\ell, 1 \leq \ell < t$  such that  $e_j^i$  connects  $C_j^i$  to  $C_{j+1}^i$  for  $j = 1, 2, \dots, \ell$  and  $e_j^i$  connects  $C_j^i$  to  $C_{j-1}^i$  for  $j = \ell + 1, \dots, t$ . This case is illustrated in Figure 2 with  $\ell = 2$ . In this case,  $w(e) > w(e_1^i)$  and  $w(e) > w(e_t^i)$  for reasons mentioned in the previous two cases. Furthermore, due to the structure of the MWOEs:  $w(e_1^i) > w(e_2^i) > \dots > w(e_\ell^i)$  and  $w(e_t^i) > w(e_{t-1}^i) > \dots > w(e_{\ell+1}^i)$ . This implies that  $w(e)$  is larger than the weights of all edges in  $F'(u, v)$ .

Thus in all three cases,  $w(e)$  is larger than the weights of all edges in  $F'(u, v)$ . Now let  $e_F = \{u', v'\} \in F$  be the maximum weight edge in  $F(u, v)$ . Since  $e$  is  $F$ -light, we have  $w(e) < w(e_F)$ . This inequality combined with the fact that  $w(e)$  is larger than the weights of all edges in  $F'(u, v)$  implies that  $u'$  and  $v'$  belong to the same Phase  $i$  component, i.e.,  $C^i(u') = C^i(v')$ . For example, in Figure 2,  $u'$  and  $v'$  are in  $C_2^i$ .

Let  $C^i(u') = C^i(v') = C_\ell^i$  for some  $\ell \leq t$ . Let  $F(u, v) = F(u, u') \cup \{u', v'\} \cup F(v', v)$ . Since  $e_F$  is the heaviest edge in  $F(u, v)$ , all the edges in  $F(u, u')$  are lighter than  $e_F$ . Hence

<sup>4</sup> If  $u$  and  $v$  are never merged into one component, i.e., they are in different components in  $F$  then  $\{u, v\} \in L_j^i$  where  $i$  is the phase in which  $u$ 's component becomes maximal with respect to  $F$  and  $j$  is such that  $u$  belongs to  $C_j^i$ . This follows from the fact that  $e_j^i = \perp$  and  $w(e_j^i) = \infty$ .



■ **Figure 2** Illustration of proof of Lemma 3.4. After Phase  $i$ , components  $C_1^i, C_2^i, C_3^i, C_4^i$  are merged together using edges  $e_1^i, e_2^i, e_3^i, e_4^i$  in  $F$ . Dashed curves represent paths in  $F$  between the respective end-points.  $e$  is an  $F$ -light edge.  $e_F$  is the heaviest edge on path from  $u$  to  $v$  in  $F$ .

■ **Table 2** Time and message complexity for steps in Algorithm 3 COMPUTE-F-LIGHT.

Step	Time	Messages	Analysis
1	$O(1)$	$\tilde{O}(n)$	Full paper
2	–	–	Local computation
3	$O(1)$	$\tilde{O}(n)$	Trivial direct communication
4	$O(1)$	$\tilde{O}(n/p)$	Theorem 1.1
5	$O(1)$	$\tilde{O}(n/p)$	Full paper

at any Phase  $i' < i$ , Borůvka's algorithm considers edges in  $F(u, u')$  for component  $C^{i'}(u')$  and edges in  $F(v', v)$  for component  $C^{i'}(v')$  before considering  $e_F$ . The implication of this is,  $C^i(u) = C^i(u')$  and  $C^i(v) = C^i(v')$ . But,  $C^i(u) \neq C^i(v)$  therefore,  $C^i(u') \neq C^i(v')$  – a contradiction. ◀

From Lemma 3.2 and Lemma 3.4 we get the following bound on the number of  $F$ -light edges in  $G$ .

▶ **Corollary 3.5.** *W.h.p., the number of  $F$ -light edges in  $G$  is  $\tilde{O}(n/p)$ .*

Table 2 summarizes the time and message complexity of each step of Algorithm COMPUTE-F-LIGHT. A naive implementation of Step 5 may require super-constant number of rounds because of receiver-side bottlenecks, but a more sophisticated implementation that appears in the full version of the paper [30] shows how to implement this step in  $O(1)$  rounds, using  $\tilde{O}(n/p)$  messages.

From Lemma 3.4 and Table 2 we get the following result.

▶ **Theorem 3.6.** *Algorithm COMPUTE-F-LIGHT computes all  $F$ -light edges for given graph  $G$  and a minimum spanning forest  $F$  of  $H$  where  $H$  is obtained by sampling each edge in  $G$  with probability  $p$  using a  $\Theta(\log n)$ -wise-independent sampler. Moreover, the computation takes  $O(1)$  rounds and uses  $\tilde{O}(n/p)$  messages w.h.p.*

## References

- 1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467, 2012.

- 2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st Symposium on Principles of Database Systems (PODS)*, pages 5–14, 2012.
- 3 Andrew Berns, James Hegeman, and Sriram V. Pemmaraju. Super-Fast Distributed Algorithms for Metric Facility Location. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 428–439, 2012.
- 4 Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21-23, 2015*, pages 143–152. ACM, 2015. doi:10.1145/2767386.2767414.
- 5 Graham Cormode and Donatella Firmani. A unifying framework for  $\ell_0$ -sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014.
- 6 Danny Dolev, Christoph Lenzen, and Shir Peled. “Tri, Tri Again”: Finding Triangles and Small Subgraphs in a Distributed Setting. In *Proceedings of the 26th International Symposium on Distributed Computing (DISC)*, pages 195–209, 2012.
- 7 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. The communication complexity of distributed task allocation. In *Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 67–76, 2012. doi:10.1145/2332432.2332443.
- 8 Michael Elkin. An Unconditional Lower Bound on the Time-Approximation Trade-off for the Distributed Minimum Spanning Tree Problem. *SIAM J. Comput.*, 36(2):433–456, 2006. doi:10.1137/S0097539704441058.
- 9 Joachim Gehweiler, Christiane Lammersen, and Christian Sohler. A Distributed  $O(1)$ -approximation Algorithm for the Uniform Facility Location Problem. In *Proceedings of the 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 237–243, 2006. doi:10.1145/1148109.1148152.
- 10 Mohsen Ghaffari and Merav Parter. MST in Log-Star Rounds of Congested Clique. In *Proc. of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC’16, 2016.
- 11 James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward Optimal Bounds in the Congested Clique: Graph Connectivity and MST. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC’15, pages 91–100. ACM, 2015. doi:10.1145/2767386.2767434.
- 12 James W. Hegeman and Sriram V. Pemmaraju. Lessons from the Congested Clique Applied to MapReduce. In *Proceedings of the 21th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 149–164, 2014. doi:10.1007/978-3-319-09620-9\_13.
- 13 James W. Hegeman, Sriram V. Pemmaraju, and Vivek B. Sardeshmukh. Near-Constant-Time Distributed Algorithms on a Congested Clique. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, pages 514–530, 2014.
- 14 Stephan Holzer and Nathan Pinsker. Approximation of Distances and Shortest Paths in the Broadcast Congest Clique. *CoRR*, abs/1412.3445, 2014.
- 15 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight Bounds for  $L_p$  Samplers, Finding Duplicates in Streams, and Related Problems. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS’11, pages 49–58. ACM, 2011. doi:10.1145/1989284.1989289.
- 16 David R. Karger, Philip N. Klein, and Robert E. Tarjan. A Randomized Linear-time Algorithm to Find Minimum Spanning Trees. *J. ACM*, 42(2):321–328, March 1995. doi:10.1145/201019.201022.
- 17 Valerie King, Shay Kutten, and Mikkel Thorup. Construction and impromptu repair of an mst in a distributed network with  $o(m)$  communication. In *Proceedings of the 2015 ACM*

- Symposium on Principles of Distributed Computing*, PODC'15, pages 71–80, New York, NY, USA, 2015. ACM. doi:10.1145/2767386.2767405.
- 18 Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed Computation of Large-Scale Graph Problems. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 391–410, 2015.
  - 19 Janne H. Korhonen. Deterministic MST sparsification in the congested clique. *CoRR*, abs/1605.02022, 2016. URL: <http://arxiv.org/abs/1605.02022>.
  - 20 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the complexity of universal leader election. *J. ACM*, 62(1):7:1–7:27, March 2015. doi:10.1145/2699440.
  - 21 Shay Kutten and David Peleg. Fast Distributed Construction of Small  $k$ -Dominating Sets and Applications. *J. Algorithms*, 28(1):40–66, 1998. doi:10.1006/jagm.1998.0929.
  - 22 Christoph Lenzen. Optimal Deterministic Routing and Sorting on the Congested Clique. In *Proceedings of the 31st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 42–50. ACM, 2013. doi:10.1145/2484239.2501983.
  - 23 Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-Weight Spanning Tree Construction in  $O(\log \log n)$  Communication Rounds. *SIAM Journal on Computing*, 35(1):120–131, 2005.
  - 24 Andrew McGregor. Graph stream algorithms: A survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.
  - 25 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.
  - 26 Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. of the 46th ACM Symp. on Theory of Computing (STOC)*, pages 565–573, 2014.
  - 27 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. A time- and message-optimal distributed algorithm for minimum spanning trees. *CoRR*, abs/1607.06883, 2016. URL: <http://arxiv.org/abs/1607.06883>.
  - 28 Boaz Patt-Shamir and Marat Teplitsky. The Round Complexity of Distributed Sorting. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 249–256, 2011. doi:10.1145/1993806.1993851.
  - 29 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial Mathematics, 2000.
  - 30 Sriram V. Pemmaraju and Vivek B. Sardeshmukh. Super-fast MST algorithms in the congested clique using  $o(m)$  messages. *CoRR*, abs/1610.03897, 2016. URL: <http://arxiv.org/abs/1610.03897>.
  - 31 Seth Pettie and Vijaya Ramachandran. Randomized minimum spanning tree algorithms using exponentially fewer random bits. *ACM Trans. Algorithms*, 4(1), 2008. doi:10.1145/1328911.1328916.
  - 32 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed Verification and Hardness of Distributed Approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.
  - 33 Jeanette P. Schmidt, Alan Siegel, and Srinivasan Aravind. Chernoff-Hoeffding Bounds for Applications with Limited Independence. *SIAM J. Discrete Math.*, 8(2):223–250, 1995. doi:10.1137/S089548019223872X.
  - 34 Robert Endre Tarjan. *CBMS-NSF Regional Conference Series in Applied Mathematics: Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, New York, NY, USA, 1983.



# Why Liveness for Timed Automata Is Hard, and What We Can Do About It\*

Frédéric Herbreteau<sup>1</sup>, B. Srivathsan<sup>2</sup>, Thanh-Tung Tran<sup>3</sup>, and Igor Walukiewicz<sup>4</sup>

<sup>1</sup> Université de Bordeaux, Bordeaux INP, CNRS, LaBRI UMR5800, France

<sup>2</sup> Chennai Mathematical Institute, India

<sup>3</sup> Université de Bordeaux, Bordeaux INP, CNRS, LaBRI UMR5800, France

<sup>4</sup> Université de Bordeaux, Bordeaux INP, CNRS, LaBRI UMR5800, France

---

## Abstract

The liveness problem for timed automata asks if a given automaton has a run passing infinitely often through an accepting state. We show that unless  $P=NP$ , the liveness problem is more difficult than the reachability problem; more precisely, we exhibit a family of automata for which solving the reachability problem with the standard algorithm is in  $P$  but solving the liveness problem is  $NP$ -hard. This leads us to revisit the algorithmics for the liveness problem. We propose a notion of a witness for the fact that a timed automaton violates a liveness property. We give an algorithm for computing such a witness and compare it with the existing solutions.

**1998 ACM Subject Classification** F.3.1 Specifying and Verifying and Reasoning about Programs

**Keywords and phrases** Timed automata, model-checking, liveness invariant, state subsumption

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.48

## 1 Introduction

Timed automata [1] are one of the standard models of timed systems. There has been an extensive body of work on the verification of reachability/safety properties of timed automata. In contrast, advances on verification of liveness properties are much less spectacular. For verification of liveness properties expressed in a logic like Linear Temporal Logic, it is best to consider a slightly more general problem of verification of Büchi properties. This means verifying if in a given timed automaton there is an infinite path passing through an accepting state infinitely often.

Testing Büchi properties of timed systems can be surprisingly useful. We give an example in Section 6 where we describe how with a simple liveness test one can discover a typo in the benchmark CSMA/CD model. This typo removes practically all the interesting behaviors from the model. Yet the CSMA/CD benchmark has been extensively used for evaluating verification tools, and nothing unusual has been observed. Therefore, even if one is interested solely in verification of safety properties, it is important to “test” the model under consideration, and for this Büchi properties are indispensable.

Verification of reachability properties of timed automata is possible in practice thanks to zones and their abstractions [4, 3, 9]. Roughly, the standard approach used nowadays for safety properties performs a breadth first search (BFS) over the set of pairs (state, zone)

---

\* This work has been supported by project AVerTS – CEFIPRA – Indo-French Program in ICST – DST/CNRS ref. 218093. Author B. Srivathsan is partially funded by a grant from Infosys Foundation.



reachable in the automaton, storing only pairs with the maximal abstracted zones (with respect to inclusion). In jargon: the algorithm constructs a zone graph with subsumption.

In this paper we give a strong evidence that verification of Büchi properties is inherently more difficult than verification of reachability properties. For a long time it has been understood that for liveness, there is a problem with the approach outlined above as it is no longer sound to keep only maximal zones with respect to inclusion (i.e. to use subsumption) [11, 13]. It is possible to use the zone graph without subsumption, but this one is almost always too big to handle. One could hope though that some modification of the notion of zone graph with subsumption can give an algorithm for Büchi properties that is provably not much more costly than that for safety properties. We show that this is impossible. We present a family of examples where reachability is much easier to decide than verification of Büchi properties. This proves that unless  $P=NP$ , there is no hope to obtain an algorithm for Büchi properties that has provably similar complexity to the standard reachability algorithm (which constructs zone graph with subsumption).

Our goal in this paper is to rethink the foundations of verification of Büchi properties for timed automata, and propose some algorithmic solutions. The first question we address is this: what can be a witness to the fact that an automaton has no Büchi accepting run? As we have mentioned above, for safety properties such a witness is a zone graph with subsumption. We propose a similar notion of a witness for Büchi properties that allows only “safe” subsumptions. As the next contribution, we give an algorithm for computing such a witness. Due to the hardness result mentioned above, we cannot hope to have as efficient an algorithm as for reachability. We propose an algorithm that will iteratively apply the reachability algorithm. It will first construct the zone graph with subsumption, stopping if it finds a Büchi run. If all subsumptions in this graph are safe according to our definition then this graph forms a witness for non-existence of a Büchi run. Otherwise the algorithm recursively refines strongly connected components of the zone graph with unsafe subsumptions. This algorithm computes the zone graph without subsumption in the worst case - this as we show is anyway necessary in some cases. The expected advantage is that in many cases our algorithm can stop sooner. We have implemented our algorithm and run it on a set of benchmarks from [11]. On these examples indeed the algorithm mostly stops after the first iteration, and constructs witnesses of size very close to those for safety. To complete the picture we also give a set of particularly hard examples for our algorithm.

**Related work:** Verification of liveness properties is decidable thanks to the region construction [1]. The use of zones and (certain) abstractions for this problem was developed in [13]. Later Li [12] has shown that existence of a Büchi run is preserved by every abstraction based on simulation. In particular, this is the case for the  $\alpha_{\leq LU}$  abstraction [3] that is the coarsest abstraction depending only on lower and upper bounds in clock guards (LU-bounds) [7]. Thanks to these results the liveness checking can be done on an abstract zone graph using  $\alpha_{\leq LU}$  abstraction (but without subsumption). The question of whether subsumption can be used to improve the liveness verification was raised in [13]. Laarman et al. [11] recently proposed a nested DFS based algorithm for checking Büchi properties of timed automata. They study in depth when it is sound to use subsumption in the nested dfs algorithm. Our conditions on the use of subsumption are expressed in terms of zone graphs and are independent of a particular algorithm. This allows us to focus on the task of finding a witness graph efficiently, in particular we can use BFS based algorithms for the task. We give a more detailed comparison of the two algorithms in Section 6.



**Organization of the paper:** In the next section we present the basic definitions, as well as the algorithms for constructing the abstract zone graph, and the abstract zone graph with subsumption. We also describe the nested DFS algorithm from [11]. In Section 3 we give our notion of a witness for non-existence of a Büchi run in a given automaton. Section 4 presents a theorem implying the above stated algorithmic difference between verification of liveness and reachability properties. In Section 5 we propose an algorithm for finding such witnesses and prove its correctness. Section 6 reports some experimental results.

## 2 Preliminaries

In this section we present the basic definitions. In particular, we define abstract zone graphs, and the use of subsumption. We also present the standard algorithm for constructing an abstract zone graph with subsumption. This can be used to answer reachability properties. We finish this section with the nested DFS algorithm for liveness properties from [11].

Let  $\mathbb{R}_{\geq 0}$  denote the set of non-negative reals. A *clock* is a variable that ranges over  $\mathbb{R}_{\geq 0}$ . Let  $X = \{x_1, \dots, x_n\}$  be a set of clocks. A *valuation* is a function  $v : X \rightarrow \mathbb{R}_{\geq 0}$ . The set of all clock valuations is denoted by  $\mathbb{R}_{\geq 0}^X$ . We denote by  $\mathbf{0}$  the valuation that associates 0 to every clock in  $X$ . A *clock constraint*  $\phi$  is a conjunction of constraints of the form  $x \sim c$  where  $x \in X$ ,  $\sim \in \{<, \leq, =, \geq, >\}$  and  $c \in \mathbb{N}$ . Let  $\Phi(X)$  denote the set of clock constraints over the set of clocks  $X$ . A valuation  $v$  is said to satisfy a constraint  $\phi$ , written as  $v \models \phi$ , when every constraint in  $\phi$  holds after replacing every  $x$  by  $v(x)$ . For  $\delta \in \mathbb{R}_{\geq 0}$ , let  $v + \delta$  be the valuation that associates  $v(x) + \delta$  to every clock  $x$ . For  $R \subseteq X$ , let  $[R]v$  be the valuation that sets  $x$  to 0 if  $x \in R$ , and that sets  $x$  to  $v(x)$  otherwise.

► **Definition 1** (Timed Büchi Automata [1]). A *Timed Büchi Automaton (TBA in short)* is a tuple  $\mathcal{A} = (Q, q_0, X, T, F)$  in which  $Q$  is a finite set of states,  $q_0$  is the initial state,  $X$  is a finite set of clocks,  $F \subseteq Q$  is a set of accepting states, and  $T \subseteq Q \times \Phi(X) \times 2^X \times Q$  is a finite set of transitions of the form  $(q, g, R, q')$  where  $g$  is a clock constraint called the *guard*, and  $R$  is a set of clocks that are *reset* on the transition from  $q$  to  $q'$ .

The semantics of a TBA  $\mathcal{A} = (Q, q_0, X, T, F)$  is given by a transition system of its configurations. A *configuration* of  $\mathcal{A}$  is a pair  $(q, v) \in Q \times \mathbb{R}_{\geq 0}^X$ , with  $(q_0, \mathbf{0})$  being the initial configuration. There are two kinds of transitions:

- **delay:**  $(q, v) \rightarrow^\delta (q, v + \delta)$  for  $\delta \in \mathbb{R}_{\geq 0}$ ;
- **action:**  $(q, v) \rightarrow^t (q', v')$  for  $t = (q, g, R, q') \in T$  such that  $v \models g$  and  $v' = [R]v$ .

A *run* of  $\mathcal{A}$  is a (finite or infinite) sequence of transitions starting from the initial configuration:  $(q_0, \mathbf{0}) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \dots$ , where  $(q, v) \xrightarrow{\delta, t} (q', v')$  denotes a delay  $\delta$  followed by action  $t$  starting from  $(q, v + \delta)$ . A configuration  $(q, v)$  is said to be *accepting* if  $q \in F$ . An infinite run *satisfies the Büchi condition* if it visits accepting configurations infinitely often. The run is *Zeno* if its accumulated duration is finite, i.e.,  $\sum_{i \geq 0} \delta_i \leq c$  for some  $c \in \mathbb{R}_{\geq 0}$ . Else it is *non-Zeno*. The problem we are interested is termed the *Büchi non-emptiness problem*.

► **Definition 2.** The *Büchi non-emptiness problem* for TBA is to decide if a given TBA  $\mathcal{A}$  has a non-Zeno run satisfying the Büchi condition.

The Büchi non-emptiness problem is known to be PSPACE-complete [1]. Standard solutions to this problem construct an untimed Büchi automaton and check for its emptiness. There are various methods to handle the non-Zeno requirement [15, 8]. In this paper, we will assume that the automata are strongly non-Zeno [13], that is, every infinite accepting run is non-Zeno. The strongly non-Zeno construction could lead to an exponential blowup [8, 6] to

the abstract zone graph (which is defined below), but we prefer to employ this commonly used assumption in order not to divert from the main subject. We will now describe a translation which reduces the Büchi non-emptiness problem to checking non-emptiness of an untimed Büchi automaton.

**Abstract zone graphs:** As the semantics of a TBA is an infinite transition system, algorithms for TBA consider special sets of valuations called *zones*. A zone is a set of valuations described by a conjunction of two kinds of constraints: either  $x_i \sim c$  or  $x_i - x_j \sim c$  where  $x_i, x_j \in X$ ,  $c \in \mathbb{Z}$  and  $\sim \in \{<, \leq, =, >, \geq\}$ . For example  $(x_1 > 3 \wedge x_2 - x_1 \leq -4)$  is a zone. Zones can be efficiently represented by Difference Bound Matrices (DBMs) [5].

The *zone graph*  $ZG(\mathcal{A})$  has as nodes pairs  $(q, Z)$  consisting of a state of the TBA and a zone. The initial node is  $(q_0, Z_0)$  where  $Z_0 = \{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$ . For every  $t = (g, R, q') \in T$ , and every set of valuations  $W$ , we define the transition  $\Rightarrow^t$  as:  $(q, W) \Rightarrow^t (q', W')$  where  $W' = \{v' \mid \exists v \in W, \exists \delta \in \mathbb{R}_{\geq 0} : (q, v) \xrightarrow{t, \delta} (q', v')\}$ . If  $W$  is a zone, then so is  $W'$ . In the zone graph, from every node  $(q, Z)$  there is a transition  $(q, Z) \Rightarrow^t (q', Z')$  corresponding to the transitions  $t$  from  $q$ . The transition relation  $\Rightarrow$  is the union of  $\Rightarrow^t$  over all  $t \in T$ .

Although the zone graph  $ZG(\mathcal{A})$  groups together valuations, the number of zones is still infinite [4]. For effectiveness, zones are further abstracted. An *abstraction operator* is a function  $\mathbf{a} : \mathcal{P}(\mathbb{R}_{\geq 0}^{|X|}) \rightarrow \mathcal{P}(\mathbb{R}_{\geq 0}^{|X|})$  such that  $W \subseteq \mathbf{a}(W)$  and  $\mathbf{a}(\mathbf{a}(W)) = \mathbf{a}(W)$  for every set of valuations  $W \in \mathcal{P}(\mathbb{R}_{\geq 0}^{|X|})$ . The abstraction is *finite* if  $\mathbf{a}$  has a finite range. An abstraction operator defines an abstract symbolic semantics:  $(q, W) \Rightarrow_{\mathbf{a}}^t (q', \mathbf{a}(W'))$  when  $\mathbf{a}(W) = W$  and  $(q, W) \Rightarrow^t (q', W')$ . We define a transition relation  $\Rightarrow_{\mathbf{a}}$  to be the union of  $\Rightarrow_{\mathbf{a}}^t$  over all transitions  $t$ . For a finite abstraction operator  $\mathbf{a}$ , the *abstract zone graph*  $ZG^{\mathbf{a}}(\mathcal{A})$  consists as nodes pairs  $(q, W)$  of the form  $W = \mathbf{a}(W)$ . The initial node is  $(q_0, \mathbf{a}(Z_0))$  where  $(q_0, Z_0)$  is the initial node of  $ZG(\mathcal{A})$ . Transitions are given by the  $\Rightarrow_{\mathbf{a}}$  relation. Such a graph  $ZG^{\mathbf{a}}(\mathcal{A})$  can be seen as a Büchi automaton with the accepting states  $(q, W)$  for  $q \in F$ .

Abstractions for timed automata are parameterized by the maximum constants appearing in the guards of the automaton. The structure of the automaton determines two functions  $L : X \mapsto \mathbb{N}$  and  $U : X \mapsto \mathbb{N}$ . For a clock  $x$ , the value  $L(x)$  denotes the maximum constant occurring in guards of the form  $x \geq c$  or  $x > c$ ; and the value  $U(x)$  denotes the maximum constant occurring in guards  $x \leq c$  or  $x < c$ . This can be further refined by considering  $LU$  bounds for each state of the automaton [2]. In this paper we will use the abstraction operator  $\mathbf{a}_{\prec LU}$  [3] and the abstract zone graph  $ZG^{\mathbf{a}_{\prec LU}}(\mathcal{A})$  induced by it. It was shown in [7] that the  $\mathbf{a}_{\prec LU}$  abstraction induces the smallest zone graphs, for a given bound function  $LU$ . Moreover, we know from [12] that  $ZG^{\mathbf{a}_{\prec LU}}(\mathcal{A})$  is sound and complete for Büchi non-emptiness: TBA  $\mathcal{A}$  has a run satisfying the Büchi condition iff  $ZG^{\mathbf{a}_{\prec LU}}(\mathcal{A})$  has one. This gives an algorithm for the Büchi non-emptiness problem: given a TBA  $\mathcal{A}$ , compute the (finite) Büchi automaton  $ZG^{\mathbf{a}_{\prec LU}}(\mathcal{A})$  and check for its emptiness.

There is a challenge due to the use of the  $\mathbf{a}_{\prec LU}$  abstraction. There are zones  $Z$  for which  $\mathbf{a}_{\prec LU}(Z)$  is non-convex and hence it is better to avoid storing  $\mathbf{a}_{\prec LU}(Z)$ . Therefore, the solution to compute  $ZG^{\mathbf{a}_{\prec LU}}(\mathcal{A})$  works with a graph consisting of (state, zone) pairs and uses the  $\mathbf{a}_{\prec LU}$  abstraction indirectly [7]. The algorithm for computing  $ZG^{\mathbf{a}_{\prec LU}}(\mathcal{A})$  is shown in Figure 1. Since we consider only  $ZG^{\mathbf{a}_{\prec LU}}(\mathcal{A})$  in the rest of the paper, we will denote the transition relation  $\Rightarrow_{\mathbf{a}_{\prec LU}}$  by  $\rightarrow$ , as shown in Figure 1, for convenience. For a node  $n \in ZG^{\mathbf{a}_{\prec LU}}(\mathcal{A})$  we write  $n.q$  and  $n.Z$  for the state and zone present in node  $n$  respectively.

**Using subsumption to compute smaller graphs:** Although  $ZG^{\mathbf{a}_{\prec LU}}(\mathcal{A})$  is the smallest abstract zone graph for a given  $LU$ , its size could be (and usually is) exponential in the size

```

1  procedure abstract_zone_graph( $\mathcal{A}$ )
2   $V := \{(q_0, Z_0)\}$ ,  $\text{Waiting} := \{(q_0, Z_0)\}$ 
3   $\rightarrow := \emptyset$  // edge relation
4  while ( $\text{Waiting} \neq \emptyset$ )
5    take and remove  $(q, Z)$  from  $\text{Waiting}$ 
6    for each  $t = (q, g, R, q') \in \mathcal{A}$ 
7      compute  $(q, Z) \Rightarrow^t (q', Z')$ 
8      if  $\exists (q', Z_1) \in V$  s.t.  $\mathbf{a}_{\leq LU}(Z') = \mathbf{a}_{\leq LU}(Z_1)$ 
9        add  $(q, Z) \rightarrow (q', Z_1)$ 
10     else
11       add  $(q', Z')$  to  $V$  and  $\text{Waiting}$ 
12       add  $(q, Z) \rightarrow (q', Z')$ 
13  return  $(V, \rightarrow)$ 
14
15 procedure subsumption_graph( $\mathcal{A}$ )
16   $V := \{(q_0, Z_0)\}$ ,  $\text{Waiting} := \{(q_0, Z_0)\}$ 
17   $\rightarrow := \emptyset$  // edge relation
18   $\rightsquigarrow := \emptyset$  // subsumption relation
19  while ( $\text{Waiting} \neq \emptyset$ )
20    take and remove  $(q, Z)$  from  $\text{Waiting}$ 
21    for each  $t = (q, g, R, q') \in \mathcal{A}$ 
22      compute  $(q, Z) \Rightarrow^t (q', Z')$ 
23      if  $\exists (q', Z_1) \in V$  s.t.  $\mathbf{a}_{\leq LU}(Z') = \mathbf{a}_{\leq LU}(Z_1)$ 
24        add  $(q, Z) \rightarrow (q', Z_1)$ 
25      else if  $\exists (q', Z_1) \in V$  s.t.  $Z' \subseteq \mathbf{a}_{\leq LU}(Z_1)$ 
26        add  $(q', Z')$  to  $V$ 
27        add  $(q, Z) \rightarrow (q', Z') \rightsquigarrow (q', Z_1)$ 
28      else
29        add  $(q', Z')$  to  $V$  and  $\text{Waiting}$ 
30        add  $(q, Z) \rightarrow (q', Z')$ 
31  return  $(V, \rightarrow, \rightsquigarrow)$ 

```

■ **Figure 1** Algorithm on the left computes  $ZG^{\mathbf{a}_{\leq LU}}(\mathcal{A})$ . The algorithm on the right uses subsumption. Methods for testing  $Z' \subseteq \mathbf{a}_{\leq LU}(Z_1)$  and  $\mathbf{a}_{\leq LU}(Z') = \mathbf{a}_{\leq LU}(Z_1)$  are given in [7].

```

1  procedure ndfs()
2    Cyan := Blue := Red :=  $\emptyset$ 
3    dfsBlue( $s_0$ )
4    report no cycle
5
6  procedure dfsRed( $s$ )
7    Red := Red  $\cup \{s\}$ 
8    for all  $s \rightarrow t$  do
9      if (Cyan  $\sqsubseteq t$ ) then report cycle
10     if ( $t \not\sqsubseteq$  Red) then dfsRed( $t$ )
11
12 procedure dfsBlue( $s$ )
13   Cyan := Cyan  $\cup \{s\}$ 
14   for all  $s \rightarrow t$  do
15     if ( $t \not\subseteq$  Blue  $\cup$  Cyan and  $t \sqsubseteq$  Red)
16       then dfsBlue( $t$ )
17   if ( $s \in F$ ) then
18     dfsRed( $s$ )
19   Blue := Blue  $\cup \{s\}$ 
20   Cyan := Cyan  $\setminus \{s\}$ 

```

■ **Figure 2** Nested DFS algorithm with subsumption [11] to compute a subgraph of  $ZG^{\mathbf{a}_{\leq LU}}(\mathcal{A})$ .

of  $\mathcal{A}$ . An essential optimization that makes analysis of timed automata feasible is the use of subsumption. For two nodes  $t$  and  $s$  of  $ZG^{\mathbf{a}_{\leq LU}}(\mathcal{A})$  we say  $t$  is subsumed by  $s$ , written as  $t \sqsubseteq s$ , if  $t.q = s.q$  and  $t.Z \subseteq \mathbf{a}_{\leq LU}(s.Z)$ . The node  $s$  simulates  $t$ . Hence, at least for testing reachability, it is enough to keep in the graph only the maximal nodes with respect to subsumption. The algorithm incorporating subsumption is shown in Figure 1.

Subsumption optimization is known to give substantial gains for the reachability problem [10]. However, subsumption is not a priori correct for liveness, and the question of how it can be used for liveness was raised in [13]. An algorithm proposed in [11] (illustrated in Figure 2) gives a restricted way of using subsumption in a nested dfs algorithm for detecting accepting cycles. If we know that from a node  $s$  there is no reachable accepting cycle, then no node  $t \sqsubseteq s$  needs to be explored. The red nodes in the nested dfs algorithm play the role of node  $s$  (cf. Lines 10 and 15 in algorithm). Another optimization occurs in Line 9 - if there is a path from a node  $t$  to node  $s$  subsuming it, then a cycle can be concluded.

The goal of this paper is to find subsumption graphs of  $ZG^{\mathbf{a}_{\leq LU}}(\mathcal{A})$  that are sound and complete for liveness, and to design efficient algorithms to compute them.

### 3 Liveness compatible subsumptions

In this section, we are interested in understanding generic conditions as to when subsumption can be used correctly for liveness analysis. We start with an example. Consider the TBA  $\mathcal{A}$  and  $ZG^{\text{a}\leq\text{LU}}(\mathcal{A})$  illustrated in Figure 3. The zone graph has an accepting cycle on the node  $(1, 101 \leq y-x)$ . For each of the states 1, 2 and 3 of the TBA, there are at least 100 nodes in the zone graph. Note that  $(1, 1 \leq y-x) \sqsubseteq (1, 0 \leq y-x)$  and  $(2, 1 \leq y-x \leq 101) \sqsubseteq (2, 0 \leq y-x)$ . If we allow the luxury to use subsumptions freely, we would get the graph consisting only of the green nodes in the figure. However, in this graph there is no accepting cycle made uniquely of  $\rightarrow$  edges. There are cycles containing subsumption edges but, as we will see later, it is not sound to take such cycles as witnesses for the existence of an accepting computation. Hence the green graph is not complete for liveness: to detect an accepting computation we should not use subsumption on  $(1, 1 \leq y-x)$ . Observe that using subsumption on the node  $(2, 1 \leq y-x \leq 101)$  would do no harm, as further exploration would not lead to accepting cycles anyway. This subsumption gives already a significant gain. In fact, the zone graph restricted to the green and grey nodes, along with the subsumption edge on the right is a liveness complete graph according to the definition below. Algorithm in Figure 2 does not detect this possibility and explores the whole graph.

Our goal is to make use of subsumption as much as possible, subject to the restriction that the resulting graph contains an accepting cycle of  $\rightarrow$  edges iff  $ZG^{\text{a}\leq\text{LU}}(\mathcal{A})$  contains one. Including the subsumption edges as part of a cycle is not sound in general - for instance in Figure 3, the subsumption edge on state 2 forms a cycle, whereas there is no cycle containing 2 in  $ZG^{\text{a}\leq\text{LU}}(\mathcal{A})$ . In this paper, we do not include the subsumption edges as part of cycles. Hence in the graphs that we construct, cycles are actual cycles in  $ZG^{\text{a}\leq\text{LU}}(\mathcal{A})$  - so every such cycle with an accepting state gives an accepting computation. The challenge is to decide what are the subsumptions that are safe and can be left in the graph. We first make precise the notion of a zone graph with subsumptions, and then follow up with a condition that makes a zone graph with subsumption complete for liveness.

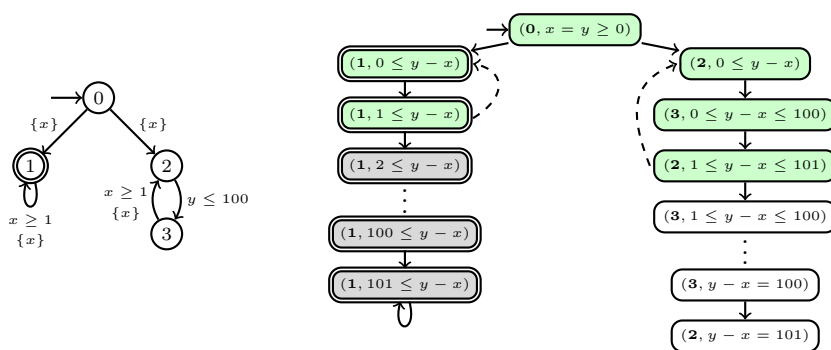
► **Definition 3** (Subsumption graph). Let  $G$  be a graph consisting of a subset of nodes and edges of  $ZG^{\text{a}\leq\text{LU}}(\mathcal{A})$  together with new edges called subsumption edges. Each node is labeled either *covered* or *uncovered*. Such a graph is called a *subsumption graph* if it satisfies the following conditions:

- C1** the initial node of  $ZG^{\text{a}\leq\text{LU}}(\mathcal{A})$  belongs to  $G$  and is labeled uncovered,
- C2** for every uncovered node  $s$ , all its successor transitions  $s \rightarrow s'$  occurring in  $ZG^{\text{a}\leq\text{LU}}(\mathcal{A})$  should be present in  $G$ ,
- C3** for every covered node  $t \in G$  there is an uncovered node  $s \in G$  such that  $t \sqsubseteq s$ ; moreover there is an explicit *subsumption edge*  $t \rightsquigarrow s$  in  $G$ ,
- C4** there is a path of  $\rightarrow$  edges from the initial node to every other node.

A path in a subsumption graph is made of both  $\rightarrow$  and  $\rightsquigarrow$  edges. We write  $s_1 \dashrightarrow^* s_2$  to denote that there is a path from  $s_1$  to  $s_2$  in the subsumption graph. We now describe the relation between paths in a zone graph and in a subsumption graph.

► **Lemma 4.** *For every (finite or infinite) path  $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots$  in  $ZG^{\text{a}\leq\text{LU}}(\mathcal{A})$  there is a path  $s'_0 \dashrightarrow^* s'_1 \dashrightarrow^* s'_2 \dashrightarrow^* \dots$  in  $G$  such that for each  $i$ ,  $s_i \sqsubseteq s'_i$  and  $s'_i$  is uncovered.*

Lemma 4 along with condition **C4** says that if there is a path  $s_0 \rightarrow^* s$  in  $ZG^{\text{a}\leq\text{LU}}(\mathcal{A})$ , there is a path  $s_0 \dashrightarrow^* s'$  with  $s \sqsubseteq s'$  in the subsumption graph  $G$ . This shows that subsumption graphs are complete for reachability. However, these conditions are not sufficient for liveness



■ **Figure 3** On the left is a TBA  $\mathcal{A}$ ; on the right the graph without the dashed edges is  $ZG^{a \Leftarrow LU}(\mathcal{A})$ . Assume that  $L = U = 100$  at every state - this can be achieved by adding more transitions on each state (which are not shown for clarity). Dashed edges show subsumption. The part of  $ZG^{a \Leftarrow LU}(\mathcal{A})$  restricted to green nodes and dashed edges is the zone graph with subsumption. In this green graph there is no accepting cycle consisting of  $\rightarrow$  edges. Removing the dashed edge on the node  $(1, 1 \leq y - x)$  and adding the grey nodes identifies the accepting cycle.

– for a cycle of  $\rightarrow$  edges in the zone graph, we may not get a corresponding cycle of  $\rightarrow$  edges in the subsumption graph (cf. Figure 3). We now give an extra criterion.

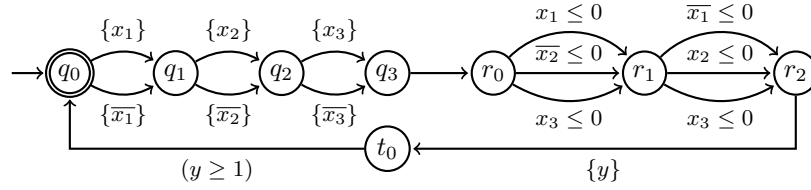
► **Definition 5** (Liveness compatible subsumption graph). A subsumption graph  $G$  is said to be *liveness compatible* if it additionally satisfies the following condition:

**C5** there is no cycle containing both an accepting node and a subsumption edge.

In Figure 3, the zone graph restricted to green nodes and the dashed edges is not liveness compatible. There is a cycle containing an accepting node  $(1, 1 \leq y - x)$  and a subsumption edge from this node. However, removing this subsumption edge and adding the grey nodes makes it liveness compatible. The only subsumption edge is from  $(2, 1 \leq y - x \leq 101)$  and it is not part of a cycle containing an accepting node. Intuitively, when we add a subsumption edge  $t \rightsquigarrow s$ , we know that paths in  $ZG^{a \Leftarrow LU}(\mathcal{A})$  starting from  $t$  can be simulated from  $s$  in the subsumption graph. But if there is a cycle containing  $t \rightsquigarrow s$  in the subsumption graph, this would mean that the simulation from  $s$  can bring us back to  $t$ . Hence some accepting runs from  $t$  in  $ZG^{a \Leftarrow LU}(\mathcal{A})$  could be lost in the subsumption graph. We show that condition **C5** above makes such a situation impossible.

► **Theorem 6.**  $ZG^{a \Leftarrow LU}(\mathcal{A})$  has an infinite accepting path iff a liveness compatible subsumption graph has an infinite accepting path consisting of  $\rightarrow$  edges.

**Proof.** Let  $G$  be a liveness compatible subsumption graph. Since all the  $\rightarrow$  edges in  $G$  come from the zone graph, a cycle of  $\rightarrow$  edges in  $G$  implies such a cycle in the zone graph. This shows the direction from right to left. Suppose  $ZG^{a \Leftarrow LU}(\mathcal{A})$  has an accepting run  $\rho: s_0 \rightarrow s_1 \rightarrow \dots$ . From Lemma 4, we have a path  $\rho'$  in  $G$  of the form:  $s'_0 \dashrightarrow^* s'_1 \dashrightarrow^* s'_2 \dashrightarrow^* \dots$  such that each  $s_i \sqsubseteq s'_i$ . Since  $\rho$  is an accepting run, some accepting node  $s$  repeats infinitely often in  $\rho$ . Corresponding positions in  $\rho'$  contain nodes which subsume  $s$ . Since there are finitely many nodes in  $G$ , there should be some accepting node  $s'$  which occurs infinitely often in  $\rho'$ . Therefore there is a cycle containing  $s'$  in  $G$ . By liveness compatibility criterion **C5** this cycle should be made of only  $\rightarrow$  edges. From condition **C4**, there should be a path consisting of  $\rightarrow$  edges from the initial node of  $G$  to  $s'$ . This gives an infinite path in  $G$  made of  $\rightarrow$  edges that visits an accepting node  $s'$  infinitely often. ◀



■ **Figure 4** Automaton for  $\phi = (p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee p_3)$ .

#### 4 Liveness is more difficult than reachability

Theorem 6 says that to solve the Büchi non-emptiness problem, one can compute a liveness compatible subsumption graph and check for cycles containing an accepting node and no subsumption edge. In general, liveness compatible subsumption graphs are smaller than the zone graph, but bigger than the usual subsumption graphs computed for reachability (cf. Figure 3). A natural question now is to ask if one can quantify this overhead created due to the liveness compatibility. In this section, we show that deciding liveness from a (reachability compatible) subsumption graph is NP-hard. Therefore, unless  $P=NP$ , one needs an object exponentially bigger than subsumption graphs to decide liveness. The proof of the following theorem uses the same kind of gadget as in [6].

► **Theorem 7.** *Given a strongly non-Zeno TBA and its subsumption graph, deciding Büchi non-emptiness is NP-hard.*

**Proof.** We give a reduction from 3SAT. Let  $P = \{p_1, \dots, p_k\}$  be a set of propositional variables and let  $\phi = C_1 \wedge \dots \wedge C_n$  be a 3CNF formula with  $n$  clauses. We will construct an automaton  $\mathcal{B}_\phi$  such that  $\phi$  has a satisfying assignment iff  $\mathcal{B}_\phi$  has an infinite run satisfying the Büchi condition. Moreover, we give a subsumption graph with the same number of nodes as the number of states in  $\mathcal{B}_\phi$ .

The timed automaton  $\mathcal{B}_\phi$  is defined as follows. For each propositional variable  $p_i$ , there are two clocks,  $x_i$  and  $\bar{x}_i$ , which encode the value of  $p_i$  when the value of one of the two clocks is 0 and the other is not. In addition, there is a clock  $y$ . In all, the set of clocks  $X$  is  $\{x_1, \bar{x}_1, \dots, x_k, \bar{x}_k, y\}$ . For a literal  $\lambda$ , let  $cl(\lambda)$  denote the clock  $x_i$  when  $\lambda = p_i$  and the clock  $\bar{x}_i$  when  $\lambda = \neg p_i$ . The set of states of  $\mathcal{B}_\phi$  is  $\{q_0, \dots, q_k, r_0, \dots, r_n, t_0\}$  with  $q_0$  being the initial and the accepting state. The transitions are as follows: for each propositional variable  $p_i$  we have transitions  $q_{i-1} \xrightarrow{\{x_i\}} q_i$  and  $q_{i-1} \xrightarrow{\{\bar{x}_i\}} q_i$ ; for each clause  $C_m = \lambda_1^m \vee \lambda_2^m \vee \lambda_3^m$ ,  $m = 1, \dots, n$ , there are three transitions  $r_{m-1} \xrightarrow{cl(\lambda) \leq 0} r_m$  for  $\lambda \in \{\lambda_1^m, \lambda_2^m, \lambda_3^m\}$ ; a transition  $q_k \rightarrow r_0$  with no guards and no resets; transitions  $r_n \xrightarrow{\{y\}} t_0$  and  $t_0 \xrightarrow{y \geq 1} q_0$ .

Figure 4 shows the automaton for the formula  $(p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee p_3)$ . A path from  $q_0$  to  $q_3$  encodes an assignment with the following convention: a reset of  $x_i$  represents  $p_i \mapsto true$  and a reset of  $\bar{x}_i$  means  $p_i \mapsto false$ . Then, from  $r_0$  to  $r_2$  we check if the formula is satisfied by this guessed assignment (clearly there is no point to let the time pass in this process). The additional parts are the transitions from  $r_2 \rightarrow t_0$  and  $t_0 \rightarrow q_0$ . Note that this makes the automaton strongly non-Zeno: between two consecutive visits to  $q_0$ , at least 1 time unit should elapse.

If  $\phi$  has a satisfying assignment, then for every  $p_i$  the path that resets  $x_i$  if  $p_i$  is true and resets  $\bar{x}_i$  if  $p_i$  is false can be extended to a run reaching  $t_0$ . For instance, the formula from Figure 4 is satisfied by every assignment that maps  $p_3$  to *true*. The path corresponding to such an assignment passes through the transition  $q_2 \xrightarrow{\{x_3\}} q_3$ . Then, it has the possibility

to follow transitions  $r_0 \xrightarrow{x_3 \leq 0} r_1$  and  $r_1 \xrightarrow{x_3 \leq 0} r_2$ . Note that this is possible independently on the amount of time elapsed at  $q_0$ . In particular, after the first iteration from  $q_0 \rightarrow q_0$  the values of all clocks become bigger than 1. By following the path corresponding to the satisfying assignment each time,  $q_0$  can be visited infinitely often. Conversely, suppose  $\mathcal{B}_\phi$  has an infinite run that visits  $q_0$  infinitely often. Since after one iteration  $q_0 \rightarrow q_0$  the value of all clocks are above 1, the only way to take transitions  $r_0 \rightarrow r_1 \cdots \rightarrow r_n$  is by resetting clocks appropriately so that at least one guard is satisfied at every  $r_i$ . One such sequence of resets would then give a satisfying assignment for  $\phi$ .

We now give an idea of the subsumption graph  $G_\phi$  for  $\mathcal{B}_\phi$  as computed by the algorithm in the right hand side of Figure 1. It turns out that for every state  $q_0, \dots, q_n$  and  $r_0, \dots, r_n$  there is a single node with the zone  $\mathbf{a}_{\preccurlyeq LU}(Z_0)$ , where  $Z_0$  is the initial zone, consisting of the time successors of  $v_0$  (the valuation mapping every clock to 0). This is because  $\mathbf{a}_{\preccurlyeq LU}$  abstraction does not remember the order of resets. If it had, there would be exponentially many nodes at  $r_n$ . It can be checked that transitions  $r_n \rightarrow t_0 \rightarrow q_0$  yield a node  $(q_0, \mathbf{a}_{\preccurlyeq LU}(Z'_0))$  where  $\mathbf{a}_{\preccurlyeq LU}(Z'_0)$  is strictly included in  $\mathbf{a}_{\preccurlyeq LU}(Z_0)$ . ◀

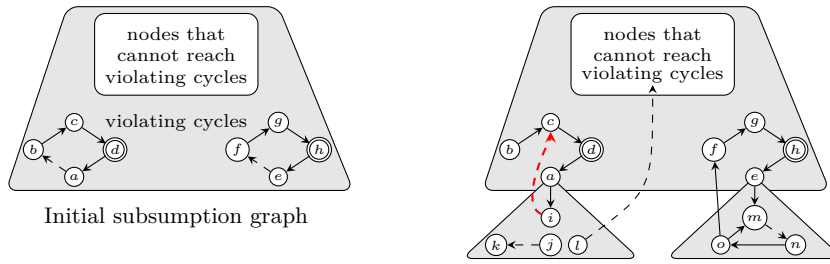
The graph  $G_\phi$  in the previous proof is not a liveness compatible subsumption graph. An algorithm for liveness would explore further from  $(q_0, \mathbf{a}_{\preccurlyeq LU}(Z'_0))$  till a complete graph is obtained. Theorem 7, says that this process essentially leads to SAT solving. The above theorem holds even if the less coarse abstraction  $\text{Extra}_{LU}^+$  [3] is used instead of  $\mathbf{a}_{\preccurlyeq LU}$ .

## 5 An algorithm

We now consider the algorithmic problem of computing a liveness compatible subsumption graph for a given automaton. The objective is of course to compute a small graph, as otherwise we could just compute the entire abstract zone graph without subsumption using the algorithm from the left of Figure 1. A better solution is the nested DFS algorithm in Figure 2 - indeed the final graph computed by it is a liveness compatible subsumption graph. In this section, we present a different algorithm, and compare its performance with the nested DFS approach. Our algorithm iterates between a reachability computation and an SCC analysis of the computed graph to find cycles violating condition **C5** in Definition 5. Figure 5 illustrates the idea. The picture on the left shows the situation after the first reachability-SCC analysis. At this point, each violating subsumption edge is removed and a subsumption graph computation is started from the corresponding covered node (nodes  $a$  and  $e$  in the figure). During this exploration, the use of subsumption is restricted in order to avoid falling repeatedly into the same bad cycle: as  $a$  is covered by  $b$  then  $i$  must be covered by  $c$ , but this covering edge will form a bad cycle anyway, so there is no point of introducing it. To achieve this behavior, a *level* field is added to each node and subsumption is allowed only to nodes of a higher level (subsumption  $j \rightsquigarrow k$  and the subsumption on node  $l$ , assuming nodes in the white part get level  $\infty$ ). The iteration between reachability and cycle detection phases continues till the computed graph does not have violating cycles.

The problem with this approach is that the same edges will be considered in an unbounded number of SCC analyses. To avoid this, each SCC analysis phase is restricted to the nodes and edges of the current level. This however could miss out certain violations that span across levels (like the one caused by the edge  $o \rightarrow f$ ). The following algorithm handles this issue by considering all such exit edges as potential causes of violation.





■ **Figure 5** On the left is an illustration of a subsumption graph. An SCC decomposition of this graph gives the nodes that are part of violating cycles, and those that cannot reach violating cycles. On the right, is a re-exploration from bad subsumptions. In the re-exploration, subsumption is restricted to nodes of same level, or nodes that are known not to reach violating cycles.

### Iterative SCC based algorithm with subsumption:

**Phase 0** Let  $K = 1$  and let  $S_{init}$  and  $S$  be the sets containing the initial (state,zone) pair.

**Phase 1** Construct a subsumption graph from nodes in  $S_{init}$ . Set the level field of all nodes in  $S_{init}$  to  $K$  and let  $S_{init}$  be the pool of nodes to be explored. Every node added in this phase will have level field set to  $K$ . Repeatedly, take a node  $(q, Z)$  from the pool. For every edge  $(q, Z) \Rightarrow (q', Z')$ , add node  $(q', Z')$  to  $S$  and to the pool unless there is already  $(q', Z_1) \in S$  s.t., either:

1.1  $\alpha_{\leq LV}(Z') = \alpha_{\leq LV}(Z_1)$ , or

1.2  $\alpha_{\leq LV}(Z') \subset \alpha_{\leq LV}(Z_1)$ , state  $q'$  is non-accepting, node  $(q', Z_1)$  is uncovered and has a level  $K$  or  $\infty$ .

In the first case, add the edge  $(q, Z) \rightarrow (q', Z_1)$  to  $S$ . In the second case, add the node  $(q', Z')$  to  $S$  and the edges  $(q, Z) \rightarrow (q', Z')$ ,  $(q', Z') \rightsquigarrow (q', Z_1)$  to  $S$ . By the end of this phase, graph  $S$  will be extended with some nodes of level  $K$ .

**Phase 2** Consider the subgraph  $G_K$  of  $S$  induced by nodes of level  $K$ , and containing all the  $\rightarrow$  and  $\rightsquigarrow$  edges between these nodes. Decompose  $G_K$  into maximal SCCs by considering both  $\rightarrow$  and  $\rightsquigarrow$  as the same kind of edges. Mark a maximal SCC as *bad* if:

2.1 either it contains an accepting node and a subsumption edge (both nodes adjacent to the  $\rightsquigarrow$  edge must be in the SCC), or

2.2 it has a node  $s$  with a successor edge  $s \rightarrow s'$  to a node  $s'$  of level strictly less than  $K$ . Note that node  $s$  is in  $G_K$ , but  $s'$  is not.

For every node in  $G_K$ : set the flag of the node to  $\infty$  if it cannot reach a bad SCC.

**Phase 3** Let  $S_{init}$  be all covered nodes in  $G_K$  which still have level  $K$ . Remove the corresponding subsumption edges. Set  $K := K + 1$ .

**Repeat** If  $S_{init}$  is non-empty, restart from Phase 1.

**Final** If  $S_{init}$  is empty, perform an SCC decomposition of  $S$ . If there is an accepting cycle of  $\rightarrow$  edges, return *non-empty*, else return *empty*.

In the above algorithm, Phase 1 is a reachability computation and Phases 2 and 3 are SCC-analysis. The final phase is a usual cycle detection algorithm.

► **Theorem 8.** *A strongly non-Zeno TBA  $\mathcal{A}$  is non-empty iff the iterative SCC based algorithm with subsumption returns non-empty.*

Although the iterative algorithm performs an unbounded number of calls to an SCC decomposition, each edge is visited in only two SCC decompositions - one in the Phase 2 of the iteration corresponding to the level of the source node when it appeared, and the second

■ **Table 1** Comparison of the size of liveness invariants generated by nested DFS algorithm, nested DFS algorithm with subsumption [11] and our Iterative algorithm running a Topological Search (see [10]) on the benchmark from [11]. We used a Linux station with an Intel i7-2600 3.40GHz processor and 8Gb of memory.

Prop	Sat	Nested DFS		NDFS subsumption		Iterative TS			UPPAAL TS
		# nodes	sec.	# nodes	sec.	# nodes	K	sec.	# nodes
Fi1	✓	26 651	0.2	26 651	0.2	7 737	1	0.3	7 737
Fi2	✓	132 808	1.4	132 808	1.3	38 238	1	1.0	38 238
Fi3		26 679	0.4	26 679	0.4	20 768	1	0.8	20 768
FD1	✓	19 858	0.3	18 246	0.2	705	1	0.1	705
CC1	✓	1 786 399	31.6	1 786 399	32.7	15 837	1	1.3	15 837
CC2	✓	22 070	0.4	22 070	0.4	12 898	1	0.3	12 898

one during the final phase. This gives a constant upper bound on the number of times an edge is visited, irrespective of the number of iterations needed for stabilization. Therefore the time spent by the algorithm is linear in the size of the final graph computed, similar to the algorithm in Figure 2 which visits each edge twice. Moreover the result is always included in an abstract zone graph (without subsumption).

Let us now compare this approach with the nested DFS algorithm from Figure 2. The advantages of our algorithm are clearly visible on a (quite pathological) case of an automaton having no reachable accepting state. In this case, nested DFS does not use subsumption at all as there are no red nodes. On the other hand, the iterative algorithm computes just a reachability subsumption graph. After the first iteration, there would be no bad SCCs, and hence the entire graph would be marked  $\infty$ . Another important class of automata for which we get a significant gain is weak timed Büchi automata (every cycle in a weak TBA consists entirely of accepting nodes or non-accepting nodes). As there can be no SCCs containing both accepting and non-accepting nodes, the iterative algorithm stabilizes after 1 iteration. The automaton of Figure 3 is a weak Büchi automaton. The nested DFS algorithm computes the entire zone graph - no subsumption will be allowed. The iterative algorithm would allow subsumption on the non-accepting part. It computes the graph consisting of the green and grey nodes. There are examples where the nested DFS approach can outperform the iterative algorithm. Modify the automaton in Figure 3 as follows: make all states accepting; add another clock  $w$  and an edge  $0 \xrightarrow{w \geq 5} 2$ . Note that the iterative algorithm cannot use subsumption at all since all nodes are accepting. Moreover this additional edge leads to  $n_1 : (2, 0 \leq y - x \wedge w \geq 5)$  which starts a long thread of zones due to the transitions between 2 and 3. Note that  $n : (2, 0 \leq y - x \wedge w \geq 5)$  is subsumed by  $(2, 0 \leq y - x)$ . If the NDFS chooses a good order and finishes exploring the latter zone first, it allows subsumption  $n_1 \rightsquigarrow n$ , and avoids computing the subtree below  $n_1$ .

## 6 Experiments

We have implemented our algorithm in our tool TChecker. Our implementation includes various optimizations that are not presented in this paper (the algorithm stops after Phase 1 when no accepting state is reachable, covered nodes are not kept in the liveness invariants, etc.). We have compared our algorithm with our own implementation of the nested DFS algorithm with subsumption from [11]. We have conducted experiments on the classical benchmarks for Timed Automata, that we describe below. We verify properties given by Büchi automata on these standard models. To do this, we take the product of the model with a property automaton, and check for Büchi non-emptiness on this product.

■ **Table 2** Comparison of the nested DFS algorithm with subsumption[11] and our Iterative algorithm running a Topological Search (see[10]) on properties with timed constraints. We used a Linux station with an Intel i7-2600 3.40GHz processor and 8Gb of memory.

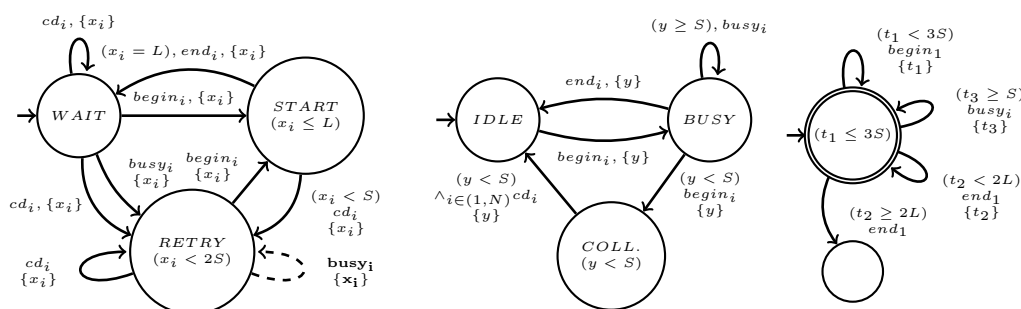
Prop	Sat	NDFS subsumption		Iterative TS			UPPAAL TS
		# visited	# nodes	# visited	# nodes	K	# nodes
Fig 4		58	50	116	50	3	8
(CC3) 8		16 165	16 164	219 561	205 656	1	205 656
(CC4) 3	✓	44 873	35 787	449 724	150 078	282	367
(CC5) 3	✓	240 111	237 548	606 421	201 740	60	772
(Fi4) 8	✓	466 572	382 936	154 854	77 427	1	77 427
(Fi5) 4	✓	48 299	24 979	88 430	29 677	17	704

As a first experiment, we considered the models and properties in [11]. Fischer’s protocol is a mutual-exclusion protocol based on real-time constraints. Let  $c$  denote the number of processes in critical section. We checked properties (Fi1)  $G(c \leq 1)$  mutual exclusion; (Fi2)  $GF(c = 0) \wedge GF(c = 1)$  non-blocking from [12]; and (Fi3)  $G(req_1 \implies Fcs_1)$  every request of process 1 is eventually satisfied. FDDI is a token ring protocol where the communication can be synchronous or asynchronous. We checked property (FD1)  $FG(async_1)$  process 1 eventually communicates asynchronously. Finally, CSMA/CD is a protocol to detect and solve message collisions that is used for communications over Ethernet networks. We checked two properties (CC1)  $G(collision \implies Factive)$  after a collision, the network becomes active again; and (CC2)  $FG(collision \implies G\neg sent)$  after some collisions, no message can ever be sent. We consider instances of the 3 models with 7 processes and standard parameter values from the literature.

Table 1 shows a comparison between the standard nested DFS algorithm, nested DFS algorithm with subsumption [11] and our Iterative algorithm. The first column corresponds to the property checked as described above. A tick in the second column indicates that the property holds: the Büchi automaton for the complement has no accepting run. For every algorithm, we report the size of the liveness invariant (# nodes) and the running time (sec.). We also report the maximum level (K) for the Iterative algorithm. The last column gives the size of the reachability invariant as computed by our implementation of UPPAAL’s algorithm [16]. This is a lower bound on the size of the liveness invariant. Both our Iterative algorithm and UPPAAL’s algorithm explore the state-space of the automata using a topological search [10].

The results show that our Iterative algorithm computes liveness invariants that are significantly smaller than those computed by both nested-DFS algorithms. Indeed, for all these examples, the reachability invariants are liveness compatible as shown by the comparison to UPPAAL’s algorithm. Our algorithm also visits significantly less nodes than nested-DFS algorithms. It stops immediately after Phase 1 for (Fi1) and (CC2) that have no reachable accepting state. Models (Fi2), (FD1) and (CC1) have reachable accepting states, but no bad SCC. As a result, the Iterative algorithm stops after Phase 2 and skips Phase 3 and the final phase. Finally, (Fi3) has an accepting run that is found during Phase 2.

We have also compared the algorithms on examples that are particularly difficult for Iterative algorithm. The results are shown in Table 2. We report for each model the number of visited nodes (# visited) and the size of the liveness invariants (# nodes). We also compare to the size of reachability invariants generated by UPPAAL’s algorithm. The first example corresponds to the automaton in Figure 4. Since it is built from a formula that is satisfiable, the automaton has an accepting run. However, the accepting cycle spans over two levels. As a result, our Iterative algorithm required 2 refinements ( $K = 3$ ), and it had to run all phases to detect the accepting run. The nested-DFS algorithm explores significantly less nodes.



■ **Figure 6** Model of CSMA/CD: station (left) and bus (middle); property (CC3) (right).

The other examples are built from the CSMA/CD model (CC) and the Fischer model (Fi) described above. We consider timed specifications with timed constraints that make covering harder. Property (CC3) is depicted in Figure 6 (right). The automaton checks that station 1 tries to transmit fast enough, and that it often achieves successful transmissions. Property (CC4) is a variation of (CC3) where cycles can be iterated only a bounded number of times. This is achieved by adding a new clock  $t_4$  that is never reset, and an invariant  $t_4 \leq K$  to the accepting state. Property (CC5) checks that if collisions are infrequent and station 1 tries infinitely often to send, then it effectively sends messages infinitely often. Property (Fi4) expresses that if process 1 can infinitely often access the critical section for  $K$  time units, then it enters the critical section infinitely often. Finally, property (Fi5) checks that process 1 requests access to the critical section frequently, but is only granted access in a certain time window. As for (CC4) the cycles in (Fi5) can be iterated only a bounded number of times.

Property (CC3) has accepting runs, thus the nested-DFS algorithm with subsumption performs significantly better than our Iterative algorithm. Indeed, the Iterative algorithm first computes a reachability invariant in Phase 1 before being able to detect the accepting run in Phase 2. Properties (CC4) and (Fi5) with bounded iterations of cycles are difficult for our Iterative algorithm. These two automata generate many bad SCCs, hence many refinements. At each refinement step, our algorithm needs to generate many new nodes as covering is restricted to nodes within the same level (or nodes with level  $\infty$ ). This results in bigger liveness invariants than those computed by nested-DFS algorithm with subsumption. On the examples (CC5) and (Fi4), on the contrary, our algorithm generates smaller invariants than nested-DFS with subsumption. Notice that in most examples, the liveness invariants computed by both algorithms are huge w.r.t. reachability invariants computed by UPPAAL's algorithm.

Finally, the case of CSMA/CD gives an interesting motivation for testing Büchi properties. Indeed property (CC2) holds for the CSMA/CD model. As a result, the model is not correct since communications should be enabled after a collision. It turns out that a transition is missing in the widely used model [14, 17]. In consequence, once some process enters in a collision, no process can send a message afterwards. The model can be fixed by allowing the *busy* action in state *RETRY* as shown in Figure 6. This example confirms once more that timed models are compact descriptions of complicated behaviors due to both parallelism and interaction between clocks. Büchi properties can be extremely useful in making sure that a model works as intended.

## 7 Conclusions

As we show in this paper, the liveness problem for timed automata is substantially more difficult algorithmically than the reachability problem. We have defined a notion of an invariant for liveness properties: a graph proving that the property does not hold. We have also proposed a high-level algorithm for constructing such an invariant. Finally, we have reported on some experiments with a preliminary implementation of this algorithm. Further work will be required to understand the relation between sizes of liveness and safety invariants, as well as to develop better algorithms for constructing liveness invariants.

---

### References

- 1 R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- 2 G. Behrmann, P. Bouyer, E. Fleury, and K.G. Larsen. Static guard analysis in timed automata verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–270. Springer, 2003.
- 3 G. Behrmann, P. Bouyer, K.G. Larsen, and R. Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *STTT*, 8(3):204–215, 2006.
- 4 C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS*, volume 1384 of *LNCS*, pages 313–329, 1998.
- 5 D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, pages 197–212, 1989.
- 6 F. Herbreteau and B. Srivathsan. Coarse abstractions make zeno behaviours difficult to detect. *Logical Methods in Computer Science*, 9, 2013.
- 7 F. Herbreteau, B. Srivathsan, and I. Walukiewicz. Better abstractions for timed automata. In *LICS*, pages 375–384. IEEE Computer Society, 2012.
- 8 F. Herbreteau, B. Srivathsan, and I. Walukiewicz. Efficient emptiness check for timed Büchi automata. *Formal Methods in System Design*, 40(2):122–146, 2012.
- 9 F. Herbreteau, B. Srivathsan, and I. Walukiewicz. Lazy abstractions for timed automata. In *CAV*, volume 8044 of *LNCS*, pages 990–1005, 2013.
- 10 F. Herbreteau and T.-T. Tran. Improving search order for reachability testing in timed automata. In *FORMATS*, pages 124–139. Springer, 2015.
- 11 A. Laarman, M.C. Olesen, A.E. Dalsgaard, K.G. Larsen, and J. van de Pol. Multi-core emptiness checking of timed Büchi automata using inclusion abstraction. In *CAV*, volume 8044 of *LNCS*, pages 968–983, 2013.
- 12 G. Li. Checking timed Büchi automata emptiness using LU-abstractions. In *FORMATS*, pages 228–242, 2009.
- 13 S. Tripakis. Checking timed büchi automata emptiness on simulation graphs. *ACM Transactions on Computational Logic (TOCL)*, 10(3):15, 2009.
- 14 S. Tripakis and S. Yovine. Analysis of timed systems using time-abstraction bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.
- 15 S. Tripakis, S. Yovine, and A. Bouajjani. Checking timed Büchi automata emptiness efficiently. *Formal Methods in System Design*, 26(3):267–292, 2005.
- 16 UPPAAL. <http://www.uppaal.org>.
- 17 UPPAAL CSMA/CD model. [http://www.it.uu.se/research/group/darts/uppaal/benchmarks/genCSMA\\_CD.awk](http://www.it.uu.se/research/group/darts/uppaal/benchmarks/genCSMA_CD.awk). Accessed: 2014-10-08.

# Verified Analysis of List Update Algorithms\*

Maximilian P. L. Haslbeck<sup>1</sup> and Tobias Nipkow<sup>2</sup>

<sup>1</sup> Technische Universität München, Germany

<sup>2</sup> Technische Universität München, Germany

---

## Abstract

This paper presents a machine-verified analysis of a number of classical algorithms for the list update problem: 2-competitiveness of move-to-front, the lower bound of 2 for the competitiveness of deterministic list update algorithms and 1.6-competitiveness of the randomized COMB algorithm, the best randomized list update algorithm known to date. The analysis is verified with help of the theorem prover Isabelle; some low-level proofs could be automated.

**1998 ACM Subject Classification** F.3.1 Specifying and Verifying Programs and F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Program Verification, Algorithm Analysis, Online Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.49

## 1 Introduction

Interactive theorem provers have been applied to deep theorems in mathematics [9, 10] or fundamental software components [20, 22] but hardly to quantitative algorithm analysis. This paper demonstrates that nontrivial results from that area are amenable to verification (which for us always means “interactive”) by analyzing the best known deterministic and randomized online algorithms for the list update problem with the help of the theorem prover Isabelle/HOL [25, 26]. Essentially, this paper formalizes the main results of the first two chapters of the classic text by Borodin and El-Yaniv [6]: 2-competitiveness of move-to-front (MTF), the lower bound of 2 for the competitiveness of all deterministic list update algorithms and 1.6-competitiveness of COMB [2], the best randomized online list update algorithm known to date. For reasons of space we are forced to refer the reader to the online formalization [12] (15600 lines) for many of the definitions and all of the formal proofs.

The list update problem is a simple model to study the competitive analysis of online algorithms (where requests arrive one by one) compared to offline algorithms (where the whole sequence of requests is known upfront). In the simplest form of the problem we are given a list of elements that can only be searched sequentially from the front and each request asks if some element is in that list. In addition to searching for the element the algorithm may rearrange the list by swapping any number of adjacent elements to improve the response time for future requests. One is usually not interested in the offline algorithm (the problem is NP-hard [3]) but merely uses it as a benchmark to compare online algorithms against.

This paper advocates to extend verification of algorithms from functional correctness to quantitative analysis. There are a number of examples of such verifications for classical algorithms, but this is the first verified analysis of any online algorithm. Our verified proof of the 1.6-competitiveness of COMB appears to be one of the most complex verified analyses

---

\* Supported by DFG Koselleck grant NI 491/16-1.



© Maximilian P. L. Haslbeck and Tobias Nipkow;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 49; pp. 49:1–49:15



Leibniz International Proceedings in Informatics

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



of a randomized algorithm to date. Our paper should be read as a contribution to the formalization of computer science foundations, here quantitative algorithm analysis.

It should be noted that although our verification is interactive, some tedious low-level proofs could be automated (see the verification of algorithm BIT in Section 4.6).

The paper is structured as follows: Section 2 explains our notation. Sections 3 and 4 roughly follow chapters 1 and 2 of [6], with some omissions: Section 3 formalizes deterministic list update algorithms, analyzes MTF and proves a lower bound. Section 4 formalizes randomized list update algorithms, proves two analysis techniques (list factoring and phase partitioning), and analyzes the algorithms BIT, TS and finally COMB.

## 1.1 Related Work

This work grew out of an Isabelle-based framework for verified amortized analysis applied to classical data structures like splay trees [24]. Charguéraud and Pottier [7] verified the almost-linear amortized complexity of Union-Find in Coq. The verification of randomized algorithms was pioneered by Hurd *et al.* [15, 16, 17] who verified the Miller-Rabin probabilistic primality test and part of Rabin’s probabilistic mutual exclusion algorithm. Barthe *et al.* (e.g. [5]) verify probabilistic security properties of cryptographic constructions.

An orthogonal line of research is the automatic resource bound analysis of deterministic functional or imperative programs, e.g., [13].

The list update problem is still an active area of research [23]; for a survey see [19].

## 2 Notation

Isabelle’s higher-order logic is a simply typed  $\lambda$ -calculus: function application is written  $f x$  and  $g x y$  rather than  $f(x)$  and  $g(x,y)$ ; binary functions usually have type  $A \rightarrow B \rightarrow C$  instead of  $A \times B \rightarrow C$ , and analogously for *n*ary functions;  $\lambda x. t$  is the function that maps argument  $x$  to result  $t$ . The notation  $t :: \tau$  means that term  $t$  has type  $\tau$ .

The type of lists over a type  $\alpha$  is  $\alpha$  *list*. The empty list is  $[]$ , prepending an element  $x$  in front of a list  $xs$  is written  $x \cdot xs$  and appending two lists is written  $xs @ ys$ . The length of  $xs$  is  $|xs|$ . Function *set* converts a list into a set. The predicate *distinct xs* expresses that there are no duplicates in  $xs$ . The *i*th element of  $xs$  (starting at 0) is  $xs_i$ . By *index xs x* we denote the index (starting at 0) of the first occurrence of  $x$  in  $xs$ ; if  $x$  does not occur in  $xs$  then *index xs x* =  $|xs|$ . If  $x$  occurs before  $y$  in  $xs$  we write  $x < y$  *in xs*:

$$x < y \text{ in } xs \iff \text{index } xs \ x < \text{index } xs \ y \wedge y \in \text{set } xs$$

The condition  $x \in \text{set } xs$  is implied by the right-hand-side.

Given two lists  $xs$  and  $ys$ , we call a pair  $(x, y)$  an *inversion* if  $x$  occurs before  $y$  in  $xs$  but  $y$  occurs before  $x$  in  $ys$ . The set of inversions is defined like this:

$$\text{Inv } xs \ ys = \{(x, y) \mid x < y \text{ in } xs \wedge y < x \text{ in } ys\}$$

Given a list  $xs$  and two elements  $x$  and  $y$ , let  $xs_{xy}$  denote the projection of  $xs$  on  $x$  and  $y$ , i.e., the result of deleting from  $xs$  all elements other than  $x$  and  $y$ .

Note that the L<sup>A</sup>T<sub>E</sub>X presentations of definitions and theorems in this paper are generated by Isabelle from the actual definitions and theorems. To increase readability we employed Isabelle’s pretty-printing facilities to emulate the notation in [6]. This has to be taken into account when comparing formulas in the paper with the actual Isabelle text [12].



## 2.1 Probability Mass Functions

Type  $\alpha$  *pmf* of *probability mass functions* [14, §4] represent distributions of discrete random variables on a type  $\alpha$ . Function  $set_{pmf} D$  denotes the support set of the distribution  $D$  and  $pmf D e$  denotes the probability of element  $e$  in the distribution  $D$ .

► **Example 1.** Our background theory defines the Bernoulli distribution  $bernoulli_{pmf}$ , a *pmf* on the type *bool* which satisfies (amongst others) the following properties:

$$\begin{aligned} set_{pmf} (bernoulli_{pmf} p) &\subseteq \{True, False\} \\ pmf (bernoulli_{pmf} (1/2)) x &= 1/2 \\ 0 \leq p \wedge p \leq 1 &\implies pmf (bernoulli_{pmf} p) True = p \end{aligned}$$

Furthermore the monadic operators  $bind_{pmf} :: \alpha pmf \rightarrow (\alpha \rightarrow \beta pmf) \rightarrow \beta pmf$  and  $return_{pmf} :: \alpha \rightarrow \alpha pmf$ , as well as the operator  $map_{pmf} :: (\alpha \rightarrow \beta) \rightarrow \alpha pmf \rightarrow \beta pmf$  are defined. With the help of these functions more complex *pmfs* can be synthesized from simpler ones. To demonstrate how to work with *pmf*, we define *bv n* the uniform distribution over bit vectors of length  $n$  recursively.

► **Example 2.** This is an example of probabilistic functional programming [8] with the help of Haskell's *do*-notation (which is just syntax for the *bind* operator).

```

bv 0 = returnpmf []
bv (n + 1) = do {
  x ← bernoullipmf (1/2);
  xs ← bv n;
  returnpmf (x · xs)
}

```

The base case *bv 0* is defined as the distribution that assigns probability 1 to the empty list. In the step case, we draw  $x$  from the Bernoulli distribution and a sample  $xs$  from the distribution *bv n* and return  $x \cdot xs$ .

We further define the simple function *flip i b* that flips the  $i$ th bit of the bit vector  $b$ . When we apply *flip i* to every element of the probability distribution *bv n* we obtain again the same probability distribution:  $map_{pmf} (flip i) (bv n) = bv n$ .

## 3 List Update: Deterministic Algorithms

### 3.1 Online and Offline Algorithms

We need to define formally what online and offline algorithms are. Our formalization is similar to request-answer systems [6] but we clarify the role of the initial state and replace a history-based formalization with an equivalent state-based one. Everything is parameterized by a type of *requests*  $\mathcal{R}$ , a type of *answers*  $\mathcal{A}$ , a type of *states*  $\mathcal{S}$ , a type of *internal states*  $\mathcal{I}$ , and by the following three functions:  $step :: \mathcal{S} \rightarrow \mathcal{R} \rightarrow \mathcal{A} \rightarrow \mathcal{S}$ ,  $t :: \mathcal{S} \rightarrow \mathcal{R} \rightarrow \mathcal{A} \rightarrow \mathbb{N}$  and  $wf :: \mathcal{S} \rightarrow \mathcal{R} list \rightarrow bool$ . Answers describe how the system state changes in reaction to a request:  $step s r a$  is the new state after  $r$  has been answered by  $a$  and  $t s r a$  is the time or cost of that step. The predicate *wf* defines the well-formed request sequences depending on the initial state. An offline algorithm is a function of type  $\mathcal{S} \rightarrow \mathcal{R} list \rightarrow \mathcal{A} list$  that computes a list of answers from a start state and a list of requests. An online algorithm is a pair  $(\iota, \delta)$  of an initialization function  $\iota :: \mathcal{S} \rightarrow \mathcal{I}$  that yields the initial internal state, and a

transition function  $\delta :: \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{R} \rightarrow \mathcal{A} \times \mathcal{I}$  that yields the answer and the new internal state. In the sequel assume  $A = (\iota, \delta)$ .

Note that we separate the problem specific states  $\mathcal{S}$  and step function *step* from the algorithm specific internal states  $\mathcal{I}$  and transition function  $\delta$  to obtain a modular framework. Elements of type  $\mathcal{S} \times \mathcal{I}$  are called *configurations*.

In this context we define the following functions:

- *Step*  $A (s, i) r = (\text{let } (a, i') = \delta (s, i) r \text{ in } (\text{step } s r a, i'))$   
transforms one configuration into the next by composing  $\delta$  and *step*.
- $T s r s a s$  is the time it takes to process a request list  $rs$  and corresponding answer list  $as$  (of the same length) starting from state  $s$  via a sequence of *steps*.
- $OPT[s;rs] = \text{Inf } \{T s r s a s \mid |as| = |rs|\}$  where *Inf* is the infimum, is the time of the optimal offline algorithm servicing  $rs$  starting from state  $s$ . Note that the infimum is taken over the times of all answer lists with appropriate length.
- $A[s;rs]$  is the time an online algorithm  $A$  takes to process a request list  $rs$  via a sequence of *Steps* starting from configuration  $(s, \iota s)$ .
- Algorithm  $A$  is deemed  $c$ -competitive if its cost is at most  $c$  times  $OPT$ . Formally:  
 $\text{compet } A c S \iff (\forall s \in S. \exists b \geq 0. \forall rs. \text{wf } s r s \implies A[s;rs] \leq c * OPT[s;rs] + b)$   
It expresses that the online algorithm  $A$  is  $c$ -competitive on the set of initial states  $S$  and well-formed request sequences.

### 3.2 On/Offline Algorithms for List Update

The list update problem consists of maintaining an unsorted list of elements while the cost of servicing a sequence of requests has to be minimized. Each request asks to search an element sequentially from the front of the list. A penalty equal to the position of the requested element has to be paid. In order to minimize the cost of future requests the requested element can be moved further to the front of the list by a *free exchange*. Any other swap of two consecutive elements in the list costs one unit and is called a *paid exchange*.

We instantiate our generic model as follows. Given a type of elements  $\alpha$ , states are of type  $\alpha$  *list*, requests of type  $\alpha$ , and answers are of type  $\mathbb{N} \times \mathbb{N}$  *list*. An answer  $(n, [n_1, \dots, n_k])$  means that the requested element is moved  $n$  positions to the front at no cost (*free exchange*) after swapping the elements at index  $n_i$  and  $n_i + 1$  ( $i = k, \dots, 1$ ) at the cost of 1 per exchange (*paid exchanges*). Based on two functions  $\text{mtf}_2 :: \mathbb{N} \rightarrow \alpha \rightarrow \alpha \text{ list} \rightarrow \alpha \text{ list}$  and  $\text{swaps} :: \mathbb{N} \text{ list} \rightarrow \alpha \text{ list} \rightarrow \alpha \text{ list}$  we define  $\text{step } s r (k, ks) = \text{mtf}_2 k r (\text{swaps } ks s)$  and  $t s r (k, ks) = \text{index } (\text{swaps } ks s) r + 1 + |ks|$ . There is no need for paid exchanges after the move to front because they can be performed at the beginning of the next step. Corner cases:  $\text{mtf}_2 k x xs$  does nothing if  $x \notin \text{set } xs$  and moves  $x$  to the front if  $x \in \text{set } xs$  and  $\text{index } xs x < k$ ;  $\text{swaps } [n_1, \dots, n_k] xs$  ignores indices  $n_i$  such that  $|xs| \leq n_i + 1$ . We focus on the *static list model* by instantiating the well-formedness predicate *wf* by the predicate *static* defined by  $\text{static } s r s \iff \text{set } rs \subseteq \text{set } s$ .

Sleator and Tarjan [30], who introduced the list update problem, claimed (their Theorem 3) that offline algorithms do not need paid exchanges. Later Reingold and Westbrook [28] refuted this and proved the opposite: offline algorithms need only paid exchanges. This may also be considered as an argument in favour of verification.

### 3.3 Move to Front

The archetypal online algorithm is *move to front* (MTF): when an element is requested, it is moved to the front of the list, without any paid exchanges. MTF needs no internal state

and thus we identify  $\mathcal{I}$  with the unit type that contains only the dummy element  $()$ . The pair  $MTF = (\lambda \_ . (), \lambda (s, i) r. ((|s| - 1, []), ()))$  is an online algorithm in the sense of our above model.

Now we verify Sleator and Tarjan's result [30] that MTF is at most 2-competitive, i.e., at most twice as slow as any offline algorithm. We are given an initial state  $s$  of distinct elements, a request sequence  $rs$  and an answer sequence  $as$  computed by some offline algorithm such that  $|as| = |rs|$ . The state of MTF after servicing the requests  $rs_0, \dots, rs_{n-1}$  is denoted by  $s_{mtf} n$ , the cost of executing step  $n$  is denoted by  $t_{mtf} n$ . The state after answering the requests  $rs_0, \dots, rs_{n-1}$  with the answers  $as_0, \dots, as_{n-1}$  is denoted by  $s_{off} n$ , the cost  $t_{off} n$  of executing  $as_n$  is broken up as follows:  $c_{off} n$  is the cost of finding the requested element  $rs_n$  and  $p_{off} n$  ( $f_{off} n$ ) is the number of the paid (free) exchanges. Following [30] we define the potential as the number of inversions that separates MTF from the offline algorithm ( $\Phi n = |Inv(s_{off} n)(s_{mtf} n)|$ ) and prove the key lemma

► **Lemma 3.**  $t_{mtf} n + \Phi(n+1) - \Phi n \leq 2 * c_{off} n - 1 + p_{off} n - f_{off} n$

Its proof is a little bit tricky and requires a number of lemmas about inversions that formalize what is often given as a pictorial argument. By telescoping and defining  $T_{mtf} n = (\sum_{i < n} t_{mtf} i)$  we obtain Sleator and Tarjan's Theorem 1:

► **Theorem 4.**  $T_{mtf} n \leq (\sum_{i < n} 2 * c_{off} i + p_{off} i - f_{off} i) - n$  [[6, Theorem 1.1]]

It follows that  $T_{mtf} n \leq (2 - 1/|s|) * T_{off} n$ , where  $T_{off} n = (\sum_{i < n} t_{off} i)$ , provided  $s \neq []$  and  $\forall i < n. rs_i \in set s$ . By definition of  $OPT$  we obtain the following corollary [6]:

► **Corollary 5.**  $s \neq [] \wedge distinct s \wedge set rs \subseteq set s \implies MTF[s;rs] \leq (2 - 1/|s|) * OPT[s;rs]$

Because *compet* is defined relative to *wf* and we have instantiated *wf* with the static list model (which implies  $set rs \subseteq set s$ ), we obtain the following compact corollary:

► **Corollary 6.**  $compet MTF 2 \{s \mid distinct s\}$

The assumption  $s \neq []$  has disappeared because we no longer divide by  $|s|$ .

### 3.4 A Lower Bound

The following lower bound for the competitiveness of any online algorithm is due to Karp and Raghavan [18]:

► **Theorem 7.**  $compet A c \{xs \mid |xs| = l\} \wedge l \neq 0 \wedge 0 \leq c \implies 2 * l / (l + 1) \leq c$

The corresponding Theorem 1.2 in [6] is incorrect because it asserts that every online algorithm is  $c$ -competitive for some constant  $c$ , but this is not necessarily the case if the algorithm uses paid exchanges. In the proof it is implicitly assumed there are no paid exchanges when claiming "The total cost incurred by the online algorithm is clearly  $l * n$ ".

Our proof roughly follows the original sketch [18, p. 302]. Let  $A = (\iota, \delta)$  be an online algorithm. We define a cruel request sequence that always requests the last element in the state of  $A$ , given a start configuration and length:

$cruel A c 0 = []$

$cruel A (s, i) (n + 1) = last s \cdot cruel A (Step A (s, i) (last s)) n$

We also define a cruel offline adversary for  $A$  that first sorts the state in decreasing order of access frequency in the cruel sequence and does nothing afterwards:

```

adv A s rs =
(if rs = [] then []
 else let crs = cruel A (Step A (s, \s) (last s)) (|rs| - 1)
      in (0, sort_sws (\x. |rs| - 1 - count_list crs x) s) .
      replicate (|rs| - 1) (0, []))

```

For the first step `sort_sws` computes the necessary paid exchanges according to the frequency count computed by `count_list` from the cruel sequence `crs`; the remaining steps are do-nothing answers  $(0, [])$ .

For the analysis let  $A[s;n]$  (resp.  $C[s;n]$ ) be the time  $A$  (resp. the adversary  $adv A$ ) requires to answer the cruel request sequence of length  $n + 1$  starting in state  $s$ . Assume  $l \neq 0$ . First we prove  $C[s;n] \leq a + (l+1)*n \text{ div } 2$  where  $a = l^2 + l + 1$ . The cost of the first step of the cruel adversary (searching and sorting) is at most  $a$ , and the cost of searching for  $n$  requested items is at most  $(l + 1) * n \text{ div } 2$ . We obtain the latter bound by writing the cost as a sum of terms  $i * f_i$  where  $f_i$  is the number of requests of the  $i$ th item in the sorted list,  $i = 0, \dots, l-1$ . Because the  $f_i$  decrease with increasing  $i$ , the result follows by Tchebychef's inequality [11, 2.17] that the mean of the product is at most the product of the means. The cost  $A[s;n]$  is  $(n + 1) * l$  if there are no paid exchanges and thus  $(n+1)*l \leq A[s;n]$ . Combining this with the upper bound for  $C[s;n]$  we obtain  $2*l/(l+1) \leq A[s;n]/(C[s;n]-a)$  for all large enough  $n$ . From  $c$ -competitiveness of  $A$  we obtain a constant  $b$  such that  $(A[s;n]-b)/C[s;n] \leq c$ . The additive constants are typically (and incorrectly) ignored, in which case  $2*l/(l+1) \leq c$  is immediate; otherwise it takes a bit of limit reasoning.

## 4 List Update: Randomized Algorithms

### 4.1 Randomized Online Algorithms

Now we generalize our model of online algorithms of §3.1 to randomized online algorithms. We view a randomized algorithm not as a distribution of deterministic algorithms, but an algorithm working on a distribution of configurations. The monad described in §2.1 suggests this view and enables us to formulate randomized algorithms concisely. Furthermore we expect that proofs can be mechanized more easily that way.

The initialization function now not only yields one initial internal state but a distribution over the type of internal states:  $\mathcal{S} \rightarrow \mathcal{I}$  *pmf*. Similarly the transition function of randomized online algorithms has the type  $(\mathcal{S} \times \mathcal{I})$  *pmf*  $\rightarrow \mathcal{R} \rightarrow (\mathcal{A} \times \mathcal{I})$  *pmf*. We now generalize a number of functions from the deterministic to the randomized setting. We overload the names because the deterministic versions are special cases of the randomized ones. Whether  $A = (\iota, \delta)$  is a randomized or deterministic online algorithm will be clear from the context.

- A compound *Step* on configurations consists of two steps: first the online algorithm will produce a distribution of answer and new internal states, then the problem (*step*) will process the answer and yield a new configuration distribution:

```

Step A r (s, i) = do {
  (a, is') ← δ (s, i) r;
  return_pmf (step s r a, is')
}

```

- *config A s rs* formalizes the execution of  $A$  by denoting the distribution of configurations after servicing the request sequence  $rs$  starting in state  $s$ .

- $A[s;rs]$  denotes the expected time  $A$  takes to process a request list  $rs$  via a sequence of *Steps* starting from the distribution of configurations obtained by combining  $s$  with  $\iota s$ .
- $\text{compet } A \ c \ S \longleftrightarrow (\forall s \in S. \exists b \geq 0. \forall rs. \text{wf } s \ rs \longrightarrow A[s;rs] \leq c * \text{OPT}[s;rs] + b)$  expresses that  $A$  is  $c$ -competitive against an oblivious adversary on the set of initial states  $S$ .

Function  $\text{embed } (\iota, \delta) = (\lambda s. \text{return}_{\text{pmf}} (\iota s), \lambda s \ r. \text{return}_{\text{pmf}} (\delta \ s \ r))$  turns a deterministic into a randomized algorithm. It preserves the above notions. For example, for any deterministic algorithm  $A$  it holds that  $\text{compet } A \ c \ S_0 \longleftrightarrow \text{compet } (\text{embed } A) \ c \ S_0$ .

## 4.2 BIT

In this section we study a simple randomized algorithm for the list update problem called BIT due to Reingold and Westbrook [28]. BIT breaks the 2-competitive barrier for deterministic online algorithms (Theorem 7): we will prove that BIT is 1.75-competitive.

BIT keeps for every element  $x$  in the list a bit  $b(x)$ . The  $b(x)$  are initialized randomly, independently and uniformly. When some  $x$  is requested, its bit  $b(x)$  is complemented; then, if  $b(x)$ ,  $x$  is moved to the front. Formally, the internal state is a pair  $(b, s_0) :: \text{bool list} \times \alpha \ \text{list}$  where  $s_0$  is the initial list and  $|b| = |s_0|$ . The informal  $b(x)$  becomes  $b_{\text{index } s_0 \ x}$ .

- **Definition 8** (BIT).  $\text{BIT} = (\iota_{\text{BIT}}, \delta_{\text{BIT}})$  where  $\iota_{\text{BIT}} \ s_0 = \text{map}_{\text{pmf}} (\lambda b. (b, s_0)) \ (bv \ |s_0|)$   
 $\delta_{\text{BIT}} \ (s, b, s_0) \ x = \text{return}_{\text{pmf}} ((\text{if } b_{\text{index } s_0 \ x} \ \text{then } 0 \ \text{else } |s|, []), \text{flip } (\text{index } s_0 \ x) \ b, s_0)$

Function  $\iota_{\text{BIT}}$  generates a random bit vector (for  $bv$  see §2.1) of length  $|s_0|$  and pairs it with  $s_0$ . Function  $\delta_{\text{BIT}}$  is given a configuration  $(s, b, s_0)$  and a request  $x$ , *flips*  $x$ 's bit in  $b$ , and if it was set, the answer is move-to-front, otherwise it is do-nothing. BIT is a barely random algorithm: only the initialization function is randomized, the transition function is deterministic.

- **Theorem 9** ([6, Theorem 2.1]).  $\text{compet } \text{BIT} \ (7/4) \ \{\text{init} \mid \text{init} \neq [] \wedge \text{distinct } \text{init}\}$

The proof of this theorem is similar to the proof that MTF is 2-competitive: the potential function involves weighted inversions. Therefore we do not discuss the details (see [12]). We now introduce an alternative to the potential function method, which allows us to analyze BIT again and move on to more advanced algorithms.

## 4.3 List Factoring

The list factoring method enables us to reduce competitive analysis of list update algorithms to lists of size two. The main idea is to modify the cost measure. The cost of accessing some element will be the number of elements that precede it. We attribute a “blocking cost” of 1 to every element that precedes the requested element. For the requested element and all following the blocking cost is 0. In summary, the cost of accessing the  $i$ th item is no longer  $i + 1$  but  $i$ . This is called the *partial* cost model, in contrast to the *full* cost model. Costs in the partial cost model are marked with an asterisk; for example,  $t^* \ s \ r \ a = t \ s \ r \ a - 1$ . Upper bounds on the competitive ratio in the partial cost model are also upper bounds on the competitive ratio in the full cost model [6, Lemma 1.3].

Let  $A^*[s;rs](x;i)$  denote the expected blocking cost of element  $x$  in the  $i$ th step of the execution of algorithm  $A$  on the request sequence  $rs$  starting from state  $s$ . The notations  $\sum_{x \in M} m_x$  and  $\sum_{x \mid P_x} m_x$  denote summations over a set or restricted by a predicate. We will need a set of all pairs  $(x, y) \in M \times M$  of distinct elements where only one of the two pairs  $(x, y)$  and  $(y, x)$  is included. For simplicity we assume  $M$  is linearly ordered and define  $\text{Diff}_2 \ M = \{(x, y) \mid x \in M \wedge y \in M \wedge x < y\}$ .

Now consider the cost incurred by an online algorithm without paid exchanges:

$$\begin{aligned}
 A^*[s;rs] &= \sum_{i < |rs|} \sum_{x \in \text{set } s} A^*[s;rs](x;i) \\
 &= \sum_{x \in \text{set } s} \sum_{i < |rs|} A^*[s;rs](x;i) \\
 &= \sum_{x \in \text{set } s} \sum_{y \in \text{set } s} \sum_i i \mid i < |rs| \wedge rs_i = y. A^*[s;rs](x;i) \\
 &= \sum_{(x,y) \in \{(x,y) \mid x \in \text{set } s \wedge y \in \text{set } s \wedge x \neq y\}} \\
 &\quad \sum_i i \mid i < |rs| \wedge rs_i = y. A^*[s;rs](x;i) \\
 &= \sum_{(x,y) \in \text{Diff}_2(\text{set } s)} \\
 &\quad \sum_i i \mid i < |rs| \wedge (rs_i = y \vee rs_i = x). \\
 &\quad A^*[s;rs](y;i) + A^*[s;rs](x;i)
 \end{aligned}$$

The inner summation of the last expression is abbreviated by  $A_{xy}^*[s;rs]$  and interpreted as the expected cost generated by  $x$  blocking  $y$  or vice versa. We can condense the above derivation:

► **Lemma 10** ([6, Equation (1.4)]).  $A^*[s;rs] = (\sum_{(x,y) \in \text{Diff}_2(\text{set } s)} A_{xy}^*[s;rs])$

As the value of any summand on the right hand side only depends on the relative order of  $x$  and  $y$  during the execution and the relative order may only change when  $x$  or  $y$  are requested (as we disallowed paid exchanges for now) one might think that this is exactly the same as the cost incurred by the algorithm when run on the projected request list  $rs_{xy}$  starting from the projected initial state  $s_{xy}$ . While this is not the case in general, this equality yields a good characterization of a subset of all list update algorithms and is thus referred to as the *pairwise property*. Most of the list update algorithms studied in the literature share this property, including MTF, BIT, TS and COMB (see Table 1 in [19] where “projective” means “pairwise”).

► **Definition 11** (pairwise property). Algorithm  $A$  satisfies the *pairwise property* if  $\text{distinct } s \wedge \text{set } rs \subseteq \text{set } s \wedge (x,y) \in \text{Diff}_2(\text{set } s) \implies A^*[s_{xy};rs_{xy}] = A_{xy}^*[s;rs]$

With a similar development as for Lemma 10 we can split the costs of  $OPT^*$  into the costs that are incurred by each pair of elements: first the costs incurred by blocking each other and second the number of paid exchanges that change the elements’ relative order:

► **Theorem 12** ([6, Equation (1.8)]).  $OPT^*[s;rs] = (\sum_{(x,y) \in \text{Diff}_2(\text{set } s)} OPT_{xy}^*[s;rs] + OPT_{P;xy}^*[s;rs])$

If we consider the summand for a specific pair  $(x,y)$ , we see that it gives rise to an (not necessarily optimal) algorithm servicing the projected request sequence  $rs_{xy}$ . Thus this term is an upper bound for the optimal cost for servicing  $rs_{xy}$ . This fact is established by constructing this projected algorithm and showing that its cost is equal to the right-hand side of the inequality:

► **Lemma 13** ([6, Equation (1.7)]).  $OPT^*[s_{xy};rs_{xy}] \leq OPT_{xy}^*[s;rs] + OPT_{P;xy}^*[s;rs]$

Now we are in the position to describe the *list factoring technique*:

► **Theorem 14** ([6, Lemma 1.2]). *Let  $A$  be an algorithm that does not use paid exchanges, satisfies the pairwise property and is  $c$ -competitive for lists of length 2. Then  $A$  is  $c$ -competitive for arbitrary lists.*

$$\begin{array}{ll}
\text{Proof. } A^*[s;rs] & \stackrel{\text{Lemma 10}}{=} \sum_{(x,y) \in \text{Diff}_2(\text{set } s)} A_{xy}^*[s;rs] \\
& \stackrel{\text{pairwise}}{=} \sum_{(x,y) \in \text{Diff}_2(\text{set } s)} A^*[s_{xy};rs_{xy}] \\
& \stackrel{c\text{-compet.}}{\leq} \sum_{(x,y) \in \text{Diff}_2(\text{set } s)} c * OPT^*[s_{xy};rs_{xy}] + b \\
& \stackrel{\text{Lemma 13}}{\leq} c * \\
& \quad \left( \sum_{(x,y) \in \text{Diff}_2(\text{set } s)} OPT_{xy}^*[s;rs] + OPT_{P,xy}^*[s;rs] \right) + \\
& \quad b * (|s| * (|s| - 1)) / 2 \\
& \stackrel{\text{Lemma 12}}{=} c * OPT^*[s;rs] + b * (|s| * (|s| - 1)) / 2 \quad \blacktriangleleft
\end{array}$$

For showing that algorithm  $A$  has the pairwise property it suffices to show that the probability that  $x$  precedes  $y$  during the service of the projected request sequence is equal to the probability when servicing the original request sequence.

► **Lemma 15** ([6,  $\Rightarrow$  of Lemma 1.1]). *For an online algorithm  $A$  without paid exchanges, let  $s_{xy}^{A;rs_{xy}}$  and  $s^{A;rs}$  denote the configuration distribution after servicing the projected respectively full request list  $rs$  starting from  $s$ . Then*  
 $\text{map}_{\text{pmf}}(\lambda(s, is). x < y \text{ in } s) s_{xy}^{A;rs_{xy}} = \text{map}_{\text{pmf}}(\lambda(s, is). x < y \text{ in } s) s^{A;rs} \implies \text{pairwise } A$

#### 4.4 $OPT_2$ : an Optimal Algorithm for Lists of Length 2

We formalize  $OPT_2$ , an optimal offline algorithm for lists of length 2 due to Reingold and Westbrook [29], verify its optimality, and determine the cost of  $OPT_2$  on different specific request sequences. The informal definition of  $OPT_2$  is as follows [29]:

► **Definition 16** ( $OPT_2$  informally). After each request, move the requested item to the front via free swaps if the next request is also to that item. Otherwise do nothing.

Observe that this algorithm only needs knowledge of the current and next request. Thus it is almost an online algorithm, except that it needs a lookahead of 1.

Function  $OPT_2 rs [x, y]$  that takes a request sequence  $rs$  and a state  $[x, y]$  and returns an answer sequence is defined easily by recursion on  $rs$ .

► **Theorem 17** (Optimality of  $OPT_2$ ).

$$\text{set } rs \subseteq \{x, y\} \wedge x \neq y \implies T^*[x, y] rs (OPT_2 rs [x, y]) = OPT^*[[x, y]; rs]$$

The proof is by induction on  $rs$  followed by a “simple case analysis” [29] the formalization of which is quite lengthy.

In an execution of  $OPT_2$ , after two consecutive requests to the same element that element will be at the front. This enables us to partition the request sequence into phases and restart  $OPT_2$  after each phase:

► **Lemma 18.** If  $x \neq y \wedge s \in \{[x, y], [y, x]\} \wedge \text{set } us \subseteq \{x, y\} \wedge \text{set } vs \subseteq \{x, y\}$  then  $OPT_2 (us @ [x, x] @ vs) s = OPT_2 (us @ [x, x]) s @ OPT_2 vs [x, y]$ .

Thus we can partition a request sequence into phases ending with two consecutive requests to the same element and we know the state of  $OPT_2$  at the end of each phase. Such a phase can have one of four forms, for any of these we have “calculated” the cost of  $OPT_2$  (see Table 1). This involves inductive proofs in the case of the Kleene star.

In the following two subsections the *phase partitioning* technique is described in more detail and is then used to prove that BIT is 7/4-competitive.



## 4.5 Phase Partitioning

In the following we will partition all request sequences into *complete phases* that end with two consecutive requests to the same element and possibly a trailing *incomplete phase*. The set of all request sequences can be described by  $(x+y)^*$ , the complete phases by  $\varphi_{xy} = (x^2(yx)^*yy+y^2(xy)^*xx)$  and the incomplete phases (that do not contain two consecutive occurrences of the same element) by  $\bar{\varphi}_{xy} = (x^2(yx)^*y+y^2(xy)^*x)$ . The regular expression  $r^?$  is short for  $r + \varepsilon$ . In order to prove identities like  $\mathcal{L}(\varphi_{xy}^* \bar{\varphi}_{xy}^?) = \mathcal{L}((x+y)^*)$  we use a regular expression equivalence checker available in Isabelle/HOL [21], which we extend to regular expressions with variables. This prevents us from overlooking corner cases, such as the missing D case in Table 1 (see the end of 4.6).

We now want to compare costs of an online algorithm  $A$  and  $OPT_2$  on a complete phase  $rs$  and lift results to arbitrary request sequences  $\sigma$  containing two different elements. This lifting requires us to show (by an invariant proof), that at the end of each complete phase  $A$  and  $OPT_2$  are in sync again.

Recall that  $OPT_2$  will have element  $rs_{|rs|-1}$  in front of the state after servicing  $rs$ . Let  $S^A$  be a configuration distribution of  $A$ , then  $S^{A;rs}$  denotes the configuration distribution after the service of  $rs$  by  $A$  starting from  $S^A$  and  $A^*[S^A;rs]$  denotes its cost. Let  $inv_A S^A x s$  be a predicate on a configuration distribution of  $A$ , a request and a state. Suppose for any  $S^A$ ,  $x$  and  $s$  that (i)  $inv_A S^A x s \implies A^*[S^A;rs] \leq c * T^*[x, y] rs$  ( $OPT_2 rs [x, y]$ ) and (ii)  $inv_A S^A x s \implies inv_A S^{A;rs} (rs_{|rs|-1}) s$ . Then we can conclude  $A^*[S^A;\sigma] \leq c * T^*[x, y] \sigma$  ( $OPT_2 \sigma [x, y]$ ) +  $c$  if the predicate  $inv_A S^A x [x, y]$  holds initially.

This fact follows by well-founded induction on the length of  $\sigma$ . If we additionally verify that the invariant  $inv_A S^A x [x, y]$  holds for  $S^A$  being the configuration distribution after initializing algorithm  $A$  from state  $[x, y]$  we can finally conclude  $A^*[[x, y], \sigma] \leq c * T^*[x, y] \sigma$  ( $OPT_2 \sigma [x, y]$ ) +  $c = c * OPT^*[[x, y];\sigma] + c$ .

Note that  $inv_A S^A x s$  must imply that all states in the configuration distribution  $S^A$  have  $x$  in front. If this is not the case and the state  $[y, x]$  has nonzero probability, for the complete phase  $rs = [x, x]$ ,  $A$  has nonzero costs whereas  $OPT_2$  pays nothing. This makes showing property (i) impossible. Consequently not all pairwise algorithms can be analyzed with this technique (e.g. RMTF [6]).

To further facilitate the analysis, all complete phases can be classified into four forms, described by the regular expressions found in the first column of Table 1. Together with the list factoring technique, the proof of an algorithm being  $c$ -competitive is reduced to determining the costs on request sequences of these forms. This kind of analysis is conducted in the next section for BIT.

## 4.6 Analysis of BIT

We now show that BIT is 7/4-competitive using both the list factoring method and the phase partitioning technique.

With the help of Lemma 15 we proved that BIT has the pairwise property. The result can be established by induction on the original request sequence and a case distinction whether the requested element is one of  $x$  and  $y$  or not, the rest is laborious bookkeeping. We refer the reader to [12] for the details.

► **Lemma 19.** *pairwise BIT*

Let us turn to showing that BIT is 7/4-competitive on lists of length 2. To that end we analyze BIT on the different forms of complete phases as explained above. At the end

■ **Table 1** Costs of BIT,  $OPT_2$  and TS for request sequence of the four phase forms;  $x$  is the first element in the state;  $k$  is the number of iterations of the Kleene star expression.

	$\sigma$	$BIT$	$OPT_2$	$TS$
A	$x^?yy$	1.5	1	2
B	$x^?yx(yx)^*yy$	$1.5 * (k + 1) + 1$	$(k + 1) + 1$	$2 * (k + 1)$
C	$x^?yx(yx)^*x$	$1.5 * (k + 1) + 0.25$	$(k + 1)$	$2 * (k + 1) - 1$
D	$xx$	0	0	0

of each complete phase, BIT will have the last request in front of the state (because the element was requested twice and one of the two requests moved it). BIT and  $OPT_2$  thus are synchronized before and after each phase. BIT's invariant  $inv_{BIT} S x s$  (in the sense of the previous subsection) is defined as saying that in every configuration in the distribution  $S$ , element  $x$  is in front of the state and the second component of the internal state is  $s$ .

Table 1 shows the costs of BIT for the four respective forms. We now verify both the preservation of  $inv_{BIT}$  and the cost incurred by BIT for a phase  $rs$  of form B, i.e.,  $rs \in \mathcal{L}(x^?yx(yx)^*yy)$ .

We start with the configuration distribution  $S$  satisfying  $inv_{BIT} S x s$  (see above). First we observe that serving an optional request  $x$  does not alter the configuration distribution nor add any cost. Now serving the request  $y$  moves  $y$  to the front for two out of four configurations and in every case adds cost 1. In one of the former two configurations the next request to element  $x$  brings  $x$  to the front again. Consequently the state after serving  $yx$  is  $[y, x]$  iff  $y$ 's bit is set and  $x$ 's bit is not set. This distribution of configurations is preserved by any number of further requests  $yx$ . Serving the first request to  $yx$  costs  $1 + 1/2$  as well as any further request to  $yx$  has cost  $3/4 + 3/4$ . Let  $\sigma$  be the part of  $rs$  with the Kleene star, by induction on the Kleene star in  $\sigma$  the cost for serving  $\sigma$  is  $3/4 * |\sigma|$ . The trailing request to  $yy$  then costs  $3/4 + 1/4$  and  $y$  is moved to the front in all configurations. Thus finally the invariant  $inv_{BIT} S' y s$  is satisfied, where  $S'$  is the configuration distribution after serving  $rs$ . Note that here the order of  $y$  and  $x$  have changed. In the analysis of the next phase,  $x$  and  $y$  take the swapped positions. But as they are interchangeable in all the theorems this does no harm.

Determining the costs in Table 1 is usually presented in the style of the last paragraph ([6, §2.4] and [2, Lemma 3]). While these calculations are tedious for humans, the proof assistant is able to carry them out almost automatically: with the help of some lemmas about how BIT transforms the configuration distribution on single requests, the costs of complete phases can be calculated and proved mechanically.

Note that in contrast to Table 1.1 in [6, §1.6.1] we define the phase forms differently. They allow more than one initial  $x$  in forms A, B and C, we only allow zero or one. Our forms precisely capture the idea of splitting the request sequence into phases that end with two consecutive requests to the same element. Moreover, form D is missing from their table.

The results for the other phase forms follows in a similar way and finally we prove  $BIT^*[[x, y];\sigma] \leq 7/4 * OPT^*[[x, y];\sigma] + 7/4$ . Together with the pairwise property of BIT and the list factoring method we can lift the result to lists of arbitrary length and obtain another proof of Theorem 9.

Actually the ratio between costs of BIT and  $OPT_2$  tends to  $3/2$  for long phases, only the poor performance of short phases of form C leads to the competitive ratio of  $7/4$ . This observation does not follow from the combinatorial proof of Section 4.2.

## 4.7 TS to the Rescue

The deterministic online algorithm TS due to Albers [1] performs well in the cases where BIT performs badly. We now present the analysis of TS and in the following subsection will show how the two algorithms can be combined. TS does the following:

► **Definition 20** (TS informally). After each request, the accessed item  $x$  is inserted immediately in front of the first item  $y$  that precedes  $x$  in the list and was requested at most once since the last request to  $x$ . If there is no such item  $y$  or if  $x$  is requested for the first time, then the position of  $x$  remains unchanged.

This algorithm has an internal state of type  $\alpha$  *list*, the history of requests already processed. The transition function  $\delta_{TS}$  is formalized as follows:

► **Definition 21.**

$$\delta_{TS}(s, is) r = \begin{aligned} & \text{(let } V_r = \{x \mid x < r \text{ in } s \wedge \text{count\_list}(\text{take}(\text{index } is \ r) \ is) \ x \leq 1\} \\ & \text{in ((if index } is \ r < |is| \wedge V_r \neq \emptyset \\ & \quad \text{then index } s \ r - \text{Min}(\text{index } s \ ' \ V_r) \ \text{else } 0, \\ & \quad \square), \\ & \quad r \cdot is)) \end{aligned}$$

Note that *take n xs* returns the length  $n$  prefix of  $xs$ ; because the history is stored in reverse order, *take (index is r) is* is the part of the history since the last request to element  $r$ .

For the analysis of TS we employ the proof methods developed in the preceding subsections. We first examine the costs of the four phase forms by simulation and induction. Then, by the phase partitioning method, we extend the result to any request sequence. The invariant needed for TS essentially says  $is = \square \vee (\exists x s' hs. s = x \cdot s' \wedge is = x \cdot x \cdot hs)$ : either TS has just been initialized or the last two requests were to the first element in the state. Intuitively this invariant implies that for the next request to  $y$ , the element would not be moved to the front of  $x$ . The last column of Table 1 shows the costs of TS for the four respective phase forms. TS performs better than BIT for short phases of forms B and C.

To lift this result to arbitrary initial lists, it remains to show that TS satisfies the pairwise property. With Lemma 15 and because we are in the deterministic domain it suffices to show that the relative order of  $x$  and  $y$  is equal both in the service of the projected as well as the original request sequence. This fact is not as obvious as for MTF or BIT; for showing this equality at any point in time during the service of TS, we do a case distinction on the history and look at most at the last three accesses to  $x$  and  $y$ . For most cases it is quite easy to determine the current relative order both in the projected as well as the full request sequence. For example, after two requests to the same element  $x$ ,  $x$  must be before  $y$  in both cases. The only tricky case is when the last requests were  $xy$ : then a proof involving infinite descent is used along the lines of Lemma 2 in [2].

Finally we obtain the fact *pairwise TS* and hence

► **Theorem 22** ([6, Theorem 1.4]). *compet TS 2 {s | distinct s  $\wedge$  s  $\neq$   $\square$ }*

## 4.8 COMB

Our development finally peaks in the formalization of the  $8/5$ -competitive online algorithm COMB due to Albers *et al.* [2] that chooses with probability  $4/5$  between executing BIT and TS. COMB's internal state type is the sum type of the internal state types of BIT and TS, function  $\iota_{COMB}$  initializes like BIT and TS with respective probabilities and function  $\delta_{COMB}$

applies  $\delta_{BIT}$  or  $\delta_{TS}$  depending on the type of the internal state it receives. As COMB is a combination of BIT and TS, several properties carry over directly: COMB is barely random, does not use paid exchanges and *pairwise COMB* holds.

Table 1 shows that BIT outperforms TS for long phases. TS is cheaper only for short phases of forms B and C. The combination of the two algorithms yields an improved competitive ratio of  $8/5$ . The result is established by analyzing the combined cost for the different phase forms and then use the phase partitioning and list factoring method. This does not involve any combinatorial tricks, but only combining certain lemmas about BIT and TS.

► **Theorem 23** ([6, Theorem 2.2]). *compet COMB*  $(8/5) \{x \mid \text{distinct } x \wedge x \neq []\}$

It can also be shown (we did not verify this) that the probability for choosing between BIT and TS is optimal and that COMB attains the best competitive ratio possible for pairwise algorithms in the partial cost model [4].

## 5 Conclusion

This paper has demonstrated that state of the art randomized list update algorithms can be analyzed with a theorem prover. In the process we found mistakes and omissions in the published literature (for example, Theorem 7 and Table 1).

The field of programming languages is full of verified material (e.g., [25, 27]) which has lead to achievements like Leroy’s verified C compiler [22]. We believe that eventually both functional correctness and performance of critical software components will be verified. Such verifications will require verified algorithm analyses such as presented in this paper.

---

## References

- 1 Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.*, 27(3):682–693, 1998.
- 2 Susanne Albers, Bernhard von Stengel, and Ralph Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Inf. Process. Lett.*, 56(3):135–139, 1995.
- 3 Christoph Ambühl. Offline list update is NP-hard. In Mike Paterson, editor, *Algorithms – ESA 2000*, volume 1879 of *LNCS*, pages 42–51. Springer, 2000.
- 4 Christoph Ambühl, Bernd Gärtner, and Bernhard von Stengel. Optimal projective algorithms for the list update problem. In U. Montanari, J. Rolim, and E. Welzl, editors, *Automata, Languages and Programming (ICALP 2000)*, volume 1853 of *LNCS*, pages 305–316. Springer, 2000.
- 5 Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, and Santiago Zanella Béguelin. Computer-aided cryptographic proofs. In L. Beringer and A. Felty, editors, *Interactive Theorem Proving (ITP 2012)*, volume 7406 of *LNCS*, pages 11–27. Springer, 2012.
- 6 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- 7 Arthur Charguéraud and François Pottier. Machine-checked verification of the correctness and amortized complexity of an efficient union-find implementation. In C. Urban and X. Zhang, editors, *Interactive Theorem Proving (ITP 2015)*, volume 9236 of *LNCS*, pages 137–153. Springer, 2015.

- 8 Martin Erwig and Steve Kollmansberger. Functional pearls: Probabilistic functional programming in Haskell. *Journal of Functional Programming*, 16(1):21–34, 2006.
- 9 Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O’Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the odd order theorem. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving (ITP 2013)*, volume 7998 of *LNCS*, pages 163–179. Springer, 2013.
- 10 Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the Kepler conjecture. *CoRR*, abs/1501.02155, 2015.
- 11 G. Hardy, J.E. Littelwood, and G. Pólya. *Inequalities*. Cambridge University Press, 1934.
- 12 Maximilian P.L. Haslbeck and Tobias Nipkow. Analysis of list update algorithms. *Archive of Formal Proofs*, 2016. [http://isa-afp.org/entries/List\\_Update.shtml](http://isa-afp.org/entries/List_Update.shtml), Formal proof development.
- 13 Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate amortized resource analysis. *ACM Trans. Program. Lang. Syst.*, 34(3):14, 2012.
- 14 Johannes Hölzl, Andreas Lochbihler, and Dmitriy Traytel. A formalized hierarchy of probabilistic system types. In C. Urban and X. Zhang, editors, *Interactive Theorem Proving (ITP 2015)*, volume 9236 of *LNCS*, pages 203–220. Springer, 2015.
- 15 Joe Hurd. *Formal Verification of Probabilistic Algorithms*. PhD thesis, University of Cambridge, 2002.
- 16 Joe Hurd. Verification of the Miller-Rabin probabilistic primality test. *J. Logic Algebraic Programming*, 56(1-2):3–21, 2003.
- 17 Joe Hurd, Annabelle McIver, and Carroll Morgan. Probabilistic guarded commands mechanized in HOL. *Theoretical Computer Science*, 346(1):96–112, 2005.
- 18 Sandy Irani. Two results on the list update problem. *Inf. Process. Lett.*, 38(6):301–306, 1991.
- 19 Shahin Kamali and Alejandro López-Ortiz. A survey of algorithms and models for list update. In A. Brodnik, A. López-Ortiz, V. Raman, and A. Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms*, volume 8066 of *LNCS*, pages 251–266. Springer, 2013.
- 20 Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: Formal verification of an OS kernel. In Jeanna Neeffe Matthews and Thomas E. Anderson, editors, *Proc. 22nd ACM Symposium on Operating Systems Principles 2009*, pages 207–220. ACM, 2009.
- 21 Alexander Krauss and Tobias Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *J. Automated Reasoning*, 49:95–106, 2012. Published online March 2011.
- 22 Xavier Leroy. A formally verified compiler back-end. *J. Automated Reasoning*, 43:363–446, 2009.
- 23 Alejandro López-Ortiz, Marc P. Renault, and Adi Rosén. Paid exchanges are worth the price. In E. W. Mayr and N. Ollinger, editors, *Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *LIPICs*, pages 636–648. Schloss Dagstuhl, 2015.
- 24 Tobias Nipkow. Amortized complexity verified. In C. Urban and X. Zhang, editors, *Interactive Theorem Proving (ITP 2015)*, volume 9236 of *LNCS*, pages 310–324. Springer, 2015.

- 25 Tobias Nipkow and Gerwin Klein. *Concrete Semantics with Isabelle/HOL*. Springer, 2014. <http://concrete- semantics.org>.
- 26 Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
- 27 Benjamin C. Pierce and Stephanie Weirich, editors. *Special Issue: The POPLMARK Challenge*, volume 49(3) of *J. Automated Reasoning*. 2012.
- 28 Nick Reingold and Jeffery Westbrook. Optimum off-line algorithms for the list update problem. Technical Report 805, Yale University, Department of Computer Science, 1990.
- 29 Nick Reingold and Jeffery Westbrook. Off-line algorithms for the list update problem. *Inf. Process. Lett.*, 60(2):75–80, 1996.
- 30 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, 1985.





# Tunable Online MUS/MSS Enumeration

Jaroslav Bendík<sup>1</sup>, Nikola Beneš<sup>2</sup>, Ivana Černá<sup>3</sup>, and Jiří Barnat<sup>4</sup>

- 1 Faculty of Informatics, Masaryk University, Brno, Czech Republic  
xbendik@fi.muni.cz
- 2 Faculty of Informatics, Masaryk University, Brno, Czech Republic  
xbenes3@fi.muni.cz
- 3 Faculty of Informatics, Masaryk University, Brno, Czech Republic  
cerna@fi.muni.cz
- 4 Faculty of Informatics, Masaryk University, Brno, Czech Republic  
barnat@fi.muni.cz

---

## Abstract

In various areas of computer science, the problem of dealing with a set of constraints arises. If the set of constraints is unsatisfiable, one may ask for a minimal description of the reason for this unsatisfiability. Minimal unsatisfiable subsets (MUSes) and maximal satisfiable subsets (MSSes) are two kinds of such minimal descriptions. The goal of this work is the enumeration of MUSes and MSSes for a given constraint system. As such full enumeration may be intractable in general, we focus on building an online algorithm, which produces MUSes/MSSes in an on-the-fly manner as soon as they are discovered. The problem has been studied before even in its online version. However, our algorithm uses a novel approach that is able to outperform the current state-of-the-art algorithms for online MUS/MSS enumeration. Moreover, the performance of our algorithm can be adjusted using tunable parameters. We evaluate the algorithm on a set of benchmarks.

**1998 ACM Subject Classification** F.4.1 Logic and constraint programming

**Keywords and phrases** Minimal unsatisfiable subsets, Maximal satisfiable subsets, Unsatisfiability analysis, Infeasibility analysis

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2016.50

## 1 Introduction

In various areas of computer science, such as constraint processing, requirements analysis, and model checking, the following problem often arises. We are given a set of constraints and are asked whether the set of constraints is feasible, i.e. whether all the constraints are satisfiable together. In requirements analysis, the constraints represent the requirements on a given system, usually described as formulae of a suitable logic, and the feasibility question is in fact the question whether all the requirements can actually be implemented at once. In some model checking systems, such as those using the counterexample-guided abstraction refinement (CEGAR) workflow, an infeasible constraint system may arise as a result of the abstraction's overapproximation. In such cases where the set of constraints is infeasible, we might want to explore the reasons of infeasibility. There are basically two approaches that can be used here. One is to try to extract a single piece of information explaining the infeasibility, such as a minimal unsatisfiable subset (MUS) or dually a maximal satisfiable subset (MSS) of the constraints. The other option is to try to enumerate all, or at least as many as possible, of these sets. In this work, we focus on the second approach. Enumerating multiple MUSes is sometimes desirable: in requirements analysis, this gives better insight



© Jaroslav Bendík, Nikola Beneš, Ivana Černá, and Jiří Barnat;  
licensed under Creative Commons License CC-BY

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).

Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 50; pp. 50:1–50:13



Leibniz International Proceedings in Informatics

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

into the inconsistencies among requirements; in CEGAR-based model checking more MUSes lead to a better refinement that can reduce the complexity of the whole procedure [1].

The enumeration of all MUSes or MSSes is generally intractable due to the potentially exponential number of results. It thus makes sense to study algorithms that are able to provide at least some of those within a given time limit. An even better option is to have an algorithm that produces MUSes or MSSes in an on-the-fly manner as soon as they are discovered. It is the goal of this paper to describe such an algorithm.

## 1.1 Related Work

The list of existing work that focuses on enumerating multiple MUSes is short as most of the related work only deals with an extraction of a single MUS or even a non-minimal unsatisfiable subset. For example all of [6, 17, 19] use information from a satisfiability solver to obtain an unsatisfiable subset but they do not guarantee its minimality. Moreover, the majority of the algorithms which enumerate all MUSes have been developed for specific constraint domains, mainly for Boolean satisfiability problems.

**Explicit Checking.** The first algorithm for enumerating all MUSes we are aware of was developed by Hou [10] in the field of diagnosis and is built on explicit enumeration of every subset of the unsatisfiable constraint system. It checks every subset for satisfiability, starting from the complete constraint set and branching in a tree-like structure. The authors presented some pruning rules to skip irrelevant branches and avoid unnecessary work. Further improvements to this approach were made by Han and Lee [9] and by de la Banda et. al. [7].

**CAMUS.** A state-of-the-art algorithm for enumerating all MUSes called CAMUS by Liffiton and Sakallah [15] is based on the relationship between MUSes and the so-called minimal correction sets (MCSes), which was independently pointed out by [2, 5, 13]. This relationship states that  $M \subseteq C$  is a MUS of  $C$  if and only if it is an irreducible hitting set of  $\text{MCS}(C)$ . CAMUS works in two phases, first it computes all MCSes of the given constraint set, and then it finds all MUSes by computing all the irreducible hitting sets of these MCSes.

A significant shortcoming of CAMUS is that the first phase can be intractable as the number of MCSes may be exponential in the size of the instance and all MCSes must be enumerated before any MUS can be produced. This makes CAMUS unsuitable for many applications which require only a few MUSes but want to get them quickly. Note that CAMUS is able to enumerate MSSes, as they are simply the complements of MCSes.

**MARCO.** The desire to enumerate at least some MUSes even in the generally intractable cases led to the development of two independent but nearly identical algorithms: MARCO [12] and eMUS [18]. Both algorithms were later joined and presented in [14] under the name of MARCO. MARCO is able to produce individual MUSes during its execution and it does it in a relatively steady rate. To obtain each single MUS, MARCO first finds a subset  $U$  whose satisfiability is not known yet, checks it for satisfiability and if it is unsatisfiable, it is “shrunk” to a MUS. In the case that  $U$  is satisfiable, it is in a dual manner expanded into an MSS. The algorithm can be supplied with any appropriate shrink and expansion procedures; this makes MARCO applicable to any constraint satisfaction domain in general.

CAMUS and MARCO were experimentally compared in [14] and the former has shown to be faster in enumerating all MUSes in the tractable cases. However, in the intractable cases, MARCO was able to provide at least some MUSes while CAMUS often provided none. One another algorithm, the Dualize and Advance (DAA) by Bailey and Stuckey [2] was also

evaluated in these experiments. DAA is also based on the relationship between MCSes and MUSes and can produce both MUSes and MSSes during its execution; however, it has shown to be substantially slower than CAMUS in the case of complete MUSes enumeration and also slower than MARCO in the partial enumeration.

## 1.2 Our Contribution

In this paper, we present our own algorithm for online enumeration of MUSes and MSSes in general constraint satisfaction domains, dubbed TOME (**T**unable **O**nline **MUS/MSS** **E**numeration), that is able to outperform the current state-of-the-art MARCO algorithm. The core of TOME is based on a novel concept of *local* MUSes/MSSes. To find these we use a binary-search-based approach. Similarly to MARCO, TOME is able to directly employ arbitrary shrinking and expanding procedures. Moreover, TOME contains certain parameters that govern in which cases the shrinking and expanding procedures are to be used. We evaluate TOME on a variety of benchmarks that show that it indeed outperforms MARCO.

This paper builds on our previous work [3] where we focused on finding boundary elements in partially ordered sets represented by explicit acyclic graphs. Here we focus on the specific case of powersets represented symbolically. Another difference is that we perform online enumeration here.

Note that there is a constraint solving approach QuickXplain [11] which uses binary search, however it solves a different problem. It uses a linear priority ordering on constraints and extracts a single maximal consistent subset w.r.t. this priority.

**Outline of The Paper.** In Section 2 we state the problem we are solving in a formal way, defining all the necessary notions. In Section 3 we describe TOME in an incremental way, starting with the basic schema of MUS/MSS computation and gradually explaining the main ideas of our algorithm. Section 4 provides an experimental evaluation on a variety of benchmarks, comparing TOME against MARCO. The paper is concluded in Section 5.

## 2 Preliminaries

Our goal is to deal with arbitrary constraint satisfaction systems. The input is given as a finite set of constraints  $C = \{c_1, c_2, \dots, c_n\}$  with the property that each subset of  $C$  is either *satisfiable* or *unsatisfiable*. The definition of satisfiability may vary in different constraint domains, we only assume that if  $X \subseteq C$  is satisfiable, then all subsets of  $X$  are also satisfiable. The subsets of interest are defined in the following.

► **Definition 1** (MSS, MUS). Let  $C$  be a finite set of constraints and let  $N \subseteq C$ .  $N$  is a *maximal satisfiable subset* (MSS) of  $C$  if  $N$  is satisfiable and  $\forall c \in C \setminus N : N \cup \{c\}$  is unsatisfiable.  $N$  is a *minimal unsatisfiable subset* (MUS) of  $C$  if  $N$  is unsatisfiable and  $\forall c \in N : N \setminus \{c\}$  is satisfiable.

Note that the maximality concept used here is set maximality, not maximum cardinality as in the MaxSAT problem. This means there can be multiple MSSes with different cardinality. We use  $MUS(C)$  and  $MSS(C)$  to denote the set of all MUSes and MSSes of  $C$ , respectively. The formulation of our problem is the following: Given a finite set of constraints  $C$ , enumerate (all or at least as many as possible) members of  $MUS(C)$  and  $MSS(C)$ .

To describe the ideas of TOME and illustrate its usage, we shall use Boolean satisfiability constraints in the following. In the examples, each of the constraints  $c_i$  is going to be a clause

(a disjunction of literals). The whole set of constraints can be then seen as a Boolean formula in conjunctive normal form.

► **Example 2.** We illustrate the concepts on a small example. Assume that we are given a set  $C$  of four Boolean satisfiability constraints  $c_1 = a$ ,  $c_2 = \neg a$ ,  $c_3 = b$ , and  $c_4 = \neg a \vee \neg b$ . Clearly, the whole set is unsatisfiable as the first two constraints are negations of each other. There are two MUSes:  $\{c_1, c_2\}$ ,  $\{c_1, c_3, c_4\}$  and three MSSes:  $\{c_1, c_4\}$ ,  $\{c_1, c_3\}$ ,  $\{c_2, c_3, c_4\}$ .

The *powerset* of  $C$ , i.e. the set of all its subsets, forms a lattice ordered via subset inclusion and denoted by  $\mathcal{P}(C)$ . In our algorithm we are going to deal with the so-called *chains* of the powerset and deal with local MUSes and MSSes, defined as follows.

► **Definition 3.** Let  $C$  be a finite set of constraints. The sequence  $K = \langle N_1, \dots, N_i \rangle$  is a *chain* in  $\mathcal{P}(C)$  if  $\forall j : N_j \in \mathcal{P}(C)$  and  $N_1 \subset N_2 \subset \dots \subset N_i$ . We say that  $N_k$  is a *local MUS* of  $K$  if  $N_k$  is unsatisfiable and  $\forall j < k : N_j$  is satisfiable. Similarly, we say that  $N_k$  is a *local MSS* of  $K$  if  $N_k$  is satisfiable and  $\forall j > k : N_j$  is unsatisfiable.

Note that there is no local MUS if all subsets on the chain are satisfiable, and there is no local MSS if all subsets on the chain are unsatisfiable.

### 3 Algorithm

In this section, we gradually present an online MUS/MSS enumeration algorithm, dubbed TOME. Consider first a naive enumeration algorithm that would explicitly check each subset of  $C$  for satisfiability, split the subsets of  $C$  into satisfiable and unsatisfiable subsets, and choose the maximal and minimal subsets of the two groups, respectively. The main disadvantage of this approach is the large number of satisfiability checks. Checking a given subset of  $C$  for satisfiability is usually an expensive task and the naive solution makes an exponential number of these checks which makes it unusable.

Note that the problem of MUS enumeration contains the solution to the problem of satisfiability of all subsets of  $C$  as each unsatisfiable subset of  $C$  is a superset of some MUS. This means that every algorithm that solves the problem of MUS enumeration has to make several satisfiability checks during its execution. These checks are usually done employing an external satisfiability solver. Clearly, the number of such external calls corresponds with the efficiency of the algorithm. Therefore, we want to minimise the number of calls to the solver.

#### 3.1 Basic Schema

Recall that the elements of  $\mathcal{P}(C)$  are partially ordered via subset inclusion and each element is either satisfiable or unsatisfiable. The key assumption on the constraint domain, as declared above, is that the partial ordering of subsets is preserved by the satisfiability of these subsets. If we thus find an unsatisfiable subset  $N_u$  of  $C$  then all supersets of  $N_u$  are also unsatisfiable; dually, if we find a satisfiable subset  $N_s$  of  $C$  then all subsets of  $N_s$  are also satisfiable. Moreover, none of the supersets of  $N_u$  can be a MUS and none of the subsets of  $N_s$  can be an MSS. In the following text we refer to this property as to the *monotonicity* of  $\mathcal{P}(C)$  and to the elements of  $\mathcal{P}(C)$  as to *nodes*.

The basic schema of TOME is shown as Algorithm 1. The schema consists of two phases. In the first phase it determines the satisfiability of all nodes and extracts from  $\mathcal{P}(C)$  a set of MSS *candidates*  $MSS_{can}$  and a set of MUS candidates  $MUS_{can}$  ensuring that  $MSS(C) \subseteq MSS_{can}$  and  $MUS(C) \subseteq MUS_{can}$ . In the second phase it reduces  $MSS_{can}$  to  $MSS(C)$  and  $MUS_{can}$  to  $MUS(C)$ .

**Algorithm 1:** The basic schema of TOME algorithm

---

```

1  $Unex \leftarrow \mathcal{P}(C)$ 
2  $MSS_{can}, MUS_{can} \leftarrow \emptyset$ 
3 while  $Unex$  is not empty do
4    $Nodes \leftarrow$  some unexplored nodes
5   for each  $N \in Nodes$  do
6     if  $N$  is satisfiable then
7        $MSS_{can} \leftarrow MSS_{can} \cup \{N\}$ 
8        $Unex \leftarrow Unex \setminus Sub(N)$ 
9     else
10       $MUS_{can} \leftarrow MUS_{can} \cup \{N\}$ 
11       $Unex \leftarrow Unex \setminus Sup(N)$ 
12 extract MSSes from  $MSS_{can}$ 
13 extract MUSes from  $MUS_{can}$ 

```

---

During the execution of the first phase the algorithm maintains a classification of nodes; each node can be either *unexplored* or *explored* and some of the explored nodes can belong to  $MSS_{can}$  or to  $MUS_{can}$ . The *explored* nodes are those whose satisfiability the algorithm already knows and the *unexplored* nodes are the remaining ones. The algorithm stores the unexplored nodes in the set  $Unex$  which initially contains all nodes from  $\mathcal{P}(C)$ . The first phase is iterative; the algorithm in each iteration selects some unexplored nodes  $Nodes$ , determines their satisfiability using an external satisfiability solver, and exploits the monotonicity of  $\mathcal{P}(C)$  to deduce satisfiability of some other unexplored nodes. At the end of each iteration the algorithm updates the set  $Unex$  by removing from it the nodes whose satisfiability was decided in this iteration. Based on its satisfiability, every node from the set  $Nodes$  is added either into  $MSS_{can}$  or  $MUS_{can}$ .

In the pseudocode, we use  $Sup(N)$  to denote the set of all unexplored supersets of  $N$  including  $N$  and  $Sub(N)$  to denote the the set of all unexplored subsets of  $N$  including  $N$ .

Clearly, the schema converges as the set of unexplored nodes decreases its size in every iteration. The schema also ensures that after the last iteration it holds that  $MUS(C) \subseteq MUS_{can}$  and  $MSS(C) \subseteq MSS_{can}$ . This is directly implied by the monotonicity of  $\mathcal{P}(C)$  as no node whose satisfiability was deduced can be an MSS and dually no node whose unsatisfiability was deduced can be a MUS.

In the second phase TOME extracts all MUSes and MSSes from  $MUS_{can}$  and  $MSS_{can}$ . Both these extractions can be done by any algorithm that extracts the highest and the lowest elements from any partially ordered set. A trivial algorithm can just test each pair of elements for the subset inclusion and remove the undesirable elements, which can be done in time polynomial to the number of constraints in  $C$  and the size of the sets of candidates. We assume that this part of our algorithm is not as expensive as the rest of it, especially when each check for a satisfiability of a set of constraints may require solving an NP-hard problem. We therefore omit the discussion of the second phase in the following and focus solely on the way the set  $Nodes$  is chosen in each iteration and the way the unexplored nodes are managed.

### 3.2 Symbolic Representation of Nodes

TOME highly depends on an efficient management of nodes. In particular it needs to reclassify some nodes from unexplored to explored and build chains from the unexplored

nodes. Probably the simplest way of managing nodes would be their explicit enumeration; however, there are exponentially many subsets of  $C = \{c_1, \dots, c_n\}$  and their explicit enumeration is thus intractable for large instances. We thus use a symbolic representation of nodes instead.

We exploit the well-known isomorphism between finite powersets and Boolean algebras. That is, we encode the set of constraints  $C = \{c_1, \dots, c_n\}$  using a set of Boolean variables  $X = \{x_1, \dots, x_n\}$ . Each subset of  $C$  (i.e. each node in our algorithm) is then represented by a valuation of the variables of  $X$ . This allows us to represent sets of nodes using Boolean formulae over  $X$ . We use  $f(Nodes)$  to denote the Boolean formula representing the set  $Nodes$  in the following.

As an example, consider a set of constraints  $C = \{c_1, c_2, c_3\}$  and let  $Nodes = \{\{c_1\}, \{c_1, c_2\}, \{c_1, c_3\}\}$  be a set of three nodes. Using the Boolean variables representation of  $C$ , we can encode the set  $Nodes$  using the Boolean formula  $f(Nodes) = x_1 \wedge (\neg x_2 \vee \neg x_3)$ .

The advantage of this representation is that we can efficiently perform set operations over sets of nodes. The union of two sets of nodes  $NodesA, NodesB$  is carried out as a disjunction and their intersection as a conjunction. To get an arbitrary node from a given set, say  $Unex$ , we use an external SAT solver (more details in the next subsection). Note that this means that TOME employs two external solvers: One is the constraint satisfaction solver that decides satisfiability of the nodes, one is the SAT solver that works with our Boolean description of the constraint set and is employed to produce unexplored nodes. To clearly distinguish between these two we shall in the following use the phrases “constraint solver” and “SAT solver” rigorously.

### 3.3 Unexplored Nodes Selection

Let us henceforth denote one specific call to the constraint solver as a *check*. We now clarify which nodes TOME chooses in each of its iterations to be *checked* and which nodes it adds into the sets of candidates on MUSes and MSSes. We also extend the basic schema which was presented as Algorithm 1. We want to minimise the ratio of performed checks to the number of nodes in  $\mathcal{P}(C)$ . Every algorithm for solving the problem of MUSes enumeration has to perform at least as many checks as there are MUSes, so this ratio can never be zero. Also, it is impossible to achieve the ratio with a minimal value without knowing which nodes are satisfiable and which are not and this information is not a part of the input of our algorithm. Instead of minimising this overall ratio, TOME tends to minimise this ratio locally in each of its iterations.

In order to select the nodes which are checked in one specific iteration, TOME at first constructs an *unexplored chain*. An *unexplored chain* is a chain  $K = \langle N_1, \dots, N_k \rangle$  that contains only unexplored nodes and that cannot be extended by adding another unexplored nodes to its ends, i.e.  $N_1$  has no unexplored subset and  $N_k$  has no unexplored superset. The monotonicity of  $\mathcal{P}(C)$  implies that either (i) all nodes of  $K$  are satisfiable, (ii) all nodes of  $K$  are unsatisfiable, or (iii)  $K$  has a local MSS and a local MUS, i.e. there is some  $j$  such that  $\forall 0 \leq i \leq j : N_i$  is satisfiable and  $\forall k \geq l > j : N_l$  is unsatisfiable. This allows us to employ binary search to find such  $j$  performing only logarithmically many checks in the length of the chain. Let us analyse the three possible cases:

- (i) all nodes of  $K$  are satisfiable, hence TOME deduces that all proper subsets of  $N_k$  are satisfiable and none of them can be an MSS, and it marks  $N_k$  as an MSS candidate;
  - (ii) all nodes of  $K$  are unsatisfiable, hence TOME deduces that all proper supersets of  $N_1$  are unsatisfiable and none of them can be a MUS, and it marks  $N_1$  as a MUS candidate;
- or

---

**Algorithm 2:** The modification of the basic schema of TOME
 

---

```

2 ...
3 while Unex is not empty do
4    $K \leftarrow$  some unexplored chain           // this line is added
5    $Nodes \leftarrow$  processChain( $K$ )           // this line is modified
6   for each  $N \in Nodes$  do
7     | ...

```

---

- (iii)  $N_j$  is the local MSS of  $K$  and  $N_{j+1}$  is its local MUS, hence TOME deduces that all proper subsets of  $N_j$  are satisfiable, all proper supersets of  $N_{j+1}$  are unsatisfiable, and it marks  $N_j$  as an MSS candidate and  $N_{j+1}$  as a MUS candidate.

Algorithm 2 shows the modification of the basic schema of TOME (see Algorithm 1) which incorporates the above method for choosing nodes to be checked. At the beginning of each iteration the algorithm finds an unexplored chain  $K$  which is subsequently processed by the *processChain* method. This method finds the local MUS and local MSS of  $K$  (possibly only one of those) using binary search and returns them.

To construct an unexplored chain, TOME first finds a pair of unexplored nodes  $(N_1, N_k)$  such that  $N_1 \subseteq N_k$  and then builds a chain  $\langle N_1, N_2, \dots, N_{k-1}, N_k \rangle$  by connecting these two nodes. The intermediate nodes  $N_2, \dots, N_{k-1}$  are obtained by adding one by one the constraints from  $N_k \setminus N_1$  to the node  $N_1$ . We refer to each such pair of unexplored nodes  $(N_1, N_k)$  that are the end nodes of some unexplored chain as to an *unexplored couple*.

In order to find an unexplored couple TOME asks for a member of *Unex* by employing the SAT solver (by asking for a model of the formula  $f(Unex)$ ). Besides the capability of finding an arbitrary member of *Unex*, we require the following capability: For a given member  $N_p \in Unex$ , the SAT solver should be able to produce a *minimal*  $N_q \in Unex$  such that  $N_q \subseteq N_p$ , where *minimal* means that there is no other  $N_r$  with  $N_r \subset N_q$ . Similarly, we require the SAT solver to be able to produce *maximal* such  $N_q$ . One of the SAT solvers that satisfies our requirements is miniSAT [8] that allows the user to fix values of some variables and to select a default polarity of variables at decision points during solving. To obtain a minimal  $N_q$  which is a subset of  $N_p$ , we set the default polarity of variables to False and fix the truth assignment to the variables that have been assigned False in  $N_p$ . Similarly for the maximal case.

We now describe two approaches of obtaining unexplored couples, assuming that we employ a SAT solver satisfying the above requirements.

**Basic approach.** The *Basic approach* consists of two calls to the SAT solver. The first call asks the SAT solver for an arbitrary minimal member of *Unex*. If nothing is returned then there are no more unexplored nodes. Otherwise we obtain a node  $N_k$  which is minimal in *Unex*. We then ask the SAT solver for a maximal node  $N_l \in Unex$  such that  $N_l$  is a superset of  $N_k$ . The pair  $(N_k, N_l)$  is then the new unexplored couple.

**Pivot based approach.** Supposing that the SAT solver works deterministically, a series of calls for maximal (minimal) nodes of *Unex* may return nodes from some local part of the search space that may lead to construction of unnecessarily short chains. Therefore, we propose to first choose a *pivot*  $N_p$ , an unexplored node which may be neither maximal nor minimal and which should be chosen somehow randomly. As the next step this approach



---

**Algorithm 3:**  $\text{processChain}(C, K = \langle N_1, \dots, N_k \rangle)$

---

```

1 find local MSS  $N_s$  and MUS  $N_u$  of  $K$  using binary search
2 if  $u < S(|K|)$  then
3    $N_u \leftarrow \text{shrink}(N_u)$ 
4   yieldMUS( $N_u$ ) // Output MUS
5 if  $s > |K| - G(|K|)$  then
6    $N_s \leftarrow \text{grow}(N_s)$ 
7   yieldMSS( $N_s$ ) // Output MSS
8 return  $\{N_u, N_s\}$  // Note that  $N_u$  or  $N_s$  may not exist

```

---

asks the SAT solver for a minimal node  $N_k$  such that  $N_k \subseteq N_p$  and for a maximal node  $N_l$  such that  $N_p \subseteq N_l$ . The new unexplored couple is then  $(N_k, N_l)$ . The randomness in choosing the node  $N_p$  is expected to ensure that we hit a part of  $Unex$  with large chains.

To get the pivot, we can set the SAT to assign a random polarity to variables at the decision points during solving.

### 3.4 Online MUS/MSS Enumeration

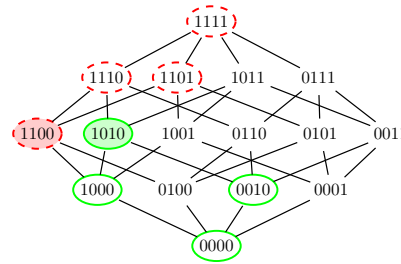
TOME as presented until now is only able to provide MUSes and MSSes in the second phase, after it finishes exploring all the nodes. We now describe the last piece of TOME, namely the way of producing MUSes and MSSes during the execution of the first phase. To do so, we need to employ two procedures: The *shrink* procedure is an arbitrary method that can turn an unsatisfiable node  $N_u$  into a MUS. Dually, the *grow* procedure is a method that can turn a satisfiable node  $N_s$  into MSS. A simple shrink (grow) method iteratively attempts to remove (add) constraints from  $N_u$  ( $N_s$ ), checking each new set for satisfiability and keeping any changes that leave the set unsatisfiable (satisfiable). These simple variants serve just as illustrations, there are known efficient implementations of both shrink and grow for specific constraint domains; as an example see MUSer2 [4] which implements the shrink method for Boolean constraints systems.

Recall that as a result of processing a single chain  $K$ , TOME finds either a local MUS  $N_u$ , or a local MSS  $N_s$ , or both of them. To get a MUS (MSS) we propose to employ the shrink (grow) method on this local MUS (MSS). However, performing shrink (grow) on each local MUS (MSS) can be quite expensive and can significantly slow down TOME. The amount of time needed for performing one specific shrink (grow) of  $N_u$  ( $N_s$ ) correlates with the position of  $N_u$  ( $N_s$ ) on  $K$ ; the closer  $N_u$  ( $N_s$ ) is to the start (end) of  $K$  the bigger amount of time needed for the shrink (grow) can be expected.

Therefore, we propose to shrink (grow) only some of the local MUSes (MSSes) based on their position on  $K$ . Let  $|K|$  be the length of  $K$ ,  $u$  the index of  $N_u$  in  $K$ , and  $S : \mathbb{N} \rightarrow \mathbb{N}$  be an arbitrary user defined function. TOME shrinks  $N_u$  into a MUS if and only if  $u < S(|K|)$ . As an example, consider  $S(x) = \frac{x}{2}$ ; in such case  $N_u$  is shrunk only if it is contained in the first half of  $K$ . Similarly, let  $s$  be the index of local MSS  $N_s$  of chain  $K$  and  $G : \mathbb{N} \rightarrow \mathbb{N}$ . The local MSS  $N_s$  is grown only if  $s > |K| - G(|K|)$ , which for example for  $G(x) = \frac{x}{2}$  means that  $N_s$  is grown only if it is contained in the second half of  $K$ . The complexity of performing shrinks also depends on the type of constrained system that is being processed, therefore the concrete choice of  $S$  and  $G$  is left as a parameter of our algorithm. Algorithm 3 shows an extended version of the method *processChain* which is able to produce MUSes and MSSes during its execution based on the above mechanism.

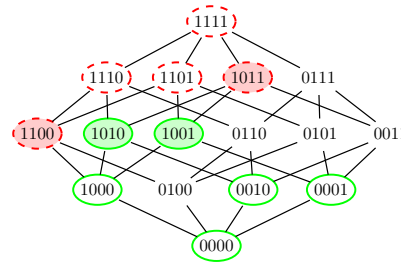
**I. iteration**

- Unex. couple  $\langle 0000, 1111 \rangle$
- Unex. chain  $\langle 0000, 1000, 1100, 1110, 1111 \rangle$
- Local MSS 1000 and local MUS 1100 are found and grown/shrunk to MSS 1010 and MUS 1100
- $MSS_{can} = \emptyset$  is updated to  $\{1010\}$
- $MUS_{can} = \emptyset$  is updated to  $\{1100\}$
- $f(Unex)$  is set to  $(x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_2)$



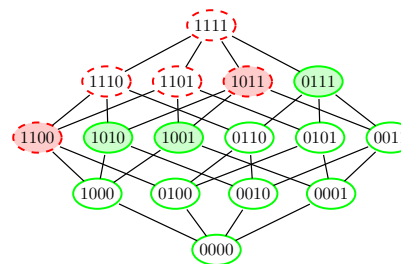
**II. iteration**

- Unexplored couple  $\langle 0001, 1011 \rangle$
- Unexplored chain  $\langle 0001, 1001, 1011 \rangle$
- Local MSS 1001 is grown to the MSS 1001
- local MUS 1011 is shrunk to the MUS 1011
- $MSS_{can} \leftarrow MSS_{can} \cup \{1001\}$
- $MUS_{can} \leftarrow MUS_{can} \cup \{1011\}$
- $f(Unex) \equiv (x_2 \vee x_4) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$



**III. iteration**

- Unexplored couple  $\langle 0011, 0111 \rangle$
- Unexplored chain  $\langle 0011, 0111 \rangle$
- Local MSS 0111 is grown to the MSS 0111
- local MUS *undefined*
- $MSS_{can} \leftarrow MSS_{can} \cup \{0111\}$
- $f(Unex) \equiv (x_2 \vee x_4) \wedge (x_2 \vee x_3) \wedge (x_1) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$



■ **Figure 1** An example execution of TOME.

**3.5 Example Execution of TOME**

The following example explains the execution of TOME on a simple set of constraints. The example is illustrated in Fig. 1. Let  $C = \{c_1 = a, c_2 = \neg a, c_3 = b, c_4 = \neg a \vee \neg b\}$ ,  $S(x) = x$  and  $G(x) = x$ .

Initially  $MSS_{can} = \emptyset$ ,  $MUS_{can} = \emptyset$  and all nodes are unexplored, i.e.  $f(Unex) = True$ . Figure 1 shows the values of control variables in each iteration and also illustrates the current states of  $\mathcal{P}(C)$ . In order to save space we encode nodes as bitvectors, for example the node  $\{c_1, c_3, c_4\}$  is written as 1011.

After the last iteration of the first phase of TOME there is no model of  $f(Unex)$  (this means that  $Unex$  is empty),  $MSS_{can} = \{1010, 1001, 0111\}$  and  $MUS_{can} = \{1100, 1011\}$ . Because functions  $S$  and  $G$  were stated in this example as  $S(x) = x, G(x) = x$ , each candidate on MUS or MSS has been already shrunk or grown to MUS or MSS, respectively, therefore  $MSS(C) = MSS_{can}, MUS(C) = MUS_{can}$  and the second phase of TOME can be omitted.

Note that in the first iteration the node 1010 was found to be a MSS, which means that all its supersets are unsatisfiable. One could use this fact to mark all supersets of 1010 as explored, however our algorithm does not do this because some of these subsets can be MUSes (1011 in this example). If we were interested only in MSS enumeration we could mark all supersets of each MSS as explored; dually in the case of only MUS enumeration.

■ **Table 1** The number of instances in which the algorithms output at least one MSS (the first number in each cell) or MUS (the second number).

		$S(x)$						
		$x$	$0.8x$	$0.6x$	$0.4x$	$0.2x$	$0x$	
Basic approach	$G(x)$							
	$x$	56   56	151   40	150   33	144   12	149   16	151   0	
	$0.8x$	56   <b>60</b>	149   44	151   37	144   16	150   20	<b>152</b>   0	
	$0.6x$	56   <b>60</b>	149   44	144   35	144   18	151   22	151   0	
	$0.4x$	54   <b>60</b>	149   45	140   36	143   32	150   30	151   0	
	$0.2x$	53   <b>60</b>	148   45	138   43	138   40	144   35	145   0	
	$0x$	0   <b>60</b>	0   47	0   46	0   44	0   37	0   0	
Pivot based app.	$x$	56   56	151   40	151   32	151   14	151   12	144   0	
	$0.8x$	56   60	151   43	151   36	150   18	149   16	145   0	
	$0.6x$	56   60	151   43	151   35	151   18	<b>152</b>   16	144   0	
	$0.4x$	54   60	150   43	147   35	151   14	150   13	144   0	
	$0.2x$	51   60	146   45	145   31	148   12	148   12	143   0	
	$0$	0   <b>61</b>	0   33	0   22	0   11	0   9	0   0	
MARCO		<b>51</b>   <b>51</b>						

## 4 Experimental Results

We now demonstrate the performance of several variants of TOME on a variety of Boolean CNF benchmarks. In particular, we implemented in C++ both the Basic and the Pivot Based approach for constructing chains and we evaluated both these approaches using several variants of the functions  $S$  and  $G$ . We also give a comparison with the MARCO algorithm [14].

The MARCO algorithm was presented by its authors in two variants, the basic variant and the optimised variant which is tailored for MUS enumeration. Both variants are iterative. The basic variant finds in each iteration an unexplored node, checks its satisfiability and based on the result the node is either shrunk into a MUS or grown into an MSS. Subsequently, MARCO uses the monotonicity of  $\mathcal{P}(C)$  to deduce satisfiability of other nodes in the same way TOME does. The optimised variant differs from the basic variant in the selection of the unexplored node; it always selects a maximal unexplored node. If the node is unsatisfiable it is shrunk into a MUS, otherwise it is guaranteed to be an MSS. We used the optimised variant in our experiments. The pseudocodes of both variants can be found in [14]. The key difference between TOME and MARCO is the usage of local MUSes and MSSes which are much easier to find and can be used to prune the powerset in the same way as global MUSes/MSSes.

Note that both compared algorithms (MARCO and TOME) employ several external tools during their execution, namely a SAT solver for finding the unexplored nodes, a constraint solver to decide the satisfiability of constraint sets, and the two procedures *shrink* and *grow* mentioned above. The list of external tools coincides for both algorithms. Therefore, we reimplemented MARCO in C++ to ensure that the two algorithms use the same implementations of the shrink and grow methods and the same solvers. As both the SAT solver and constraint solver we used the miniSAT tool [8] and we used the simple implementation of the shrink and grow methods as described earlier. Note that there are some efficient implementations of the shrink and grow methods for Boolean constraints, however, in general there might be no effective implementation of these methods. That is why we used the simple implementations.

As experimental data we used a collection of 294 unsatisfiable Boolean CNF Benchmarks that were taken from the MUS track of the 2011 SAT competition [16]. The benchmarks range in their size from 70 to 16 million constraints and from 26 to 4.4 million variables

■ **Table 2** The 5% trimmed sum of outputted MSSes and MUSes (summed over all 294 instances). The first number in each cell is the number of outputted MSSes, the second is the number of outputted MUSes.

		$S(x)$						
		$x$	$0.8x$	$0.6x$	$0.4x$	$0.2x$	$0x$	
Basic approach	$G(x)$							
	$x$	1744   339	9798   212	9936   87	6942   0	9726   2	10216   0	
	$0.8x$	1741   344	9908   217	9756   94	6787   2	9684   6	9378   0	
	$0.6x$	1740   348	9859   224	6969   40	6999   4	9696   8	9436   0	
	$0.4x$	1877   436	10013   252	7218   67	7694   50	10420   39	10114   0	
	$0.2x$	1757   <b>635</b>	10161   527	7925   262	8196   101	<b>10853</b>   66	10111   0	
	0	0   632	0   554	0   356	0   107	0   68	0   0	
Pivot based app.	$x$	2535   349	8330   208	7775   71	6705   0	6725   0	5089   0	
	$0.8x$	2660   492	8336   255	7680   85	6961   4	6889   2	5061   0	
	$0.6x$	2771   567	8481   290	7779   92	7066   4	6830   2	5067   0	
	$0.4x$	2814   597	8418   388	7975   145	6814   0	6950   0	5302   0	
	$0.2x$	2763   837	<b>8633</b>   697	7220   41	6563   0	6409   0	4910   0	
		0	0   <b>839</b>	0   404	0   10	0   0	0   0	0   0
MARCO		<b>749</b>   <b>215</b>						

and were drawn from a variety of domains and applications. All experiments were run with a time limit of 60 seconds.

Due to the potentially exponentially many MUSes and/or MSSes in each instance, the complete MUS and MSS enumeration is generally intractable. Moreover, even outputting a single MUS/MSS can be intractable for larger instances as it naturally includes solving the satisfiability problem, which is for Boolean instances NP-complete. Table 1 shows in how many instances the variants of TOME were able to output at least one MUS or MSS. MARCO was able to output at least one MUS and one MSS in 51 instances whereas several variants of TOME were able to output some MSSes in about 150 instances and some MUSes in up to 60 instances. Some of the 296 instances are just intractable for the solver which is not able to perform even a single consistency check within the used time limit. The other significant factor that affected the results is the complexity of the shrink method. MARCO in every iteration either “hits” a satisfiable node and directly outputs it as an MSS or waits till the shrink method shrinks the unsatisfiable node into a MUS. Therefore, each call of the shrink method can suspend the execution for a nontrivial time.

One can see that TOME also suffers from the possibly very expensive shrink calls and performs very poorly when the  $S$  function is set to  $S(x) = x$ . On the other hand, the variants that perform only the “easier” shrinks by setting  $S$  to be  $S(x) < x$  achieved better results. The grow method is generally cheaper to perform than the shrink method as checking whether an addition of a constraint to a satisfiable set of constraints makes this set unsatisfiable is usually cheaper than the dual task. No significant difference between the Basic and the Pivot based approach was captured in this comparison.

Another comparison can be found in Table 2 that shows the 5% trimmed sums of outputted MSSes and MUSes (summed over all of the 294 instances), i.e. 5% of the instances with the least outputted MSSes (MSSes) and 5% of the instances with the most outputted MSSes (MSSes) were discarded. The trimmed sum is based on a trimmed median which is useful estimator in statistics because it discards the most extreme observations.

All variants of TOME were noticeably better in MSS enumeration than MARCO. In the case of MUS enumeration MARCO outperformed these variants of TOME that shrink only some of the local MUSes, i.e. variants where  $S(x) = 0.6x$  and  $S(x) = 0.4x$ . However, the variants with  $S(x) = x$  and  $S(x) = 0.8x$  performed better, especially the variant with

■ **Table 3** The results of the experiments with a time limit of 1800 seconds.

	MSS enumeration		MUS enumeration	
	at least one MSS	5% trimmed sum	at least one MUS	5% trimmed sum
MARCO	112	50855	112	7337
BA $S(x) = 0.2x, G(x) = 0.2x$	167	80921	52	159
BA $S(x) = x, G(x) = 0.2x$	106	61010	<b>114</b>	<b>19059</b>
PBA $S(x) = 0.8x, G(x) = 0.2x$	<b>170</b>	<b>118151</b>	76	14565
PBA $S(x) = x, G(x) = 0.2x$	104	61537	112	19030

$G(x) = 0.2x, S(x) = x$  outputted about three times more MUSes than MARCO. As the Pivot based approach is randomised its performance may vary if it is run repeatedly on the same instances; the result of a single run may be misleading. Therefore, we ran all tests of the Pivot based approach repeatedly and the tables show the average values.

The time limit of 60 seconds is quite short and the results of such experiments may be misleading. Therefore, we also evaluated MARCO and both the Basic approach (BA) and the Pivot based approach (PBA) on the same set of benchmarks with a time limit of 1800 seconds. The results of these experiments are shown in Table 3. We used two different settings for BA and two different settings for PBA which were chosen based on the results of the experiments with the time limit of 60 seconds. MARCO was able to output at least one MSS in 112 instances whereas PBA with  $S(x) = 0.8x$  and  $G(x) = 0.2x$  was able to output at least one MSS in 170 instances. Also, the 5% trimmed sum of outputted MSSes by PBA is more than 2 times higher the 5% trimmed sum of outputted MSSes by MARCO.

In the case of MUS enumeration the number of instances in which MARCO was able to output at least one MUS is almost the same as the number achieved by BA and PBA with  $S(x) = x, G(x) = 0.2x$ . However, the 5% trimmed sum of outputted MUSes by MARCO is significantly lower. We believe that this is caused by the relative complexity of performing shrinks. TOME performs easier shrinks because it shrinks local MUSes which are usually “closer” to (global) MUSes whereas MARCO shrinks random nodes. Therefore, MARCO may be able to perform some shrinks within the given time limit but it is able to perform significantly fewer shrinks than TOME.

## 5 Conclusion

In this paper, we have presented a novel algorithm for online enumeration of MUSes and MSSes, dubbed TOME, which is applicable to any type of constraint system. The core of the algorithm is based on a novel approach utilising the so-called local MUSes/MSSes found using binary search. This approach allows the algorithm to efficiently explore the space of all subsets of a given set of constraints. We have made an experimental comparison with MARCO, the state-of-the-art algorithm for online MUS and MSS enumeration. The results show that TOME outperforms MARCO. TOME can be built on top of any consistency solver and can employ any implementation of the *shrink* and *grow* methods, therefore any future advance in this areas can be reflected in the performance of TOME.

One direction of future research is to aim at parallel processing of the search space in order to improve the performance of our approach; there are usually many disjoint unexplored chains that can be processed concurrently. Another possible direction is to focus on some specific types of constraint systems and customise TOME to be more efficient for these systems.

---

**References**

---

- 1 Zaher S. Andraus, Mark H. Liffiton, and Karem A. Sakallah. Reveal: A formal verification tool for verilog designs. In *LPAR*, volume 5330 of *Lecture Notes in Computer Science*, pages 343–352. Springer, 2008.
- 2 James Bailey and Peter J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Practical Aspects of Declarative Languages*, pages 174–186. Springer, 2005.
- 3 Jiri Barnat, Petr Bauch, and Lubos Brim. Checking sanity of software requirements. In *SEFM 2012 Proceedings*, volume 7504 of *LNCS*, pages 48–62. Springer, 2012. doi:10.1007/978-3-642-33826-7\_4.
- 4 Anton Belov and Joao Marques-Silva. MUSer2: An efficient MUS extractor. *Journal on Satisfiability, Boolean Modeling and Computation*, 8:123–128, 2012.
- 5 Elazar Birnbaum and Eliezer L. Lozinskii. Consistent subsets of inconsistent systems: structure and behaviour. *J. Exp. Theor. Artif. Intell.*, 15(1):25–46, 2003.
- 6 Renato Bruni and Antonio Sassano. Restoring satisfiability or maintaining unsatisfiability by finding small unsatisfiable subformulae. *Electronic Notes in Discrete Mathematics*, 9:162–173, 2001.
- 7 Maria Garcia de la Banda, Peter J. Stuckey, and Jeremy Wazny. Finding all minimal unsatisfiable subsets. In *Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 32–43. ACM, 2003.
- 8 Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.
- 9 Benjamin Han and Shie-Jue Lee. Deriving minimal conflict sets by cs-trees with mark set in diagnosis from first principles. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 29(2):281–286, 1999.
- 10 Aimin Hou. A theory of measurement in diagnosis from first principles. *Artif. Intell.*, 65(2):281–328, 1994.
- 11 Ulrich Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In *AAAI*, pages 167–172. AAAI Press / The MIT Press, 2004.
- 12 Mark H. Liffiton and Ammar Malik. Enumerating infeasibility: Finding multiple muses quickly. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings*, volume 7874 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2013.
- 13 Mark H. Liffiton, Michael D. Moffitt, Martha E. Pollack, and Karem A. Sakallah. Identifying conflicts in overconstrained temporal problems. In *IJCAI*, pages 205–211. Professional Book Center, 2005.
- 14 Mark H. Liffiton, Alessandro Previti, Ammar Malik, and Joao Marques-Silva. Fast, flexible MUS enumeration. *Constraints*, pages 1–28, 2015.
- 15 Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, 40(1):1–33, 2008.
- 16 MUS track of the 2011 SAT competition. <http://www.cril.univ-artois.fr/SAT11/>.
- 17 Yoonna Oh, Maher N. Mneimneh, Zaher S. Andraus, Karem A. Sakallah, and Igor L. Markov. AMUSE: a minimally-unsatisfiable subformula extractor. In *DAC*, pages 518–523. ACM, 2004.
- 18 Alessandro Previti and João Marques-Silva. Partial MUS enumeration. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press, 2013.
- 19 Lintao Zhang and Sharad Malik. Extracting small unsatisfiable cores from unsatisfiable boolean formula. *SAT*, 3, 2003.

