31st International Symposium on Theoretical Aspects of Computer Science

STACS'14, March 5th to March 8th, 2014, Lyon, France

Edited by

Ernst W. Mayr

Natacha Portier





Editors

Ernst W. Mayr
Fakultät für Informatik
Technische Universität München
mayr@in.tum.de

Natacha Portier
LIP
École Normale Supérieure de Lyon
natacha.portier@ens-lyon.fr

ACM Classification 1998
F.1.1 Models of Computation, F.2.2 Nonnumerical Algorithms and Problems, F.4.1 Mathematical Logic, F.4.3 Formal Languages, G.2.1 Combinatorics, G.2.2 Graph Theory

ISBN 978-3-939897-65-1

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/978-3-939897-65-1.

Publication date
March, 2014

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:
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.STACS.2014.i

ISBN 978-3-939897-65-1 **ISSN 1868-8969** **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 (RWTH Aachen)
- Pascal Weil (*Chair*, CNRS and University Bordeaux)
- Reinhard Wilhelm (Saarland University and Schloss Dagstuhl)

ISSN 1868-8969

www.dagstuhl.de/lipics

Foreword

The Symposium on Theoretical Aspects of Computer Science (STACS) conference series is an international forum for original research on theoretical aspects of computer science. Typical areas are (cited from the call for papers for this year's conference): algorithms and data structures, including: parallel, distributed, approximation, and randomized algorithms, computational geometry, cryptography, algorithmic learning theory, analysis of algorithms; automata and formal languages, games; computational complexity, parameterized complexity, randomness in computation; logic in computer science, including: semantics, specification and verification, rewriting and deduction; current challenges, for example: natural computing, quantum computing, mobile and net computing.

STACS is held alternately in France and in Germany. This year's conference (taking place March 5–8 in Lyon) is the 31st in the series. Previous meetings took place in Paris (1984), Saarbrücken (1985), Orsay (1986), Passau (1987), Bordeaux (1988), Paderborn (1989), Rouen (1990), Hamburg (1991), Cachan (1992), Würzburg (1993), Caen (1994), München (1995), Grenoble (1996), Lübeck (1997), Paris (1998), Trier (1999), Lille (2000), Dresden (2001), Antibes (2002), Berlin (2003), Montpellier (2004), Stuttgart (2005), Marseille (2006), Aachen (2007), Bordeaux (2008), Freiburg (2009), Nancy (2010), Dortmund (2011), Paris (2012), and Kiel (2013).

The interest in STACS has remained at a high level over the past years. The STACS 2014 call for papers led to 210 submissions with authors from 35 countries. Each paper was assigned to three program committee members who, at their discretion, asked external reviewers for reports. The committee selected 54 papers during a three-week electronic meeting held in November/December. As co-chairs of the program committee, we would like to sincerely thank all its members and the many external referees for their valuable work. In particular, there were intense and interesting discussions. The overall very high quality of the submissions made the selection a difficult task.

This year, the conference included a tutorial. We would like to express our thanks to the speaker Neeraj Kayal for this tutorial, as well as to the invited speakers, Javier Esparza, Peter Bro Miltersen, and Luc Segoufin. Special thanks go to Andrei Voronkov for his EasyChair software (`http://www.easychair.org`). Moreover, we would like to warmly thank Marie Bozo and Chiraz Benamor for continuous help throughout the conference organization.

We would also like to warmly thank Nathalie Aubrun for preparing these conference proceedings, and Marc Herbstritt from the Dagstuhl/LIPIcs team for assisting us in the publication process and the final production of the proceedings. These proceedings contain extended abstracts of the accepted contributions and abstracts of the invited talks and the tutorial. The authors retain their rights and make their work available under a Creative Commons license. The proceedings are published electronically by Schloss Dagstuhl – Leibniz-Center for Informatics within their LIPIcs series.

STACS 2014 has received funds and help from the Labex MILYON, the École Normale Supérieure de Lyon, inria, the CNRS and the lab LIP at the École Normale Supérieure de Lyon. We thank them for their support!

Munich and Lyon, February 2014 — Ernst W. Mayr and Natacha Portier

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr, Natacha Portier

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

Conference Organization

Program Committee

Edith Cohen	Microsoft Research (Silicon Valley)
Samir Datta	Chennai Mathematical Institute
Laurent Doyen	CNRS, LSV, École Normale Supérieure de Cachan
Yuval Emek	Technion
Kousha Etessami	University of Edinburgh
Martin Hofmann	Ludwig-Maximilians-Universität München
Stefan Kratsch	Technische Universität Berlin
Jerzy Marcinkowski	Uniwersytet Wrocławski
Wim Martens	Universität Bayreuth
Ernst W. Mayr	TUM – Technische Universität München (co-chair)
Gonzalo Navarro	Universidad de Chile
Eldar Fischer	Technion
Joachim Niehren	INRIA Lille
Simon Perdrix	CNRS, Université de Grenoble
Natacha Portier	École Normale Supérieure de Lyon (co-chair)
Alex Rabinovich	Tel Aviv University
Venkatesh Raman	The Institute of Mathematical Sciences, Chennai
Heiko Röglin	Universität Bonn
Nitin Saxena	Indian Institute of Technology Kanpur
Sven Schewe	University of Liverpool
Henning Schnoor	Christian-Albrechts-Universität zu Kiel
Micha Sharir	Tel Aviv University
Wojciech Szpankowski	Purdue University
Ioan Todinca	Université d'Orléans
Stéphane Vialette	Université de Marne-la-Vallée
Marius Zimand	Towson University

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr, Natacha Portier
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Local Organization Committee

Pablo Arrighi
Nathalie Aubrun
Chiraz Benamor
Marie Bozo
Pascal Koiran
Aurélie Lagoutte
William Lochet
Joachim Niehren
Timothée Pécatte
Irena Penev
Aniela Popescu
Natacha Portier
Michael Rao
Matthieu Rosenfeld
Mathieu Sablik
Maxime Senot
Sébastien Tavenas
Eric Thierry
Stéphan Thomassé
Nicolas Trotignon
Théophile Trunck
Petru Valicov

External Reviewers

Pankaj K. Agarwal
Michael H. Albert
Eric Allender
Noga Alon
Kazuyuki Amano
Andris Ambainis
Hyung-Chan An
Daniel Apon
Diego Arroyuelo
Christian Artigues
Eugene Asarin
Noa Avigdor-Elgrabli
Yossi Azar
Ashwinkumar Badanidiyuru
Guillaume Bagan
Sebastian Bala
Nikhil Balaji
Grey Ballard
Evripidis Bampis
Nikhil Bansal
Pablo Barceló
Stephan Barth
Surender Baswana
Bruno Bauwens
Cristina Bazgan
Paul Beame
Florent Becker
Djamal Belazzougui
Amir M. Ben-Amram
Itay Berman
Dietmar Berwanger
Daniela Besozzi
Olaf Beyersdorff
Abhishek Bhrushundi
Laurent Bienvenu
Somenath Biswas
Alexandre Blondin Massé
Andrej Bogdanov
Adrien Boiret
Marthe Bonamy
Vasco Brattka
Nicolas Broutin
Nathan Brunelle
Laurent Bulteau
Leizhen Cai
Gruia Calinescu
Cezar Câmpeanu
David Cattanéo
Keren Censor-Hillel
Sourav Chakraborty
Parinya Chalermsook
Timothy M. Chan
Mathieu Chapelle
Krishnendu Chatterjee
Wei Chen
Siu-Wing Cheng
Alexey Chernov
Rajesh Hemant Chitnis
Eden Chlamtac
Yongwook Choi
Manolis Christodoulakis
Jacek Cichon
Christophe Clavier
Ilan Reuven Cohen
Alfredo Costa
Christophe Crespelle
Vladimír Cunát
Radu Curticapean
Marek Cygan
Ugo Dal Lago
Anindya De
Ronald de Wolf
Daniel Delling
Xiaotie Deng
Luc Devroye
Giuseppe Di Battista
Michael Dinitz
Riccardo Dondi
Swan Dubois
Johannes Ebbing
Alon Efrat
Michael Elberfeld
Matthias Englert
David Eppstein
Leah Epstein
Isabelle Fagnot
Arash Farzan
Michal Feldman
Stefan Felsner
Stephen A. Fenner
Diodato Ferraioli
Esteban Feuerstein
Nathanaël Fijalkow
Emmanuel Filiot
Bernd Finkbeiner
Johannes Fischer
Vojtech Forejt
Fabrizio Frati
Bin Fu
Hu Fu
Martin Fürer
Travis Gagie
Martin Gairing
Anahí Gajardo
Anna Gál
Robert Ganian
Pawel Gawrychowski
Mina Ghashami
Emanuele Giaquinta
Matt Gibson
Christian Glaßer
Xavier Goaoc
Tomasz Gogacz
Yonatan Goldhirsh
Daniel Gonçalves
Laurent Gourvès
Bruno Grenet
Elena Grigorescu
Alexander Grigoriev
Martin Grohe
Roberto Grossi
Benoît Groz
Sudipto Guha
Sylvain Guillemot
Jiong Guo
Anshul Gupta
Leonid Gurvits
Serge Haddad
Emmanuel Hainry
Yijie Han
Sariel Har-Peled
Meng He
Lauri Hella
Danny Hermelin
John M. Hitchcock

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr, Natacha Portier
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Martin Hoefer
Thomas Holenstein
Stephan Holzer
Rupert Hölzl
Mathieu Hoyrup
Peter Høyer
Tomohiro I
Kazuo Iwama
Ragesh Jaiswal
Bart Jansen
Emmanuel Jeandel
Stacey Jeffery
Mark Jerrum
Artur Jez
Minghui Jiang
Jan Johannsen
Steffen Jost
Lukasz Kaiser
Iyad A. Kanj
Mamadou Moustapha Kanté
Haim Kaplan
Telikepalli Kavitha
Wojciech Kazana
Thomas Kesselheim
Stefan Kiefer
Christian Knauer
Bojana Kodric
Sudeshna Kolay
Giorgos Kollias
Christian Komusiewicz
Roberto Konow
Christian Konrad
Tsvi Kopelowitz
Amos Korman
Robin Kothari
Ioannis Koutis
Nagarajan Krishnamurthy
Ravishankar Krishnaswamy
Danny Krizanc
Gregory Kucherov
Sebastian Kuhnert
Raghav Kulkarni
Petr Kurka
Piyush P. Kurur
Samuel Kutin
Antti Kuusisto
Timo Kötzing
Oded Lachish
Aurélie Lagoutte
Martin Lange
Stefan Langerman
Elmar Langetepe
Markus Latte
Troy Lee
Virginie Lerays
Moshe Lewenstein
Mathieu Liedloff
Nutan Limaye
Vincent Limouzy
Kamal Lodaya
Daniel Lokshtanov
Katja Losemann
Zvi Lotker
Pinyan Lu
Giorgio Lucarelli
Michel Ludwig
Christof Löding
Guillaume Madelaine
Abram Magner
Kalpana Mahalingam
Sebastian Maneth
Bodo Manthey
Nicolas Markey
Barnaby Martin
Luke Mathieson
Ross M. McConnell
Andrew McGregor
Pierre McKenzie
Arne Meier
Reshef Meir
Robert Mercas
Ulrich Meyer
Ludovic Mignot
Joseph S. Miller
Peter Bro Miltersen
Mia Minnes
Neeldhara Misra
Johannes Mittmann
Matthias Mnich
Shahin Mohammadi
Ankur Moitra
Morteza Monemizadeh
Benjamin Monmege
Pedro Montealegre-Barba
Benjamin Moseley
Amer E. Mouawad
Moritz Mueller
Partha Mukhopadhyay
Viswanath Nagarajan
Satyadev Nandkumar
Joseph Naor
Guyslain Naves
Jesper Nederlof
Yakov Nekrich
Ilan Newman
Patrick K. Nicholson
André Nies
Matthias Niewerth
Prajakta Nimbhorkar
Nicolas Nisse
Joseph O'Rourke
Krzysztof Onak
Jan Otop
Giuseppe Ottaviano
Joël Ouaknine
Shannon Overbay
Zbigniew Palmowski
Katarzyna E. Paluch
Vinayaka Pandit
Fahad Panolan
David Parker
Vangelis Th. Paschos
Vinayak Pathak
Christophe Paul
Daniël Paulusma
Aduri Pavan
Anthony Perez
Clément Pernet
Geevarghese Philip
Chris Pinkau
Nir Piterman
Alexandru Popa
Lionel Pournin
Sebastian Preugschat
Simon J. Puglisi
Vuong Anh Quyen
Harald Räcke
Ashutosh Rai
M. S. Ramanujan
R. Ramanujam
Ramyaa Ramyaa
B. V. Raghavendra Rao
Michaël Rao
Ivan Rapaport

Jan Reimann
Lev Reyzin
Romeo Rizzi
Liam Roditty
Martin Rötteler
Andrei E. Romashchenko
Vincenzo Roselli
Günter Rote
Irena Rusu
Ignaz Rutter
Kunihiko Sadakane
Ocan Sankur
Jayalal M. N. Sarma
S. Srinivasa Rao
Ignasi Sau
Thomas Sauerwald
Saket Saurabh
Daniel M. Savel
Manfred Schmidt-Schauß
Thomas Schneider
Oded Schwartz
Ulrich Schöpp
Danny Segev
Maxime Senot
Frédéric Servais
Chintan Shah
Adam Sheffer
Alexander Shen
Mahsa Shirmohammadi
Somnath Sikdar
Florian Sikora
Narges Simjour
Alexander Skopalik
Shay Solomon
Srikanth Srinivasan
Damian Straszak
Ileana Streinu
Yann Strozecki
Karol Suchan
Xiaoming Sun
Chaitanya Swamy
Marek Szykula
Kunal Talwar
Seiichiro Tani
Sébastien Tavenas
Aris Tentes
Sharma V. Thankachan
Johan Thapper
Sophie Tison
Tomas Toft
Stefan Toman
Leen Torenvliet
Szymon Torunczyk
Jerzy Tyszkiewicz
Hanjo Täubig
Gregory Valiant
Rossano Venturini
Oleg Verbitsky
Nikolay K. Vereshchagin
Cristian Versari
José Verschae
Aravindan Vijayaraghavan
N. V. Vinodchandran
Nisheeth K. Vishnoi
Heribert Vollmer
Tjark Vredeveld
Magnus Wahlström
Justin Ward
Mark Daniel Ward
Jeremias Weihmann
Pascal Weil
Oren Weimann
Marcelo J. Weinberger
Piotr Wieczorek
Oliver Woizekowski
David Xiao
Liang Yu
Chenyi Zhang
Qin Zhang
Gelin Zhou
Sandra Zilles
Martin Zimmermann
Stanislav Zivny
Uri Zwick
Damien Woods

Contents

Invited talks

Keeping a Crowd Safe: On the Complexity of Parameterized Verification
Javier Esparza 1

Semi-algebraic geometry in computational game theory – a consumer's perspective
Peter Bro Miltersen 11

A glimpse on constant delay enumeration
Luc Segoufin 13

Tutorial

Arithmetic Circuit Complexity
Neeraj Kayal 28

Regular contributions

Submodular Stochastic Probing on Matroids
Marek Adamczyk, Maxim Sviridenko, and Justin Ward 29

On Symmetric Circuits and Fixed-Point Logics
Matthew Anderson and Anuj Dawar 41

Throughput Maximization in the Speed-Scaling Setting
Eric Angel, Evripidis Bampis, and Vincent Chau 53

Efficient Computation of Optimal Energy and Fractional Weighted Flow Trade-off Schedules
Antonios Antoniadis, Neal Barcelo, Mario Consuegra, Peter Kling, Michael Nugent, Kirk Pruhs, and Michele Scquizzato 63

Weighted Coloring in Trees
Julio Araujo, Nicolas Nisse, and Stéphane Pérennes 75

Generalized Reordering Buffer Management
Yossi Aza, Matthias Englert, Iftah Gamzu, and Eytan Kidron 87

Shapley meets Shapley
Haris Aziz and Bart de Keijzer 99

Complexity classes on spatially periodic Cellular Automata
Nicolas Bacquey 112

Asymmetry of the Kolmogorov complexity of online predicting odd and even bits
Bruno Bauwens 125

Two-Page Book Embeddings of 4-Planar Graphs
Michael A. Bekos, Martin Gronemann, and Chrysanthi N. Raftopoulou 137

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr, Natacha Portier
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Palindrome Recognition In The Streaming Model
Petra Berenbrink, Funda Ergün, Frederik Mallmann-Trenn, and Erfan Sadeqi Azer 149

New Bounds and Extended Relations Between Prefix Arrays, Border Arrays, Undirected Graphs, and Indeterminate Strings
Francine Blanchet-Sadri, Michelle Bodnar, and Benjamin De Winkle 162

Online Bin Packing with Advice
Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz 174

Balls into bins via local search: cover time and maximum load
Karl Bringmann, Thomas Sauerwald, Alexandre Stauffer, and He Sun 187

Meet Your Expectations With Guarantees: Beyond Worst-Case Synthesis in Quantitative Games
Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin 199

Chordal Editing is Fixed-Parameter Tractable
Yixin Cao and Dániel Marx 214

Online Dynamic Power Management with Hard Real-Time Guarantees
Jian-Jia Chen, Mong-Jen Kao, D.T. Lee, Ignaz Rutter, and Dorothea Wagner ... 226

Depth-4 Lower Bounds, Determinantal Complexity: A Unified Approach
Suryajith Chillara and Partha Mukhopadhyay 239

Constant Factor Approximation for Capacitated k-Center with Outliers
Marek Cygan and Tomasz Kociumaka 251

Bounds on the Cover Time of Parallel Rotor Walks
Dariusz Dereniowski, Adrian Kosowski, Dominik Pająk, and Przemysław Uznański 263

Packing a Knapsack of Unknown Capacity
Yann Disser, Max Klimm, Nicole Megow, and Sebastian Stiller 276

Exploring Subexponential Parameterized Complexity of Completion Problems
Pål Grønås Drange, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger 288

From Small Space to Small Width in Resolution
Yuval Filmus, Massimo Lauria, Mladen Mikša, Jakob Nordström, and Marc Vinyals 300

Explicit Linear Kernels via Dynamic Programming
Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos 312

Partition Expanders
Dmitry Gavinsky and Pavel Pudlák 325

Testing Generalised Freeness of Words
Paweł Gawrychowski, Florin Manea, and Dirk Nowotka 337

Counting Homomorphisms to Cactus Graphs Modulo 2
Andreas Göbel, Leslie Ann Goldberg, and David Richerby 350

Irreversible computable functions
Mathieu Hoyrup 362

Ehrenfeucht-Fraïssé Games on Omega-Terms
Martin Huschenbett and Manfred Kufleitner 374

Faster Sparse Suffix Sorting
Tomohiro I, Juha Kärkkäinen, and Dominik Kempa 386

Generalized Wong sequences and their applications to Edmonds' problems
Gábor Ivanyos, Marek Karpinski, Youming Qiao, and Miklos Santha 397

Read-Once Branching Programs for Tree Evaluation Problems
Kazuo Iwama and Atsuki Nagao 409

Computability of the entropy of one-tape Turing machines
Emmanuel Jeandel 421

Computing Optimal Tolls with Arc Restrictions and Heterogeneous Players
Tomas Jelinek, Marcus Klaas, and Guido Schäfer 433

Approximation of smallest linear tree grammar
Artur Jeż and Markus Lohrey 445

Coloring 3-colorable graphs with $o(n^{1/5})$ colors
Ken-ichi Kawarabayashi 458

Randomized Online Algorithms with High Probability Guarantees
Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke 470

An optimal quantum algorithm for the oracle identification problem
Robin Kothari 482

A Solution to Wiehagen's Thesis
Timo Kötzing 494

Space-Efficient String Indexing for Wildcard Pattern Matching
Moshe Lewenstein, Yakov Nekrich, and Jeffrey Scott Vitter 506

Synchronizing Relations on Words
Diego Figueira and Leonid Libkin 518

On Boolean closed full trios and rational Kripke frames
Markus Lohrey and Georg Zetzsche 530

Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but were afraid to ask)
Dániel Marx and Michał Pilipczuk 542

Data-Oblivious Data Structures
John C. Mitchell and Joe Zimmerman 554

Higher randomness and forcing with closed sets
Benoit Monin 566

Near-Optimal Generalisations of a Theorem of Macbeath
Nabil H. Mustafa and Saurabh Ray 578

Non-autoreducible Sets for NEXP
Dung T. Nguyen and Alan L. Selman 590

Differentiability of polynomial time computable functions
André Nies 602

2-Stack Sorting is polynomial
Adeline Pierrot and Dominique Rossin 614

Communication Lower Bounds for Distributed-Memory Computations
Michele Scquizzato and Francesco Silvestri 627

Stochastic Scheduling on Unrelated Machines
Martin Skutella, Maxim Sviridenko, and Marc Uetz 639

Computational Complexity of the Extended Minimum Cost Homomorphism Problem on Three-Element Domains
Hannes Uppman 651

The Complexity of Deciding Statistical Properties of Samplable Distributions
Thomas Watson 663

Faster Compact On-Line Lempel-Ziv Factorization
Jun'ichi Yamamoto, Tomohiro I, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda 675

Keeping a Crowd Safe: On the Complexity of Parameterized Verification

Javier Esparza

Faculty of Computer Science, Technical University of Munich, Germany
esparza@in.tum.de

Abstract

We survey some results on the automatic verification of parameterized programs without identities. These are systems composed of arbitrarily many components, all of them running exactly the same finite-state program. We discuss the complexity of deciding that no component reaches an unsafe state. The note is addressed at theoretical computer scientists in general.

1998 ACM Subject Classification F.1.1 Models of Computation, D.2.4 Software/Program Verification

Keywords and phrases Parameterized verification, automata theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.1

Category Invited Talk

1 Introduction

Parameterized programs (where "program" is used here in a wide sense) consist of arbitrarily many instantiations of the same piece of code. We call each of these instantiations a *process*, and the set of processes a *crowd*. Examples include many classical distributed algorithms (for mutual exclusion, leader election, distributed termination, and other problems), families of hardware circuits (for instance, a family of carry-look-ahead adders, one for each bitsize), cache-coherence protocols, telecommunication protocols, replicated multithreaded programs, algorithms for ad-hoc and vehicular networks, crowdsourcing systems, swarm intelligence systems, and biological systems at molecular level.

If automatic verification is not your field of expertise, then you may find awkward to study the complexity of verification problems for parameterized programs. Since Rice's theorem shows that any non-trivial question on the behavior of one single while-program is undecidable, is there any more to say? Actually, yes. Rice's theorem refers to while-programs acting on variables over an infinite domain (typically the integers). Since the primary task of distributed algorithms or cache-coherence protocols is not to compute a function, but solve a coordination problem, they typically use only boolean variables as semaphores, or variables ranging between 0 and the number of processes. So for each number N, the set of reachable configurations of the crowd with N processes is finite, and most verification questions can be decided by means of an exhaustive search of the configuration space.

However, this brute force technique can only show correctness for a finite number of values of N. This is not what we usually understand under "proving a parameterized program correct" , we mean proving that the property holds for *all* values of N. In other words, the task consists of proving that each member of an infinite family of systems, each of them having a finite state space, satisfies a given property. Are questions of this kind always undecidable?


© Javier Esparza;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 1–10

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


In the way we have formulated the problem, the answer is still negative: yes, all non-trivial problems are still undecidable. Let us sketch a proof for a simple reachability problem. Given a Turing machine M and an input x, we can easily construct a little finite-state program that simulates a tape cell. The program has a boolean variable indicating whether the head is on the cell or not, a variable storing the current tape symbol, and a third variable storing the current control state when the head is on the cell (if the head is not on the cell the value of this variable is irrelevant). A process running the program communicates with its left and right neighbors by message passing. If M accepts x, then it does so using a finite number N of tape cells. Therefore, the crowd containing N processes eventually reaches a configuration in which the value of the control-state variable of a process is a final state of M. On the contrary, if M does not accept x, then no crowd, however large, ever reaches such a configuration. So the reachability problem for parameterized programs is undecidable.

But this proof sketch contains the sentence "the program communicates with its left and right neighbors". How is this achieved? A communication structure where processes are organized in an array (like in our simulation of M), in a ring, a tree, or some other shape is achieved by giving processes an *identity*, typically a number in the range $[1..N]$. This identifier appears as a parameter i in the code, and so it is not the case that all processes execute *exactly* the same code, but the code where the parameter is instantiated with the process identity. For instance, the instruction "if you're not the last process in the array, then send the content of variable x to your right neighbor" is encoded as "if $i < N$, then send the content of variable x to process $i + 1$". (Observe that, since N also appears on the code, the processes also know how many they are.)

There are applications where processes have no identities and do not know—or do not care about—how many they are: for instance, in natural computing processes may be molecules swimming in a solution. In others applications identities are not needed. A typical example are cache-coherence protocols, whose purpose is to guarantee the consistency of all cache lines containing copies of a memory cell. The protocol should guarantee that if a process updates of a variable in its cache, the other processors mark their cached values as no longer valid. Since the situation is completely symmetric, and processors are connected by a bus implementing a broadcast communication primitive, identities are not needed. The same holds for many multithreaded programs where one only cares about, say, the number of threads that are still active. Finally, there is an increasing number of applications where identities are considered *harmful*. For instance, in vehicular networks cars may communicate with each other to interchange information about traffic jams. Since cars must necessarily communicate their positions, identities might allow one to track individual cars. Applications involving secret voting are another example.

These considerations lead us to our problem, which can be informally, but suggestively, formulated as follows:

What is the complexity of checking that a (finite, but arbitrarily large) anonymous crowd will stay safe?

Formally, the input to the problem is a finite automaton A, the *template*, representing the finite-state code to be executed by each process, and a state q_u of A, the *unsafe state*, modelling some kind of error or undesirable situation. The transitions of A correspond either to internal moves or to communications with the rest of the system. The question to be answered is whether there exists a number N such that some execution of the system composed by N identical copies of A reaches a configuration in which at least one of the processes is in the unsafe state q_u. We say that such configurations *cover* q_u, and so the problem is called the *coverability* problem.

The complexity of the coverability problem crucially depends on the power of the communication mechanism between processes. So first we must analyze these mechanisms in some detail. This is done in Section 2. Section 3 presents the complexity results. Finally, Section 4 briefly describes some additional work in which the template A is allowed to have more computational power than that of a finite automaton.

2 How Crowds Communicate

The two main communication paradigms are message-passing (typical of communication protocols and distributed systems where processes reside in different machines) and communication through global variables (typical of multithreaded programs). Within each paradigm there is a number of mechanisms. We informally describe the syntax and operational semantics of the template A for the four mechanisms most commonly found in the literature. In particular, we give the syntax of the transition labels of A, and describe how a communication takes place. We assume a finite set V of values which can be communicated.

Broadcast communication. Transition labels: $v!!$, $v??$.
We assume that for every state q and every value v the template A has at one transition $q \xrightarrow{\mathbf{v}??} q'$ for some state q' (which may be equal to q). In a communication step of the system *all processes make a move*. Exactly one of the processes takes a transition labelled by $\mathbf{v}!!$, with the intended meaning that this process broadcasts the value v to all others; simultaneously, all other processes take $v??$-transitions, depending on their current states.

Rendez-vous communication. Transition labels: $v!$, $v?$.
In a communication step of the system, exactly two processes make a move: a process takes a transition labelled by $\mathbf{v}!$, and, simultaneously, another process takes a transition labeled by $\mathbf{v}?$. The intended meaning is that the first process sends the value v to the second process.

Communication by global store. Transition labels: $w(v)$, $r(v)$.
In this paradigm we assume that all processes in the crowd communicate with a global store. At every time point the store contains an element of V. In a communication step, exactly one process makes a move. The process either takes a transition labeled by $w(v)$, which writes v into the store, or, if the current value of the store is v, it takes a transition labeled by $r(v)$, meaning that it reads the value v from the store.

Communication by global store with locking. Transition labels: *lock*, *unlock*, $w(v)$, $r(v)$.
A process must first obtain a lock on the store before being able to write or read. The processes keeps the lock until it releases it by means of a transition labeled by *unlock*. While in possession of the lock, the process is the only one that can perform reads and writes.

We shall see that the complexity of the coverability problem depends on two parameters of the communication mechanism:

(1) Who listens when a process speaks ? When a process sends a message, different mechanisms provide different guarantees on who will receive it, and we can classify them accordingly:

- **Everyone listens.** This is the case of broadcast communication.
- **At least someone listens.** This is obviously the case of rendez-vous, but also of global store with locking. Indeed, we can easily use a global store with locking to simulate rendez-vous communication. The store initially contains a special value, say f, standing for "store is free". A process wishing to communicate a value v acquires the lock, reads the content of the store, and, if its value is f, changes it to v and releases the lock. If the

value is not f, it just releases the lock. A process wishing to receive a value acquires the lock and reads the store: if its value is f, the process just releases the lock; otherwise, it copies the value into its local state and releases the lock. This guarantees that the value will be preserved until someone reads it, and, under a suitable fairness assumption, that it will eventually read.
However, neither rendez-vous communication nor global store with locking can implement broadcast. Intuitively, in these paradigms there is no way to detect that a process does not react to any message.

– **No guarantee.** This is the case of a global store without locking. A value written by a process can be overwritten by another process before anyone reads it. Notice that we can no longer implement rendez-vous using the trick above. Since the store cannot be locked, two processes P_1 and P_2 wishing to write values v_1, v_2 may both read the value f and proceed to write. If P_1 writes immediately before P_2, then the value v_1 is not read by anyone.

(2) Can the crowd produce a leader? Loosely speaking, this is the question whether a perfectly symmetric crowd in which initially all processes are in the same state can be forced to split into a distinguished process which stays within a special subset of states of the template, and an arbitrarily large crowd that stays within another subset. More precisely (but still a bit informally) the question is the following. Is there a template A with two distinguished states q_1, q_2 and all processes initially in q_1, such that some reachable configuration has one process in q_2, and no reachable configuration has more than one process on q_2?

Broadcast communication and communication through global store with locking can both easily produce a leader. In the case of broadcast communication, the template with transitions $q_1 \xrightarrow{a!!} q_2$ and $q_1 \xrightarrow{a??} q_3$ already does the trick. The process broadcasting the message moves to q_2 and, since all other processes *must* listen, they all move to q_3. In the case of global store, we choose a template in which all processes initially compete for the lock; the process that acquires it changes the value of the store to "we have a leader" and moves to q_2.

Rendez-vous communication and communication through a global store without a lock cannot produce a leader. Intuitively, the reason is that when process makes a move, arbitrarily many processes follow suit, making exactly the same move immediately after. We will come back to this point later.

3 The Power of Crowds

We can sort the four communication mechanism of the previous section in order of decreasing power according to our two criteria:

broadcast communication	(everybody must listen, leader can be produced)
global store with locking	(somebody must listen, leader can be produced)
rendez-vous communication	(somebody must listen, no leader can be produced)
global store without locking	(nobody must listen, no leader can be produced)

In this section we show that this informal classification is confirmed by the mathematical results: the complexity of the coverability problem decreases as we move down through the list.

Before describing the results, it is important to observe that the complexity of the coverability problem is related to the crowd's computational power seen as a nondeterministic

machine. If coverability is hard for a complexity class $\mathcal{C}$, then any problem in $\mathcal{C}$ can be reduced to coverability. Therefore, given an instance of the problem, we can construct a template A such that a large enough crowd will solve it: a process will reach the state q_u, which now instead of an unsafe state becomes the state at which the process can post the answer "yes". So—informally but suggestively—studying the complexity of the coverability problem amounts to studying the following question:

What is the computational power of a (finite but arbitrarily large) anonymous crowd?

In particular, a result proving high complexity of the coverability problem means bad news for crowd verifiers, but good news for crowd designers, and vice versa.

We are now ready to analyze the complexity of the four communication mechanisms above.

3.1 Communication by broadcast

Despite the power of broadcast communication, it was proved in [8] by Finkel, Mayr, and the author that the coverability problem is decidable. So we have:

> Anonymous crowds are not Turing powerful, or, conversely, identities are necessary in order to achieve full Turing power.

The proof is a straightforward application of a more general result of [1] on well-structured systems (see also [2, 10]). Let us sketch it. The configuration of a crowd with template A is completely determined by the number of processes at each state of A. So, given a numbering $\{q_1, \ldots, q_n\}$ of the states of A, a configuration can be formalized as a vector of $\mathbb{N}^n$. Assume without loss of generality that $q_u = q_1$. We wish to know whether, for some number N, a crowd of N individuals can reach a configuration $(k_1, \ldots, k_n)$ such that $k_1 \geq 1$, or, equivalently, a configuration $(k_1, \ldots, k_n) \geq (1, 0, \ldots, 0)$, where $\geq$ is defined componentwise. The set of configurations $(k_1, \ldots, k_n) \geq (1, 0, \ldots, 0)$ is *upward closed* (with respect to $\leq$), i.e., if a configuration c belongs to the set, then so does any other configuration of the form $c + c'$, where $c' \in \mathbb{N}^n$ and $+$ is defined componentwise.

Given an upward-closed set C of configurations, it is easy to show that its set of immediate predecessors (i.e., the set of configurations from which some configuration of C can be reached in one step) is also upward-closed. Indeed, assume we can reach a configuration $c \in C$ from some configuration d by means of the broadcast of a value v. Now, consider a configuration $d + d'$. If we perform the same broadcast, then the processes of d move to the same states as before, yielding again the configuration c, and the processes of d' move somewhere, yielding a configuration c'. The result is a configuration $c + c'$, where addition of configurations is defined componentwise. Since $c \in C$ and C is upward-closed, we have $c + c' \in C$, and we are done. So letting C_0 be the set of configurations $(k_1, \ldots, k_n) \geq (1, 0, \ldots, 0)$, the sequence $C_0, C_1, C_2, \ldots$, where C_{i+1} is the set of immediate predecessors of C_i, is a sequence of upward-closed sets.

We now exploit the well-known fact that the order $\geq$ is a *well-quasi-order*: every infinite sequence $v_1, v_2, \ldots$ of elements of $\mathbb{N}^n$ contains an infinite ordered subsequence $v_{i_1} \leq v_{i_2} \leq \ldots$. A first easy consequence of the theory of well-quasi-orders is that any upward-closed set of configurations has finitely many minimal elements with respect to $\leq$. So, since an upward-closed set is completely determined by its minimal elements, we can use the minimal elements as a finite representation of the set. This allows to explicitly construct the sequence

$C_0, C_1, C_2 \ldots$. A second easy consequence is that this sequence contains two indices $i < j$ such that $C_i \supseteq C_j$. So we can stop the construction at C_j, because subsequent steps will not discover any new configuration. The set $\bigcup_{k=0}^{j} C^k$ contains all configurations from which a configuration of C_0 can be reached. We can then inspect this set, and check whether is contains one of the possible initial configurations of a crowd.

So crowds communicating by broadcasts are not Turing powerful. But, how powerful are they? The answer, due to Schmitz and Schnoebelen [21], is very surprising:

> The complexity time of the coverability problem for anonymous crowds communicating by broadcast grows faster than any primitive recursive function.

More precisely, the result is that coverability of broadcast protocols is $\mathbf{F}_\omega$-hard, where $\mathbf{F}_\omega$ is a class of problems of "Ackermannian complexity" (i.e., whose complexity is bounded by an Ackermann-like function). In particular, $\mathbf{F}_\omega$ is closed under primitive recursive reductions. We refer to [21] for a more precise description. In any case, this is one of the most natural problem with provably non-primitive recursive complexity.

As a summary, we have that crowds communicating by broadcast may not be Turing powerful, but keeping them under control may quickly exceed any reasonable amount of computational resources.

3.2 Communication by global store with locking.

Global variables with locking is the natural communication mechanism for multithreaded programs. The coverability problem for this kind of communication reduces to the coverability problem of Petri nets, and vice versa, a fact that was already essentially observed by German and Sistla [13].

The coverability problem for Petri nets was proved to be EXPSPACE-complete already in the 70s, which yields the following result:

> The coverability problem for a crowd communicating by global variables with locking is EXPSPACE-complete.

EXPSPACE-hardness was proved by Lipton [16] (see also [7]) who showed that a counter able to count up to 2^{2^n} can be simulated by a Petri net (or an automaton) of size n^2. Membership in EXPSPACE was proved by Rackoff [19]. He shows that, if the state q_u is coverable, then it is coverable by a sequence of moves of double exponential length in the size of the template. This yields immediately a NEXPSPACE algorithm, after which we use NEXPSPACE=EXPSPACE.

Rackoff's nondeterministic algorithm is not useful in practice. A more practical algorithm was suggested (some years before Rackoff's paper) by Karp and Miller [15]. The algorithm uses the notion of *generalized configuration*, which for a template with n states is a vector of dimension n whose elements are either natural numbers of the symbol ω, which intuitively stands for "arbitrarily many processes", or "as many process as necessary". The algorithm starts at a generalized configuration describing the initial situation: for example, we may have exactly one process in state q_1, and arbitrarily many in state q_2, modelled by $(1, \omega, 0, \ldots, 0)$. Given a generalized configuration, we construct its successors (that is, the algorithm explores new configurations in the forward direction, contrary to the algorithm for broadcasts, which explores backwards). If the template, say, has transitions $q_1 \xrightarrow{v!} q_3$ and $q_2 \xrightarrow{v?} q_4$, then

a rendez-vous can take place, and we can move from $(1, \omega, 0, \ldots, 0)$ to $(0, \omega, 1, 1, 0, \ldots, 0)$. The important point is that this construction can be "accelerated". For example, if the template has transitions $q_1 \xrightarrow{v!} q_1$ and $q_2 \xrightarrow{v?} q_4$, then we can move from $(1, \omega, 0, \ldots, 0)$ to $(1, \omega, 0, 1, 0, \ldots, 0)$ (state q_2 loses a process, but we apply $\omega - 1 = \omega + 1 = \omega$) and, since $(1, \omega, 0, 1, 0, \ldots, 0) \geq (1, \omega, 0, \ldots, 0)$, the rendez-vous can take place again, leading to $(1, \omega, 0, 2, 0, \ldots, 0)$, $(1, \omega, 0, 3, 0, \ldots, 0)$, etc. The algorithm "jumps to the limit", and moves directly from $(1, \omega, 0, \ldots, 0)$ to $(1, \omega, 0, \omega, 0, \ldots, 0)$. Termination of the algorithm follows once more from a very simple application of the theory of well-quasi-orders.

Karp and Miller's algorithm has been recently improved in a number of ways: efficient data structures, construction of a minimal set of generalized configurations, etc. (see e.g. [18, 22, 12, 20]). However, these improvements do not change its worst-case complexity, which is surprisingly worse than that of Rackoff's algorithm: Karp and Miller's algorithm can take non-primitive recursive time and space. Recently, this puzzling mismatch has lead to two beautiful results. First, Bozzelli and Ganty have shown that the backwards algorithm described above for broadcast systems no longer has non-primitive recursive complexity when applied to the rendez-vous case. Instead, it runs in double exponential time, much closer to the lower bound [3]. Geeraerts, Raskin, and Van Begin have proposed another simple algorithm based on forward exploration [11]. It applies a so-called "Enlarge, Expand, and Check" algorithmic principle, which constructs a sequence of under- and overapproximations of the set of reachable generalized configurations.Very recently, Majumdar and Wang have shown that this algorithm also runs in double exponential time [17].

Early work by Delzanno, Raskin and Van Begin [5] and more recent work by Kaiser, Kröning and Wahl [14] (see also [6]) has applied these coverability algorithms and other techniques for the construction of over- and underapproximations, to verify safety of a large number of multithreaded programs.

3.3 Communication by rendez-vous

Rendez-vous communication is a natural communication model for systems whose processes "move" in some medium where they occasionally meet and interact. Natural computing systems in which computing entities are molecules moving in a "soup" are an example.

When studying the complexity of this problem there is a subtle point. As we have seen in Section 2, a crowd communicating by rendez-vous communication cannot produce a leader. However, one can set up the system so that the initial configuration *already contains one*. For instance, we can choose an initial configuration with exactly one process in state q_1, and arbitrarily many processes in state q_2. So we have to examine two cases.

Crowds with an initial leader. In this case we can easily use rendez-vous to simulate global store with locking. Intuitively, the template is designed so that the leader simulates the store, and the rest of the crowd only communicates with the leader. Conversely, as we saw in Section 2, rendez-vous communication can be simulated by a global store with locking, and so we obtain:

> The coverability problem for crowds communicating by rendez-vous and having an initial leader is EXPSPACE-complete.

Leaderless crowds. This is the case in which all processes are initially in the same state. In other words, if we assume that this state is q_1, then the initial generalized configuration of the system is $(\omega, 0, \ldots, 0)$. We can again solve the coverability problem by means of the

Karp-Miller algorithm. However, it is easy to see that in this special case the algorithm can only generate new configurations whose components are either ω or 0. Even more, a successor $(k'_1, \ldots, k'_n)$ of a generalized configuration $(k_1, \ldots, k_n)$ necessarily satisfies $k_i = \omega \Rightarrow k'_i = \omega$ for every $1 \leq i \leq n$. Therefore, a sequence of pairwise distinct generalized configurations has length at most n. We can then easily prove that the coverability problem is NP-complete, and so much simpler than the case of a leader.

3.4 Communication by global store without locking

Locking mechanisms are easy to implement in a multithreading environment where all threads are executed on a processor, or on a number of processors physically closed to each other. They become more problematic for crowdsourcing systems, ad-hoc networks, vehicular networks or, more generally, any sort of decentralized system where processes may enter or leave the system at any time. The danger of this setting is obvious: a process may acquire the lock, and leave the system without returning it, blocking the complete crowd. Additionally, the locking mechanism is not as easy to implement as in a multithreading environment.

The case of communication by global variables without locking has been recently investigated in [9]. The main finding is that the absence of locking drastically simplifies the task of controlling the crowd (good news for verifiers), or, equivalently, decreases the computational power (bad news for designers):

> The coverability problem for a crowd communicating by global variables without locking is NP-complete.

Moreover, in this case the result does not depend on the initial existence of a leader. Intuitively, in the rendez-vous case the template can be designed so that a process communicates a value to, say, exactly three other processes, which allows the crowd to perform some arithmetic. In particular, the crowd can store an integer n by putting exactly n processes in a given state of the template. This is not possible in a global store without a lock, because the process has no control on how many processes may read a value.

The NP-completeness result is proved with the help of two lemmas. The first lemma shows that the crowd can be simulated by a system composed of a finite number of *simulators*, one for each value of V. The simulator for the value v is an automaton A_v that can be easily constructed from the template A and the value v. So we can construct a finite crowd that simulates the behavior of any crowd with template A, of any size. This result already shows that the coverability problem is in PSPACE, but not yet that it belongs to NP. Membership in NP is proved with the help of a second lemma. Loosely speaking, the lemma states that, if the unsafe state is reachable, then it can be reached by means of computations of the simulators that can be guessed in polynomial time.

4 Some Results on Crowds of Infinite-State Processes

So far we have assumed that processes are finite state (i.e., the template is a finite automaton). If we totally relax this condition (for instance, if we allow processes to be Turing machines), then the coverability problem becomes of course undecidable: a crowd of one suffices to achieve Turing power! But we can consider milder extensions of the computational power of a process.

For broadcast communication and global variables with locking, even very modest extensions already make the crowd Turing powerful. In particular, this is already the case if

processes can count, i.e., if the template is a finite automaton whose transitions may act on a counter, increasing or decreasing it by one, or testing it for zero. Two processes suffices to simulate a two-counter machine, which are known to be Turing powerful. A crowd can select a leader, who can then select a second leader, and these two leaders can then communicate with each other, ignoring the messages from the rest of the crowd. The same applies to rendez-vous if the crowd initially contains a leader.

For global variables without locking, the situation is more interesting. In [9] two extensions are considered. First, the paper studies the case in which processes are pushdown automata (since stack can be used as a counter, this includes the counter case). The coverability problems remains NP-complete for "leaderless crowds" and becomes PSPACE-complete for crowds with one leader.

The second extension considers the case in which processes are Turing machines that can only run for polynomial time. This models the situation in which each process has no restrictions in computational power, but can only contribute a polynomial amount of work to the crowd. Since the crowd is arbitrarily large, the total amount of work is not bounded, and so we could hope to be able to show problems far beyond NP. However, the coverability problem remains NP-complete. Interpreting the result, we conclude that without a locking mechanism the crowd cannot distribute an arbitrary exponential computation among its members in such a way that each individual only does a polynomial amount of work.

Acknowledgements. Very special thanks to Pierre Ganty, Jan Křetínský, Michael Luttenberger, and Rupak Majumdar for numerous comments on former versions of this note. In particular, Rupak suggested the final structure.

References

1 Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321. IEEE Computer Society, 1996.

2 Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inf. Comput.*, 160(1–2):109–127, 2000.

3 Laura Bozzelli and Pierre Ganty. Complexity analysis of the backward coverability algorithm for vass. In Giorgio Delzanno and Igor Potapov, editors, *RP*, volume 6945 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2011.

4 Pedro R. D'Argenio and Hernán C. Melgratti, editors. *CONCUR 2013 – Concurrency Theory – 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, volume 8052 of *Lecture Notes in Computer Science*. Springer, 2013.

5 Giorgio Delzanno, Jean-François Raskin, and Laurent Van Begin. Towards the automated verification of multithreaded java programs. In Joost-Pieter Katoen and Perdita Stevens, editors, *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 173–187. Springer, 2002.

6 Alastair F. Donaldson, Alexander Kaiser, Daniel Kroening, Michael Tautschnig, and Thomas Wahl. Counterexample-guided abstraction refinement for symmetric concurrent programs. *Formal Methods in System Design*, 41(1):25–44, 2012.

7 Javier Esparza. Decidability and complexity of petri net problems - an introduction. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer, 1996.

8 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *Logic in Computer Science, 1999. Proceedings. 14th Symposium on*, pages 352–359. IEEE, 1999.

9 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 124–140. Springer, 2013.

10 Alain Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

11 Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. Expand, enlarge, and check: New algorithms for the coverability problem of wsts. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS*, volume 3328 of *Lecture Notes in Computer Science*, pages 287–298. Springer, 2004.

12 Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. On the efficient computation of the minimal coverability set of petri nets. *Int. J. Found. Comput. Sci.*, 21(2):135–165, 2010.

13 Steven M German and A Prasad Sistla. Reasoning about systems with many processes. *Journal of the ACM (JACM)*, 39(3):675–735, 1992.

14 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Dynamic cutoff detection in parameterized concurrent programs. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 645–659. Springer, 2010.

15 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.

16 R.J. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976. Available online at `http://www.cs.yale.edu/publications/techreports/tr63.pdf`.

17 Rupak Majumdar and Zilong Wang. Expand, enlarge, and check for branching vector addition systems. In D'Argenio and Melgratti [4], pages 152–166.

18 Artturi Piipponen and Antti Valmari. Constructing minimal coverability sets. In Parosh Aziz Abdulla and Igor Potapov, editors, *RP*, volume 8169 of *Lecture Notes in Computer Science*, pages 183–195. Springer, 2013.

19 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978.

20 Pierre-Alain Reynier and Frédéric Servais. Minimal coverability set for petri nets: Karp and miller algorithm with pruning. In Lars Michael Kristensen and Laure Petrucci, editors, *Petri Nets*, volume 6709 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.

21 Sylvain Schmitz and Philippe Schnoebelen. The power of well-structured systems. In D'Argenio and Melgratti [4], pages 5–24.

22 Antti Valmari and Henri Hansen. Old and new algorithms for minimal coverability sets. In Serge Haddad and Lucia Pomello, editors, *Petri Nets*, volume 7347 of *Lecture Notes in Computer Science*, pages 208–227. Springer, 2012.

Semi-algebraic geometry in computational game theory – a consumer's perspective*

Peter Bro Miltersen

Aarhus University, Aarhus, Denmark
pbmiltersen@cs.au.dk

Abstract

We survey recent applications of real algebraic and semi-algebraic geometry in (computational) game theory.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems

Keywords and phrases Real Algebraic Geometry, Computational Game Theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.11

Category Invited Talk

1 Introduction to the talk

Real algebraic geometry and semi-algebraic geometry are well-established tools in game theory (e.g., [2, 11, 9]). In this talk, we survey recent work applying these tools to (mostly) computational settings of game theory. Examples include:

- A bound on the discount factor for which the value of a discounted stochastic game is guaranteed to well-approximate the value of the corresponding undiscounted stochastic game [6, 7, 5, 10]. The bound is in terms of the combinatorial parameters of the game and is relatively tight. This refines work of Milman [9].
- An analysis of a recursive bisection algorithm for solving stochastic games [6, 7].
- A tight upper bound on the worst case complexity of the strategy iteration algorithm for concurrent reachability games [6, 7, 5].
- An existence proof of "monomial" near-optimal strategies for concurrent reachability games [4].
- Approximating the value of a concurrent reachability game can be done in the polynomial time hierachy [8, 3].
- Computational (polynomial-time) equivalence between approximating a Nash equilibrium and approximating a trembling hand equilibrium of a game in strategic form (joint work with Etessami, Hansen, and Sørensen, in preparation).

The applications rely on generic tools and off-the-shelf theorems of real algebraic and semi-algebric geometry [1]. The talk is therefore given from the perspective of a consumer of real and semi-algebraic geometry and should be accessible to an audience with little or no knowledge of this topic (which is a level of knowledge similar to that of the speaker). We do briefly discuss what kind of improvements of the results might hopefully be obtained in the future by looking under the hood into the beautiful machinery of semi-algebraic geometry.

* The author acknowledges support from The Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation and from the Center for research in the Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council.



© Peter Bro Miltersen;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 11–12



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



References

1 S. Basu, R. Pollack, and M.F. Roy. *Algorithms in Real Algebraic Geometry*. Algorithms and Computation in Mathematics. Springer, 2006.

2 Truman Bewley and Elon Kohlberg. The asymptotic theory of stochastic games. *Mathematics of Operations Research*, 1(3):197–208, 1976.

3 Søren Kristoffer Stiil Frederiksen and Peter Bro Miltersen. Approximating the value of a concurrent reachability game in the polynomial time hierarchy. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *Algorithms and Computation – 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16–18, 2013, Proceedings*, volume 8283 of *Lecture Notes in Computer Science*, pages 457–467. Springer, 2013.

4 Søren Kristoffer Stiil Frederiksen and Peter Bro Miltersen. Monomial strategies for concurrent reachability games and other stochastic games. In Parosh Aziz Abdulla and Igor Potapov, editors, *Reachability Problems – 7th International Workshop, RP 2013, Uppsala, Sweden, September 24–26, 2013 Proceedings*, volume 8169 of *Lecture Notes in Computer Science*, pages 122–134. Springer, 2013.

5 Kristoffer Arnsfelt Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. The complexity of solving reachability games using value and strategy iteration. In Alexander S. Kulikov and Nikolay K. Vereshchagin, editors, *Computer Science – Theory and Applications – 6th International Computer Science Symposium in Russia, CSR 2011, St. Petersburg, Russia, June 14–18, 2011. Proceedings*, volume 6651 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 2011.

6 Kristoffer Arnsfelt Hansen, Michal Koucký, Niels Lauritzen, Peter Bro Miltersen, and Elias P. Tsigaridas. Exact algorithms for solving stochastic games: extended abstract. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6–8 June 2011*, pages 205–214. ACM, 2011.

7 Kristoffer Arnsfelt Hansen, Michal Koucký, Niels Lauritzen, Peter Bro Miltersen, and Elias P. Tsigaridas. Exact algorithms for solving stochastic games. *CoRR*, abs/1202.3898, 2012.

8 Kristoffer Arnsfelt Hansen, Michal Koucky, and Peter Bro Miltersen. Winning concurrent reachability games requires doubly exponential patience. In *24th Annual IEEE Symposium on Logic in Computer Science (LICS'09)*, pages 332–341. IEEE, 2009.

9 Emanuel Milman. The semi-algebraic theory of stochastic games. *Mathematics of Operations Research*, 27(2):401–418, 2002.

10 Peter Bro Miltersen. Discounted stochastic games poorly approximate undiscouted ones. Available at `http://www.daimi.au.dk/~bromille/Papers/note2.pdf`, 2011.

11 Abraham Neyman. Real algebraic tools in stochastic games. In Abraham Neyman and Sylvain Sorin, editors, *Stochastic Games and Applications*, volume 570 of *NATO Science Series C*. Springer, 2003.

A glimpse on constant delay enumeration

Luc Segoufin

INRIA and ENS Cachan

Abstract

We survey some of the recent results about enumerating the answers to queries over a database. We focus on the case where the enumeration is performed with a constant delay between any two consecutive solutions, after a linear time preprocessing.

This cannot be always achieved. It requires restricting either the class of queries or the class of databases. We describe here several scenarios when this is possible.

1998 ACM Subject Classification F.4 Mathematical logic and formal languages

Keywords and phrases Enumeration, constant delay, logic

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.13

Category Invited Talk

1 Introduction

The evaluation of queries is a central problem in database management systems. Given a query q and a database $\mathcal{D}$ the evaluation of q over $\mathcal{D}$ consists in computing the set $q(\mathcal{D})$ of all answers to q on $\mathcal{D}$. The complexity of this problem has been widely studied. However most of the complexity bounds are extrapolated from the boolean case (aka the model checking problem, where the answer to the query is either a "yes" and a "no") and expressed as a function of the sizes of q and $\mathcal{D}$. In this case we know that the model checking problem for first-order queries is PSpace-complete, for conjunctive queries it is NP-complete and that for acyclic conjunctive queries it can be done in polynomial time. For non boolean queries it may be not satisfactory enough to express complexity results just in terms of the sizes of $\mathcal{D}$ and q. A simple observation shows that the set $q(\mathcal{D})$ may be huge, even larger than the database itself, as it can have a number of elements of the form $\|\mathcal{D}\|^l$, where $\|D\|$ is the size of the database and l the arity of the query. The fact that the solution set $q(\mathcal{D})$ may be of size exponential in the query is intuitively not sufficient to make the problem hard, and alternative complexity measures had to be found for query answering. For instance one could consider output-sensitive complexity measures expressed as a function of the sizes of q, $\mathcal{D}$ but also $q(\mathcal{D})$. In this direction, one way to define tractability is to assume that tuples of the query result can be generated one by one with some regularity, for example by ensuring a fixed delay between two consecutive outputs once a necessary precomputation has been done to construct a suitable index structure.

This approach, that considers query answering as an enumeration problem, has deserved some attention over the last few years. In this vein, the best that one can hope for is constant delay, i.e., the delay depends only on the size of q (but not on the size of $\mathcal{D}$). A number of query evaluation problems have been shown to admit constant delay algorithms, usually preceded by a preprocessing phase that is linear in the size of the database. We survey some of these results in this paper.

This imposes drastic constraints. In particular, the first answer is output after a time linear in the size of the database and once the enumeration starts a new answer is being



© Luc Segoufin;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 13–27



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



output regularly at a speed independent from the size of the database. Altogether, the set $q(\mathcal{D})$ is entirely computed in time $f(q)(\|\mathcal{D}\| + |q(\mathcal{D})|)$ for some function f depending only on q and not on $\mathcal{D}$. In particular, for boolean queries, the model checking problem can be solved in time linear in the size of the database. However, as shown in [4], the fact that evaluation of boolean queries is easy does not guarantee the existence of such efficient enumeration algorithms in general: under some reasonable complexity assumption, there is no constant delay algorithm with linear preprocessing enumerating the answers of acyclic conjunctive queries, although it is well-known that the model-checking of boolean acyclic queries can be done in linear time [45].

We stress that our study is theoretical. If most of the algorithms we will mention here are linear in the size of the database, the multiplicative factors are often very big, making any practical implementation difficult. However, we believe that the index structures designed for making these algorithms work are interesting and, with extra assumptions, could possibly be turned into something practical.

The first part of the paper, Section 3, is devoted to conjunctive queries. We will see how acyclicity plays here a crucial role.

We will then move on to first-order queries in Section 4. In this case we need to restrict the class of databases. We will see that constant delay algorithms can be obtained over classes of databases with bounded degree, bounded treewidth, bounded expansion and low degree.

In Section 5 we will see that, in the bounded treewidth case, one can even enumerate monadic second-order queries with constant delay.

There are many related problems. Typically one could imagine computing the top-ℓ most relevant answers relative to some ranking function or to provide a sampling of $q(\mathcal{D})$ relative to some distribution. One could also imagine computing only the number of solutions $|q(\mathcal{D})|$ or providing an efficient test for whether a given tuple belongs to $q(\mathcal{D})$ or not. It is not clear a priori how these problems are related to constant delay enumeration. However, it turns out that in the scenarios where constant delay enumeration can be achieved, one can often also count the number of solutions in time linear in the size of the database and, after linear time preprocessing on the database, one can test in constant time whether a given tuple is part of the answers set. We will not survey those results here, the interested reader is referred to [41].

This survey is by no means exhaustive. It is only intended to survey the major theoretical results concerning database querying and enumeration. Hopefully it will convince the reader that this is an important subject for research that still contains many interesting and challenging open problems.

2 Preliminaries

2.1 Database as finite relational structures, queries

In this paper a database is a finite relational structure. All interesting examples can be found over graphs or colored graphs. Hence the reader can safely replace relational structure with graph while reading this paper.

A *relational signature* is a tuple $\sigma = (R_1, \ldots, R_l)$, each R_i being a relational symbol of arity r_i. A *relational structure* over σ is a tuple $\mathcal{D} = \left(D, R_1^{\mathcal{D}}, \ldots, R_l^{\mathcal{D}}\right)$, where D is the *domain* of $\mathcal{D}$ and $R_i^{\mathcal{D}}$ is a subset of D^{r_i}. We define the *size* of $\mathcal{D}$ as $\|\mathcal{D}\| = |\sigma| + |D| + \sum_{R_i} |R_i^{\mathcal{D}}| r_i$. It corresponds to the size of a reasonable encoding of $\mathcal{D}$. The number of elements in the domain of $\mathcal{D}$ is denoted by $|\mathcal{D}|$.

A *query* is a computable function associating to a database $\mathcal{D}$ a relation over the domain of $\mathcal{D}$. In this paper, a query takes as input a database of a given signature σ and returns a relation of a fixed arity, *the arity of the query*. A query is a *sentence* if its arity is 0. The query is then either true or false on $\mathcal{D}$ and defines a property of $\mathcal{D}$. A query is *unary* if its arity is 1. If q is a query and $\bar{a}$ is in the image of q on $\mathcal{D}$, then we write $\mathcal{D} \models q(\bar{a})$. Finally we set $q(\mathcal{D}) = \{\bar{a} \mid \mathcal{D} \models q(\bar{a})\}$. Note that the size of $q(\mathcal{D})$ may be exponential in the arity of q. A query language is a class of queries. Typically it is defined as a logical formalism such as CQ (for *conjunctive* queries), FO (for *first-order* queries), MSO (for *monadic second-order* queries) and so on. As usual, $|q|$ denotes the size of q.

Given a query language $\mathcal{L}$, the *model checking problem for* $\mathcal{L}$ is the computational problem of given a **sentence** $q \in \mathcal{L}$ and a database $\mathcal{D}$, to test whether $\mathcal{D} \models q$ or not. The database $\mathcal{D}$ is often restricted to a class $\mathcal{C}$ of finite structures. In this case we speak of *the model checking problem for* $\mathcal{L}$ *over* $\mathcal{C}$.

2.2 Model of computation

We use Random Access Machines (RAM) with addition and uniform cost measure as a model of computation, cf. [1]. Our algorithms will take as input a query q of size k and a database $\mathcal{D}$ of size n. We then say that an algorithm runs in *linear time* (respectively, *quasi-linear* or *constant time*) if it outputs the solution within $f(k)n$ steps (respectively, $f(k)n \log n$ steps or $f(k)$ steps), for some function f

Given an $n \times n$ matrix, and two numbers $i, j \leq n$ the RAM model returns the content to the entry (i, j) of the matrix in constant time. Therefore when given the adjacency matrix of a graph it can test in constant time where two given nodes are adjacent or not. However our databases are encoded by the list of their tuples and we therefore do not have access to the adjacency matrix. Testing whether a tuple belongs to a relation may therefore require more than a constant time.

In the sequel we assume that the input structure comes with a linear order on the domain. If not, we use the one induced by the encoding of the structure as an input to the RAM. Whenever we iterate through all nodes of the domain, the iteration is with respect to the initial linear order.

An important observation is that the RAM model can sort m elements of size $O(\log m)$ in time $O(m \log m)$ [28]. In particular, we can sort lexicographically the tuples of a relation in linear time. As a consequence, a simple merge-sort algorithm we can compute the relation $\{\bar{x}\bar{y} \mid R(\bar{x}\bar{y}) \wedge S(\bar{x})\}$ in time linear in the sizes of R and S.

2.3 Parametrized complexity

The database $\mathcal{D}$ and the query q play different roles as input of our problems. It is often assumed that $|\mathcal{D}|$ is large while $|q|$ is small. Hence it is useful to distinguish them in the input of the query answering problem. Parametrized complexity is a suitable framework for analyzing such situations. We only provide here the basics of parametrized complexity needed for understanding this paper. The interested reader is referred to the monograph [24].

In parametrized complexity, a problem is an input together with a parameter, as a number computable from the input, and a question. A typical example is the parametrized model checking problem for $\mathcal{L}$ where the input is a database $\mathcal{D}$ and a sentence $q \in \mathcal{L}$, the parameter is $|q|$ and the problem asks whether $\mathcal{D} \models q$.

A parametrized problem is Fixed Parameter Tractable, i.e. can be solved in FPT, if, on input of size n and parameter k, it can be solved in time $f(k)n^c$ for some suitable computable

function f and constant c. The idea behind this definition is that for many scenarios, like query answering in databases, it is preferable to have an algorithm working in $2^k n^2$ rather than n^k.

In parametrized complexity there is also a suitable notion of reduction, called FPT-reduction. FPT is closed under FPT-reductions and there are some hard classes of parametrized problems, closed under FPT-reductions, containing problems with no known FPT algorithms and that are believed to be different from FPT. In parametrized complexity, completeness relative to a complexity class is always understood to be under FPT-reductions.

An important hard class is denoted W[1]. W[1] plays in parametrized complexity the role of NP in classical complexity. A typical problem which is complete for W[1] is the parametrized model checking problem for CQ [39]. Another important hard class is denoted AW[$*$]. It plays in parametrized complexity the role of PSpace in classical complexity. A typical problem which is complete for AW[$*$] is the parametrized model checking problem for FO [39].

2.4 The enumeration class CD∘Lin

Let $\mathcal{L}$ be a query language and $\mathcal{C}$ be a class of databases. We say that the *enumeration problem* for $\mathcal{L}$ over $\mathcal{C}$ can be solved with *constant delay after linear preprocessing* (is in CD∘Lin), if it can be solved by a RAM algorithm which, on input $q \in \mathcal{L}$ and $\mathcal{D} \in \mathcal{C}$, can be decomposed into two phases:

- a preprocessing phase that is performed in linear time, and
- an enumeration phase that outputs $q(\mathcal{D})$ with no repetition and a delay depending only on q between any two consecutive outputs. The enumeration phase has full access to the output of the preprocessing phase and can use extra memory whose size depends only on q.[1]

The definition of CD∘Lin requires a preprocessing time linear in $\|\mathcal{D}\|$ and a delay not depending on $\mathcal{D}$. There are hidden multiplicative factors that are function on the size of the query. These factors may be huge. We will refer to them in the sequel as the *multiplicative factors*.

Before we proceed with the technical presentation of the results, it is worth spending some time with examples.

▶ **Example 1.** Consider a database schema containing a binary relational symbol R and the query $q(x, y) := \neg R(x, y)$. On input $\mathcal{D}$, the following simple algorithm enumerates $q(\mathcal{D})$:
Go through all pairs (a, b); test if it is a fact of $R^{\mathcal{D}}$; if so skip this pair; otherwise output it.
However, a simple complexity analysis shows that the delay between any two outputs is not constant. There are two reasons for this. First, arbitrarily long sequences of pairs can be skipped. Second, it is not obvious how to test whether $(a, b) \in R^{\mathcal{D}}$ in constant time (i.e. without going through the whole relation $R^{\mathcal{D}}$). In order to enumerate this query with constant delay it is necessary to perform a preprocessing. We first decide on an arbitrary

[1] In the literature one can sometimes find a more liberal definition only requiring constant time delay with no constraints on the memory. Of course this implies that between two consecutive outputs the memory used is constant, but the global memory affected could be linear in the total number of outputs. In our more constrained setting the enumeration algorithm is essentially a finite state automaton running over the index structure produced during the precomputation phase. It turns out that most of the existing enumeration algorithms do satisfy the extra constraint on memory.

linear order on the domain of $\mathcal{D}$. We then order all $R^{\mathcal{D}}$ according to the lexicographical order. Recall that with the RAM model this can be done in linear time. We then compute for each tuple $\bar{u}$ of $R^{\mathcal{D}}$ the tuples $\bar{v} = f(\bar{u})$ and $\bar{v}' = g(\bar{u})$ such that $\bar{v}$ is the smallest (relative to the lexicographical order) element $\bar{w} \notin R^{\mathcal{D}}$ such that all tuples between $\bar{u}$ and $\bar{w}$ are in $R^{\mathcal{D}}$ and $\bar{v}'$ is the smallest (relative to the lexicographical order) element $\bar{w} \in R^{\mathcal{D}}$ bigger than $\bar{v}$. These functions can be computed in linear time by a simple pass on the ordered list of $R^{\mathcal{D}}$ from its last element to the first one. This concludes the preprocessing phase. The enumeration phase is now simple. We maintain two pairs of elements of $\mathcal{D}$: one is initialized with the smallest pair according to the lexicographical order, the other one with the smallest pair in $R^{\mathcal{D}}$. The second pair will always be pointing to an element of $R^{\mathcal{D}}$. Assuming the current pairs are $\langle \bar{u}, \bar{v} \rangle$, we then do the following until $\bar{u}$ is maximal. If $\bar{u} = \bar{v}$ then we move to $\langle f(\bar{v}), g(\bar{v}) \rangle$. Note that $f(\bar{v}) \neq g(\bar{v})$. If $\bar{u} \neq \bar{v}$ we output $\bar{u}$ and replace it by its successor in the lexicographical order without changing $\bar{v}$. This algorithm is clearly constant delay as an output is performed at least every other step. All output tuples are clearly not in $R^{\mathcal{D}}$ and the reader can check that all skipped tuples are in $R^{\mathcal{D}}$.

▶ **Example 2.** Same schema but the query is now computing the pairs of nodes at distance 2: $q(x, y) := \exists z R(x, z) \wedge R(z, y)$. We will see in Section 3 that it is likely that this query cannot be enumerated with constant delay. However, if we assume that R has degree bounded by d, then for any node a of the graph, at most d^2 nodes v are at distance 2 from u. Moreover, it is easy to see that we can compute in linear time the function $f(u)$ associating to u the list of its nodes at distance 1. An extra linear pass based on the function f computes the function $g(u)$ associating to u the list of its nodes at distance 2. From there the enumeration algorithm with constant delay is trivial.

▶ Remark. *Notice that if the enumeration problem for $\mathcal{L}$ over $\mathcal{C}$ is in* CD∘Lin*, then all answers can be output in time $O(\|\mathcal{D}\| + |q(D)|)$ and the first output is computed in time linear in $\|\mathcal{D}\|$. In particular the model checking problem for $\mathcal{L}$ over $\mathcal{C}$ is in* FPT*. Hence if the model checking problem for $\mathcal{L}$ over $\mathcal{C}$ is known to be* W[1]*-hard, then the enumeration problem for $\mathcal{L}$ over $\mathcal{C}$ cannot be in* CD∘Lin*, unless* W[1] =FPT*.*

Notice that if the arity of q is less or equal to 1, then $|q(\mathcal{D})| \leq |\mathcal{D}| \leq \|\mathcal{D}\|$. It is then plausible that the whole set of answers can be computed in time linear in $\|\mathcal{D}\|$. If this is the case then we have a simple constant delay algorithm that precomputes all answers during the precomputation phase and then scans the set of answers and outputs them one by one during the enumeration phase. Hence enumeration becomes relevant when the arity of q is at least 2. In this case $q(\mathcal{D})$ can be quadratic in $\|\mathcal{D}\|$ and hence can certainly not be computed within the linear time constraint of the precomputation phase. The index structure built during the preprocessing phase is then a non trivial object. One can also view this index structure as a compact (of linear size) representation of the set $q(\mathcal{D})$ (that can be of polynomial size) and the enumeration algorithm as an output streaming decompression algorithm.

▶ Remark. *One difficulty for obtaining constant delay enumeration algorithms is that the class* CD∘Lin *is not known to be closed under boolean operations. Closure under disjunction is difficult because of the requirement that each solution must be output only once. There are two particular cases when closure under disjunction can be obtained. The first one is trivial: It assumes that we have* CD∘Lin *algorithms for q and q' over a class $\mathcal{C}$ of databases and that, on input $\mathcal{D} \in \mathcal{C}$, both algorithms output the answers relative to the same linear order on all tuples (for instance the lexicographical order). In this case a simple argument that resembles the problem of merging two sorted lists gives a* CD∘Lin *algorithm for $q \vee q'$ over $\mathcal{C}$. The second case is more subtle. Instead of assuming a linear order on the output tuples, it*

assumes that after preprocessing in time linear in $\|\mathcal{D}\|$, given a tuple $\bar{a}$, one can test whether $\mathcal{D} \models q(\bar{a})$ in constant time. Then there is a CD∘Lin *algorithm for $q \vee q'$ over $\mathcal{C}$ [42].*

3 Restricting the queries

In this section we consider the evaluation of simple queries over the class of all databases.

3.1 Conjunctive queries

A conjunctive query (CQ) is a query of the form

$$q(\bar{x}) := \exists y_1 \cdots y_l \quad \bigwedge_i R_i(\bar{z}_i)$$

where $R_i(\bar{z}_i)$ is an *atom* of q, R_i being a relational symbol and $\bar{z}_i$ containing variables from $\bar{x}$ or $\bar{y}$. A typical example is the distance 2 query of Example 2 in in CQ. Another example is the query returning all triangles in a graph. The model checking problem for CQ is W[1]-complete and we therefore restrict our attention to *acyclic* conjunctive queries (ACQ) that can be evaluated in time $|q| \cdot \|\mathcal{D}\| \cdot |q(\mathcal{D})|$ [45]. We will see that it is very unlikely that constant delay enumeration can be achieved for ACQ. It is only achieved for a subset of ACQ called *free-connex*. We start with the necessary definitions.

A *join tree* of $q \in$ CQ is a tree T whose nodes are atoms of q and such that

(i) each atom of q is the label of exactly one node of T,

(ii) for each variable x of q, the set of nodes of T in which x occurs is connected.

A conjunctive query q is said to be *acyclic* if it has a join tree. In graph theoretical terms this is equivalent to saying that the hypergraph formed by the atoms of q is α-acyclic.

An acyclic conjunctive query $q(\bar{x})$ is said to be *free-connex* if the query $q(\bar{x}) \wedge R(\bar{x})$ where R is a new symbol of appropriate arity, is acyclic.[2] Note that all boolean acyclic query are free-connex.

For example the acyclic conjunctive query $q(x, y) = \exists u, v \;\; S(x, y, u) \wedge T(x, y, v)$ is free-connex because the following join tree shows acyclicity of the extended query:

$$\begin{array}{ccc} & R(x,y) & \\ S(x,y,u) & & T(x,y,v) \end{array}$$

However the distance 2 query $q(x, y) = \exists z \;\; S(x, z) \wedge S(z, y)$ is not free-connex as the query $\exists z \;\; S(x, z) \wedge S(z, y) \wedge R(x, y)$ is clearly cyclic.

▶ Theorem 1. [4] The enumeration for free-connex ACQ over the class of all databases is in CD∘Lin.

We note that the multiplicative factors involved in Theorem [4] are polynomial in the query size.

The result of Theorem 1 also holds if the queries contain inequalities (ACQ$^{\neq}$). In this case atoms with inequalities are not involved when building the (generalized) join trees. In the presence of inequalities, the multiplicative factors are now exponential in the query size.

It turns out that free-connexity characterizes exactly those acyclic queries that can be enumerated in constant delay, assuming boolean matrix multiplication cannot be done in

[2] This is not the initial definition of free-connex as given in [4]. This presentation is from Brault-Baron [9].

quadratic time. Boolean matrix multiplication is the problem of given two $n \times n$ matrices with boolean entries M, N to compute their product MN. The best known algorithms so far (based on the Coppersmith–Winograd algorithm [12]) require more than $n^{2.37}$ steps.

▶ Theorem 2. [4] If boolean matrix multiplication cannot be done in quadratic time then the following are equivalent for $q \in$ ACQ:

1. q is free-connex
2. q can be enumerated in CD$\circ$Lin
3. q can be evaluated in time $O(\|\mathcal{D}\| + |q(\mathcal{D})|)$.

In particular the distance 2 query cannot be enumerated with constant delay after linear time preprocessing unless boolean matrix multiplication can be done in quadratic time.

▶ Remark. Theorem 2 is based on the complexity assumption that boolean matrix multiplication cannot be done in quadratic time. Another hypothesis yielding the same result was provided by [9]. This hypothesis requires that it is not possible to test the existence of a triangle in a graph of n vertices in time $O(n^2)$ and that for any k testing the presence of a k-dimensional tetrahedron cannot be tested in linear time (see [9] for precise definitions).

3.2 Signed conjunctive queries

We are now interested in evaluating signed conjunctive queries (SCQ). Those extends the syntax of conjunctive queries by allowing negated atoms. In other words they are of the form

$$q(\bar{x}) := \exists \bar{y} \quad q^+(\bar{x}\bar{y}) \wedge q^-(\bar{x}\bar{y})$$

where q^+ is a conjunction of positive atoms whiles q^- is a conjunction of negated atoms.

When q^- is empty we have seen in the previous section that q can be enumerated with constant delay after a linear preprocessing as soon as q^+ is α-acyclic. When q^+ is empty it has been shown in [8, 9] that constant delay enumeration can be achieved if q^- is *β-acyclic*. β-acyclicity is the hereditary extension of α-acyclicity. It requires that the hypergraph and all its subhypergraphs are α-acyclic. When neither q^+ nor q^- are empty then a notion of *signed-acyclicity* was introduced in [9]. It yields α-acyclicity and β-acyclicity in the corresponding limits case. It also allows for tractable enumeration algorithms.

▶ Theorem 3. [9] The enumeration for signed-acyclic SCQ over the class of all databases can be done with constant delay after a preprocessing time of the form $\|\mathcal{D}\|(\log \|\mathcal{D}\|)^{|q|}$.

The enumeration for signed-acyclic SCQ over the class of all databases can be done with logarithmic delay after a quasi-linear time preprocessing.

The multiplicative factors are exponential in the size of the query for the constant delay result but polynomial in the logarithmic delay result. As in the ACQ case, modulo complexity hypothesis, typically that testing the existence of a triangle cannot be done in $O(n^2 \log n)$ time on a graph of size n, the signed-acyclicity hypothesis cannot be avoided [9].

3.3 Guarded First-Order Queries

Guarded first-order formulas (GFO) are defined using the following grammar.

$$\phi ::= \quad R(\bar{x}) \mid x = y \mid \phi \wedge \phi \mid \neg\phi(x) \mid \exists \bar{x}\alpha(\bar{x}\bar{y}) \wedge \phi(\bar{x}\bar{y}) \mid \forall \bar{x}\alpha(\bar{x}\bar{y}) \rightarrow \phi(\bar{x}\bar{y})$$

where R is an arbitrary relation symbol from the schema and $\alpha(\bar{x}\bar{y})$ is an atom containing all variables in $\bar{x}\bar{y}$. See [27] for more details about guarded logics. It has been shown in [5] that the model checking for sentences from GFO could be done in linear time. This can be

extended to a constant delay algorithm assuming acyclicity of the *quantifier-free part* of the query.

Indeed consider a subformula γ of the form $\exists \bar{x}\alpha(\bar{x}\bar{y}) \wedge \phi(\bar{x}\bar{y})$ where ϕ is quantifier free. It defines a relation $R_\gamma(\bar{y})$ whose size is linear in the size of the relation occurring in α. A simple argument as the one for $R(\bar{x}\bar{y}) \wedge S(\bar{x})$ explained in Section 2.2 shows that this relation can be computed in linear time.

Therefore, after a linear preprocessing, any GFO query can be transformed into a quantifier free one. Turned into DNF the resulting query is a union of SCQ. By a simple exclusion-inclusion argument the union can be assumed to give disjoint results. Hence it remains to enumerate each disjunct separately. From Theorem 3 this can be done efficiently if each disjunct is signed-acyclic.

This suggest the following definition. Given a GFO query q, it's *quantifier-free part* is the quantifier free query constructed from q by pushing negation down to the atoms and then replacing its maximal subformula $\gamma(\bar{y})$ of the form $\exists \bar{x}\alpha(\bar{x}\bar{y}) \wedge \phi(\bar{x}\bar{y})$ by $R_\gamma(\bar{y})$. It's *normalized quantifier-free part* further transforms the quantifier-free part by putting it into DNF and applying the exclusion-inclusion principle to get disjoint conjunctive formulas.

Let's denote by *signed-acyclic* GFO those queries of GFO whose *normalized quantifier-free part* are such that each conjunct is signed-acyclic. From the previous argument and Theorem 3 the following result follows:

▶ Theorem 4. The enumeration for signed-acyclic GFO over the class of all databases can be done with logarithmic delay after a quasi-linear preprocessing time.

▶ Remark. The same result can probably be obtained with a more natural syntactic fragment of GFO.

4 Restricting the class of structures

In this section we consider first-order queries (FO) and restrict the classes of databases to sparse structures. All these classes are defined over graphs and are generalized to arbitrary relational structures via their Gaifman graphs: Given a class $\mathcal{C}$ of graphs, the associated class $\mathcal{C}$' of databases contains exactly all the databases whose Gaifman graphs are in $\mathcal{C}$.

The *Gaifman graph* of a relational structure $\mathcal{D}$ is defined as follows: the set of vertices is the domain D of $\mathcal{D}$ and there is an edge (a, b) iff there exists a relation R_i and a tuple $t \in R_i$ such that both a and b occur in t. For a graph G we denote by $|G|$ its number of vertices and by $||G||$ its number of edges.

4.1 Bounded degree

A class of graphs has *bounded degree* if there exists a d such that all nodes of all graphs in the class have at most d neighbors. It is known that the model checking problem for FO over structures with bounded degree can be solved in linear time [40].

▶ Theorem 5. [18, 33] The enumeration for FO over a class of structures with bounded degree is in CD∘Lin.

The initial proof of [18] is using the fact that structures in a class of graphs of bounded degree can be encoded using finitely bijective unary functions. Moreover, over such structures, there exists a quantifier elimination method for FO formulas [18]. Once the query is quantifier free, it is not too difficult to design for it a constant delay enumeration algorithm.

Another idea is to use the Gaifman Locality Theorem showing that for FO queries only the r-neighborhoods (i.e. substructures of all nodes at distance at most r) occurring in the

structures are relevant, for a suitable value of r depending only on the query. In a class of graphs with bounded degree, there are only finitely many such r-neighborhoods and it is possible to compute them in linear time, hence during the preprocessing phase. The enumeration algorithm follows [33].

The multiplicative factors are a tower of exponential whose height depends on $|q|$ in the case of [18] and are triply exponential in $|q|$ in the case of [33]. This latter multiplicative factor cannot be significantly improved: it follows from [26] that a multiplicative factor only doubly exponential in the size of the formula is not possible unless AW[$*$] =FPT.

4.2 Bounded expansion

The bounded degree case can be generalized to a larger class of structures known as *bounded expansion* and defined in [37]. In [37] a number of equivalent characterizations were given for bounded expansion giving evidence that this class is robust. Many known families of graphs have bounded expansion. We list below some notable examples.

- Class of graphs with bounded degree.
- Class of graphs with bounded treewidth.
- Class of planar graphs.
- Class of graphs excluding at least one minor.

The model checking problem for FO over classes of structures with bounded expansion can be solved in linear time [21, 30].

▶ Theorem 6. [34] The enumeration for FO over the class of structures with bounded expansion is in CD$\circ$Lin.

This result generalizes the bounded degree case. If structures in a class of bounded degree could be represented using finitely many unary bijections, structures in a class of bounded expansion can be represented using finitely many unary functions of a special kind. A quantifier elimination method is then given over such structures. However solving the quantifier-free case is not immediate.

The multiplicative factors are a tower of exponentials whose height is the quantifier alternation depth of the first-order query. This non-elementary multiplicative factor is unavoidable already on the class of unranked trees, assuming FPT $\neq$ AW[$*$] [26]. In comparison, recall that this factor is triply exponential in the size of the query over bounded degree structures.

4.3 Nowhere dense

It turns out that the notion of bounded expansion can be further generalized. A class $\mathcal{C}$ of graphs is *nowhere dense* if for all r there exists a graph H_r that is not a r-minor of all graphs of $\mathcal{C}$ (a r-minor is a minor where the collapsed balls have radius at most r).

This class was introduced in [38] with a number of equivalent characterizations giving evidence that it is a robust class. It contains all class of graphs of bounded expansion but also any class of graphs that locally excludes a minor or that has local bounded treewidth. We refer to [17, 25] for precise definitions of these notions.

It has recently been claimed that the model checking problem for FO over nowhere dense graphs can be done in quasi-linear time [31].

▶ Open problem 1. Can enumeration for FO over the class of nowhere dense graphs be done with constant delay after a quasi-linear time preprocessing?

If the class of graphs is closed under subgraphs, nowhere dense is the limit for the existence of FPT algorithms.

▶ Theorem 7. [22] If $\mathcal{C}$ is a somewhere dense class of graphs closed under subgraphs, then the model checking problem for FO over this class is W[1]-hard (actually existential formula suffices).

An even stronger result was obtained in [36] assuming that $\mathcal{C}$ is somewhere dense in an "effective way". In this case it is shown that the model-checking for FO is even AW[$*$]-complete.

4.4 Low Degree

For classes of graphs not closed under subgraphs, we can still obtain positive results over a somewhere dense class of graphs.

A class of graphs has low degree if for all δ, all but finitely many graphs in the class have degree at most n^δ, where n is the size of the graph. Typical examples are structures of bounded degree or of degree bounded by $\log n$.

It has been proved in [29] that over a class of structures of low degree, first-order boolean queries can be checked in pseudo-linear time, i.e. in time bounded by $O(n^{1+\epsilon})$, for all $\epsilon > 0$. This can be extended to an efficient enumeration algorithm assuming that sufficient memory is available. The result below assumes that the computation starts with an initial memory of $O(n^3)$ on input of size n. It will use only a small fragment of this memory, as it runs in pseudo-linear time, but for reasons detailed in [19], it requires initially more.

▶ Theorem 8. [19] The enumeration for FO over a class of structures of low degree can be done with constant delay after a pseudo-linear preprocessing time.

5 Structures with bounded treewidth

We have seen in Section 4.2 that structures of bounded treewidth are a special case of structures of bounded expansion. Therefore, over such classes, FO queries can be enumerated with constant delay after a linear time preprocessing. Over structures of bounded treewidth, the enumeration result can be extended to a larger class of queries: MSO queries. It is well known that the associated model checking problem can be solved in linear time by Courcelle's theorem [13].

Recall that MSO extends FO with the possibility to quantify existentially and universally over monadic second order variables. Those variables range over sets of elements of the input domain. By MSO query we mean here a query of the form $q(\bar{x})$ where q is in MSO and $\bar{x}$ are first-order free variables. The case where $\bar{x}$ can also contain free monadic variables has also been considered in [14, 2] but those cannot be enumerated in CD$\circ$Lin mainly because outputting one solution may require linear time. See Section 6.

However, when restricted to first-order free variables, constant delay enumeration can be achieved. Two different index structures were proposed in the literature. Actually a third one was also proposed in [14], but it requires a precomputation phase of $O(n \log n)$ to build it.

▶ Theorem 9. [2, 35] The enumeration for MSO queries over the class of structures with bounded treewidth is in CD$\circ$Lin.

The difficulty of Theorem 9 lies entirely in the tree case. We present the key ingredients of the proof of [35] below as the intermediate results are of independent interest.

Let L be a regular word language over an alphabet $\mathbb{A}$. A typical binary MSO query over trees is the query $q_L(x, y)$ returning the pairs of nodes (u, v) within a tree such that u is an ancestor of v and the labels of the nodes in the path from u to v forms a word in L.

Given a tree $\mathbf{t}$, there exists an index structure computable in time linear in $\|\mathbf{t}\|$ such that, given two nodes u and v of $\mathbf{t}$ one can test in constant time whether $(u, v) \in q_L(\mathbf{t})$ or not. This is a nontrivial and powerful result of Colcombet based on deep algebraic constructions.

▶ Proposition 10. [11] For any regular language L over an alphabet $\mathbb{A}$ and any $\mathbb{A}$-labeled tree $\mathbf{t}$ one can

- construct in time $O(\|\mathbf{t}\|)$ an index structure such that,
- for all nodes u, v of $\mathbf{t}$, testing whether $(u, v) \in q_L(\mathbf{t})$ can be done in constant time.

The multiplicative factors resulting from the construction of the index and during the constant time tests depend on the presentation of L. They are non elementary if L is given as an MSO sentence. They are exponential if L is given as an automaton, even in the deterministic case (see also [6]). However, there exist cases where these multiplicative factors are polynomial (see for instance [7]).

The index structure built for proving Proposition 10 has the following interesting normal form for MSO queries over trees as a consequence.

▶ Proposition 11. [implicit in [11], see also [10]] Over trees, every binary MSO query $q(x, y)$ is equivalent to a disjunction of queries of the form $\exists \bar{y} \forall \bar{z}\ \theta$, where θ is a disjunction of conjunctions of atomic predicates, ancestor relationships, or **unary** MSO queries.

The index constructed in [35] for enumerating MSO queries over trees builds on Proposition 11. The so called "composition method", or a simple Ehrenfeucht-Fraïssé game, shows that any MSO query is equivalent to a boolean combination of binary queries. For binary queries, Proposition 11 applies. The unary MSO subformulas can be precomputed in linear time by Courcelle's theorem and can therefore be considered as new colors. Hence it is enough to consider $\exists \bar{y} \forall \bar{z}$ first-order queries using the ancestor relationship. Those queries being rather simple, an induction on the number of free variables solves the problem, see [35]. The multiplicative factors of Theorem 9 deviates from those of Proposition 10 only by a polynomial factor. Hence their size depends on the presentation of the MSO query as explained above.

6 Discussion

6.1 The impact of order

With the current definition of CD∘Lin, there is no constraint on the order in which the answers are output. One could require a specific order, relevant to the context in which the query is evaluated. For instance, if there is a linear order on the domain of the database, one could require that the tuples of the result are output in lexicographical order. Another typical example is when there is a relevance measure associated to each tuple and one would like the answers to the query to be output in the order of their relevance.

Requiring a specific order when outputting the answers to a query may have a dramatic impact on the existence of constant delay algorithms. This is not surprising as the index built during the preprocessing phase is designed for a particular order.

In the presence of a linear order on the database, the enumeration algorithms of Theorem 5 (bounded degree) and Theorem 6 (bounded expansion) can output the solutions in lexicographical order. However, it is not clear how to achieve lexicographical output in the case of MSO over bounded treewidth (Theorem 9).

6.2 Longer delay

Delay linear in the size of the database

We could consider enumeration algorithms allowing for non constant delay. We have already seen logarithmic delays in Theorem 3 and Theorem 4. Another interesting case is linear delay. In this setting, the preprocessing phase remains linear in the size of the database but the delay between any two consecutive outputs is now linear in the size of the database. Notice that linear delay still implies that the associated model checking problem is in FPT, hence CQ cannot be enumerated with linear delay unless W[1] =FPT.

One can then consider restricting the class of structures. A class of structures, called $\underline{\mathrm{X}}$-structures, has been exhibited such that CQ can be enumerated over it with linear delay. We will not define $\underline{\mathrm{X}}$-structures in this note. Typical examples are grids and trees with all XPath axis.

▶ Theorem 12. [3]. The enumeration for CQ over $\underline{\mathrm{X}}$-structures can be done with linear delay.

For acyclic conjunctive queries linear delay enumeration can be obtained with no restriction on the structures.

▶ Theorem 13. [4]. The enumeration for ACQ over all structures can be done with linear delay.

Delay linear in the size of the output

A trivial case when constant delay enumeration cannot be achieved is when the size of one output is too big. This is for instance the case when considering MSO formulas with monadic second-order free variables. Then each answer is a tuple of sets of elements of the domain and can have a size linear in the size of the database. In constant time such an answer can not even be written in the output tape. For such queries it is convenient to allow a delay linear in the size of the output, but still independent from the size of the database[3]. We then speak of an output-linear delay.

The result of Theorem 9 can be generalized to this setting (the preprocessing phase of [14] is quasi-linear while it is linear in the case of [2]).

▶ Theorem 14. [14][2] The enumeration for MSO (allowing monadic second-order free predicates) over the class of structures with bounded treewidth can be done with output-linear delay.

Polynomial delay

One could also allow polynomial precomputation and polynomial delay. This notion is maybe less relevant in the database context. Indeed, the degree of the polynomial could depend on the size of the query and in this case the preprocessing phase can often precompute all solutions. This notion is however relevant when considering first-order queries with free second-order variables. In this case, for Σ_1-queries, polynomial delay enumeration can be achieved [20].

[3] There is actually another approach which consists in having an output tape and only modify the output tape in order to transform the previous solution into the next one. In special cases the delta between two consecutive solutions only affect a constant part of the output and the enumeration can be done with constant delay, see for instance [20].

6.3 Other enumeration problems

In this abstract we focused on the problem of enumerating the output of a query on a database. There exist also interesting enumeration algorithms for enumerating all the solutions of a SAT instance. For 2SAT, this is in CD∘Lin [23], for 3SAT dichotomy results exists [16, 15]. There exists also enumeration algorithms for various kinds of other problems like enumerating monomials of a polynomial [43], enumerating perfect matchings in bipartite graphs [44], independent sets [32] and so on. The interested reader is refereed to the thesis [42, 9] for learning more about enumerations outside of the database context.

7 Conclusions

We mentioned several results about constant delay enumeration. We hope that we succeeded to convince the reader that this is a very interesting topic.

The main open problem is probably the evaluation of first-order queries over nowhere dense structures mentioned in Open Problem 1.

One could also consider relaxing the "no duplicate" constraint and enumerate conjunctive queries with the "bag semantics", i.e. each answer occurs as many times as there are valuations witnessing it.

We would like to conclude with lower bounds. Of course one can construct artificial problems, based on the fact that there exist quadratic but not linear problems, that do not admit constant delay enumeration algorithms. For the concrete problems mentioned in this note, the lower bounds have been proved using complexity assumptions, either in parametrized complexity, or for the Boolean Matrix Multiplication problem. But it is also plausible (i.e. there are no known consequences in complexity theory nor in algorithmic) that the non existence of constant delay enumeration algorithms could be proved with no assumptions. We believe this is an interesting and challenging question.

Acknowledgment. We thanks Johann Brault-Baron and Thomas Colcombet for useful comments on earlier versions of this paper.

References

1 A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

2 G. Bagan. MSO Queries on Tree Decomposable Structures Are Computable with Linear Delay. In *Conf. on Computer Science Logic (CSL)*, pages 167–181, 2006.

3 G. Bagan, A. Durand, E. Filiot, and O. Gauwin. Efficient Enumeration for Conjunctive Queries over X-underbar Structures. In *Conf. on Computer Science Logic (CSL)*, pages 80–94, 2010.

4 G. Bagan, A. Durand, and E. Grandjean. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *Conf. on Computer Science Logic (CSL)*, pages 208–222, 2007.

5 D. Berwanger and E. Grädel. Games and model checking for guarded logics. In *Intl. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pages 70–84, 2001.

6 M. Bojańczyk. Factorization forests. In *Developments in Language Theory (DLT)*, 2009.

7 M. Bojańczyk and P. Parys. XPath evaluation in linear time. *J. of the ACM*, 58(4), 2011.

8 J. Brault-Baron. A Negative Conjunctive Query is Easy if and only if it is Beta-Acyclic. In *Conf. on Computer Science Logic (CSL)*, pages 137–151, 2012.

9 J. Brault-Baron. *De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre*. PhD thesis, Université de Caen, 2013.

10 T. Colcombet. The factorisation forest theorem. To appear in the handbook "Automata: from Mathematics to Applications".

11 T. Colcombet. A Combinatorial Theorem for Trees. In *Intl. Coll. on Automata, Languages and Programming (ICALP)*, pages 901–912, 2007.

12 D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions. *J. on Symbolic Computation*, 9(3):251–280, 1990.

13 B. Courcelle. Graph Rewriting: An Algebraic and Logic Approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. 1990.

14 B. Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009.

15 N. Creignou and J.-J. Hébrard. On Generating All Solutions of Generalized Satisfiability Problems. *Informatique Théorique et Applications (ITA)*, 31(6):499–511, 1997.

16 N. Creignou, F. Olive, and J. Schmidt. Enumerating All Solutions of a Boolean CSP by Non-decreasing Weight. In *Theory and Applications of Satisfiability Testing (SAT)*, pages 120–133, 2011.

17 A. Dawar, M. Grohe, and S. Kreutzer. Locally Excluding a Minor. In *Symp. on Logic in Computer Science (LICS)*, pages 270–279, 2007.

18 A. Durand and E. Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. on Computational Logic (ToCL)*, 8(4), 2007.

19 A. Durand, N. Schweikardt, and L. Segoufin. Enumerating first-order queries over databases of low degree. submitted.

20 A. Durand and Y. Strozecki. Enumeration Complexity of Logical Query Problems with Second-order Variables. In *Conf. on Computer Science Logic (CSL)*, pages 189–202, 2011.

21 Z. Dvořák, D. Král, and R. Thomas. Deciding First-Order Properties for Sparse Graphs. In *Symp. on Foundations of Computer Science (FOCS)*, pages 133–142, 2010.

22 Z. Dvořák, D. Král, and R. Thomas. Testing first-order properties for subclasses of sparse graphs. *CoRR*, abs/1109.5036, 2011.

23 T. Feder. Network flow and 2-satisfiability. *Algorithmica*, 11:291–319, 1994.

24 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

25 M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. of the ACM*, 48(6):1184–1206, 2001.

26 M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004.

27 E. Grädel. On the restraining power of guards. *J. on Symbolic Logic*, 64(4):1719–1742, 1999.

28 E. Grandjean. Sorting, Linear Time and the Satisfiability Problem. *Annals of Mathematics and Artificial Intelligence*, 16:183–236, 1996.

29 M. Grohe. Generalized model-checking problems for first-order logic. In *Symp. on Theoretical Aspects in Computer Science (STACS)*, 2001.

30 M. Grohe and S. Kreutzer. *Model Theoretic Methods in Finite Combinatorics*, chapter Methods for Algorithmic Meta Theorems. American Mathematical Society, 2011.

31 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. personal communication.

32 D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.

33 W. Kazana and L. Segoufin. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science (LMCS)*, 7(2), 2011.

34 W. Kazana and L. Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. *Symp. on Principles of Database Systems (PODS)*, 2013.

35 W. Kazana and L. Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. on Computational Logic (ToCL)*, 14(4), 2013.

36 S. Kreutzer and A. Dawar. Parameterized complexity of first-order logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:131, 2009.

37 J. Nešetřil and P. O. de Mendez. Grad and classes with bounded expansion I. Decompositions. *Eur. J. Comb.*, 29(3):760–776, 2008.

38 J. Nešetřil and P. O. de Mendez. On nowhere dense graphs. *European J. of Combinatorics*, 32(4):600–617, 2011.

39 C. H. Papadimitriou and M. Yannakakis. On the Complexity of Database Queries. *J. on Computer and System Sciences (JCSS)*, 58(3):407–427, 1999.

40 D. Seese. Linear Time Computable Problems and First-Order Descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.

41 L. Segoufin. Enumerating with constant delay the answers to a query. In *Intl. Conf. on Database Theory*, pages 10–20, 2013.

42 Y. Strozecki. *Enumeration complexity and matroid decomposition.* PhD thesis, Université de Paris 7, 2010.

43 Y. Strozecki. Enumeration of the Monomials of a Polynomial and Related Complexity Classes. In *Intl. Symp. on Mathematical Foundations of Computer Science (MFCS)*, pages 629–640, 2010.

44 T. Uno. Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs. In *Intl. Symp. on Algorithms and Computation*, pages 92–101, 1997.

45 M. Yannakakis. Algorithms for Acyclic Database Schemes. In *Intl. Conf. on Very Large Databases (VLDB)*, pages 82–94, 1981.

Arithmetic Circuit Complexity

Neeraj Kayal

Microsoft Research India
neeraka@microsoft.com

Abstract

Arithmetic Circuits compute polynomial functions over their inputs via a sequence of arithmetic operations (additions, subtractions, multiplications, divisions, etc.). This tutorial will give an overview of arithmetic circuit complexity, focusing on the problem of proving lower bounds for arithmetic circuits.

In the first part, we begin with a few nontrivial upper bounds – matrix multiplication and the computation of symmetric polynomials. We then motivate some open problems we deal with in arithmetic circuit complexity. We will look at the problem of polynomial identity testing - motivating it by its application to bipartite matching, the problem of learning arithmetic circuits or circuit reconstruction and the problem of proving lower bounds for arithmetic circuits (motivating it via the problem of computing the permanent and the Hamiltonian polynomials). We will also see depth reduction for circuits – the tradeoffs involved (with respect to size) in squashing a circuit into one with smaller depth.

In the second part, we will see some classical lower bounds. In particular, we will see lower bounds for monotone arithmetic circuits and multilinear formulas. We then give a very quick overview of approaches being investigated (including geometric complexity theory and tau-conjecture) aiming to prove lower bounds.

In the third part, we begin with a warm-up by proving lower bounds for homogeneous depth three circuits. We will then see recent lower bounds for homogeneous depth four circuits and its consequences.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Circuit complexity, arithmetic circuits, lower bounds, polynomial identity testing

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.28

Category Tutorial



© Neeraj Kayal;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 28–28

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

Submodular Stochastic Probing on Matroids

Marek Adamczyk[*1], **Maxim Sviridenko**[2], **and Justin Ward**[3]

1 **Department of Computer, Control, and Management Engineering, Sapienza University of Rome, Italy,** adamczyk@dis.uniroma1.it
2 **Department of Computer Science, University of Warwick, United Kingdom,** M.I.Sviridenko@warwick.ac.uk
3 **Department of Computer Science, University of Warwick, United Kingdom,** J.D.Ward@warwick.ac.uk

Abstract

In a *stochastic probing* problem we are given a universe E, where each element $e \in E$ is *active* independently with probability $p_e \in [0,1]$, and only a *probe* of e can tell us whether it is active or not. On this universe we execute a process that one by one probes elements — if a probed element is active, then we have to include it in the solution, which we gradually construct. Throughout the process we need to obey *inner* constraints on the set of elements taken into the solution, and *outer* constraints on the set of all probed elements. This abstract model was presented by Gupta and Nagarajan [18], and provides a unified view of a number of problems. Thus far all the results in this general framework pertain only to the case in which we are maximizing a linear objective function of the successfully probed elements. In this paper we generalize the stochastic probing problem by considering a monotone submodular objective function. We give a $(1-1/e)/(k^{in}+k^{out}+1)$-approximation algorithm for the case in which we are given $k^{in} \geq 0$ matroids as inner constraints and $k^{out} \geq 1$ matroids as outer constraints. There are two main ingredients behind this result. First is a previously unpublished stronger bound on the continuous greedy algorithm due to Vondrak [22]. Second is a rounding procedure that also allows us to obtain an improved $1/(k^{in}+k^{out})$-approximation for linear objective functions.

1998 ACM Subject Classification F. Theory of Computation

Keywords and phrases approximation algorithms, stochastic optimization, submodular optimization, matroids, iterative rounding

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.29

1 Introduction

Uncertainty in input data is a common feature of most practical problems, and research in finding good solutions (both experimental and theoretical) for such problems has a long history dating back to 1950 [6, 11]. We consider adaptive stochastic optimization problems in the framework of Dean et al. [13]. Here the solution is in fact a process, and the optimal one might even require larger than polynomial space to describe. Since the work of Dean et al. a number of such problems were introduced [10, 14, 15, 16, 4, 17, 12]. Gupta and Nagarajan [18] present an abstract framework for a subclass of adaptive stochastic problems giving a unified view for Stochastic Matching [10] and Sequential Posted Pricing [9].

We describe the framework following [18]. We are given a universe E, where each element $e \in E$ is *active* with probability $p_e \in [0,1]$ independently. The only way to find out if an

* Supported by the ERC StG project PAAl no. 259515, and by NCN grant N N206 567940.

© Marek Adamczyk, Maxim Sviridenko, and Justin Ward;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 29–40
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

element is active, is to *probe* it. We call a probe *successful* if an element turns out to be active. On universe E we execute an algorithm that probes the elements one-by-one. If an element is active, the algorithm must add it to the current solution. In this way, the algorithm gradually constructs a solution consisting of active elements.

Here, we consider the case in which we are given constraints on both the elements probed and the elements included in the solution. Formally, suppose that we are given two independence systems of downward-closed sets: an *outer* independence system $(E, \mathcal{I}^{out})$ restricting the set of elements probed by the algorithm, and an *inner* independence system $\left(E, \mathcal{I}^{in}\right)$, restricting the set of elements taken by the algorithm. We denote by Q^t the set of elements probed in the first t steps of the algorithm, and by S^t the subset of active elements from Q^t. Then, S^t is the partial solution constructed by the first t steps of the algorithm. We require that, at each time t, $Q^t \in \mathcal{I}^{out}$ and $S^t \in \mathcal{I}^{in}$. Thus, at each time t, the element e that we probe must satisfy both $Q^{t-1} \cup \{e\} \in \mathcal{I}^{out}$ and $S^{t-1} \cup \{e\} \in \mathcal{I}^{in}$. Gupta and Nagarajan [18] considered many types of systems $\mathcal{I}^{in}$ and $\mathcal{I}^{out}$, but we focus only on matroid intersections, i.e. on the special case in which $\mathcal{I}^{in}$ is an intersection of k^{in} matroids $\mathcal{M}_1^{in}, \ldots, \mathcal{M}_{k^{in}}^{in}$, and $\mathcal{I}^{out}$ is an intersection of k^{out} matroids $\mathcal{M}_1^{out}, \ldots, \mathcal{M}_{k^{out}}^{out}$. We always assume that $k^{out} \geq 1$ and $k^{in} \geq 0$. We assume familiarity with matroid algorithmics (see [20], for example) and, above all, with principles of approximation algorithms (see [21], for example).

Considering submodular objective functions is a common practice in combinatorial optimization as it extends the range of applicability of many methods. So far, the framework of stochastic probing has been used to maximize the expected weight of the solution found by the process. We were given weights $w_e \geq 0$ for $e \in E$ and, if S denotes the solution at the end of a process, the goal was to maximize $\mathbb{E}_S\left[\sum_{e \in S} w_e\right]$. We generalize the framework as we consider a monotone submodular function $f : 2^E \mapsto \mathbb{R}_{\geq 0}$, and objective of maximizing $\mathbb{E}_S\left[f\left(S\right)\right]$.

1.1 Our results

Our result is a new algorithm for stochastic probing problem based on iterative randomized rounding of linear programs and the *continuous greedy process* introduced by Calinescu et al. [8].

▶ **Theorem 1.** *An algorithm based on the continuous greedy process and iterative randomized rounding is a* $\frac{\left(1-e^{-1}\right)}{k^{in}+k^{out}+1}$*-approximation for stochastic probing problem with monotone submodular objective function.*

Additionally, we improve the bound of $\frac{1}{4(k^{in}+k^{out})}$ given by Gupta and Nagarajan [18] in the case of a linear objective.

▶ **Theorem 2.** *The iterative randomized rounding algorithm is a* $\frac{1}{k^{in}+k^{out}}$*-approximation for the stochastic probing problem with a linear objective function.*

1.2 Applications

On-line dating and kidney exchange [10]

Consider an online dating service. For each pair of users, machine learning algorithms estimate the probability that they will form a happy couple. However, only after a pair meets do we know for sure if they were successfully matched (and together leave the dating service). Users have individual patience numbers that bound how many unsuccessful dates

they are willing to go on until they will leave the dating service forever. The objective of the service is to maximize the number of successfully matched couples.

To model this as a stochastic probing problem, users are represented as vertices V of a graph $G = (V, E)$, where edges represent matched couples. Set E of edges is our universe on which we make probes, with p_e being the probability that a couple $e = (u_1, u_2)$ forms a happy couple after a date. The inner constraints are matching constraints — a user can be in at most one couple —, and outer constraints are b–matching — we can probe at most $t(u)$ edges adjacent to user u, where $t(u)$ denotes the patience of u. Both inner and outer constraints are intersections of two matroids for bipartite graphs. In similar way we can model kidney exchanges.

In weighted bipartite case Theorem 2 gives a $1/4$-approximation. Even though b–matchings in general graphs are not intersections of two matroids, we are able to exploit the matching structure to give the same factor-$1/4$ approximation. Since the technique is very similar to the case of intersection of two matroids, we omit the proof. This matches the current-best bound for general graphs of Bansal et al. [5], who also give a $1/3$-approximation in the bipartite case.

Bayesian mechanism design [18]

Consider the following mechanism design problem. There are n agents and a single seller providing a certain service. Agent's i value for receiving service is v_i, drawn independently from a distribution D_i over set $\{0, 1, \ldots, B\}$. The valuation v_i is private, but the distribution D_i is known. The seller can provide service only for a subset of agents that belongs to system $\mathcal{I} \in 2^{[n]}$, which specifies feasibility constraints. A mechanism accepts bids of agents, decides on subset of agents to serve, and sets individual prices for the service. A mechanism is called truthful if agents bid their true valuations. Myerson's theory of virtual valuations yields *truthful* mechanisms that maximize the expected revenue of a seller, although they sometimes might be impractical. On the other hand, practical mechanisms are often non-truthful. The Sequential Posted Pricing Mechanism (SPM) introduced by Chawla et al. [9] gives a nice trade-off — it is truthful, simple to implement, and gives near-optimal revenue. An SPM offers each agent a "take-it-or-leave-it" price for the service. Since after a refusal a service won't be provided, it is easy to see that an SPM is a truthful mechanism.

To see an SPM as a stochastic probing problem, we consider a universe $E = [n] \times \{0, 1, \ldots, B\}$, where element (i, c) represents an offer of price c to agent i. The probability that i accepts the offer is $\mathbb{P}[v_i \geq c]$, and seller earns c then. Obviously, we can make only one offer to an agent, so outer constraints are given by a partition matroid; making at most one probe per agent also overcomes the problem that probes of $(i, 1), ..., (i, B)$ are not independent. The inner constraints on universe $[n] \times \{0, 1, \ldots, B\}$ are simply induced by constraints $\mathcal{I}$ on $[n]$.

Gupta and Nagarajan [18] give an LP relaxation for any single-seller Bayesian mechanism design problem. Provided that we can optimize over $\mathcal{P}(\mathcal{I})$, the LP can be used to construct an efficient SPM. Moreover, the approximation guarantee of the constructed SPM is with respect to the optimal mechanism, which need not be an SPM.

In the case constraints $\mathcal{I}$ are an intersection of k matroids the resulting SPM is a $\frac{1}{4(k+1)}$-approximation [18]. Here, we give an improved approximation algorithm with a factor-$\frac{1}{k+1}$ guarantee. In particular, when $k = 1$ we match [9, 19] with $1/2$-approximation.

1.3 Related work

The stochastic matching problem with applications to online dating and kidney exchange was introduced by Chen et al. [10], where authors proved a 1/4-approximation of a greedy strategy for unweighted case. The authors also show that the simple greedy approach gives no constant approximation in the weighted case. Their bound was later improved to 1/2 by Adamczyk [1]. As noted in our discussion of applications, Bansal et al. [5] gave 1/3 and 1/4-approximations for weighted stochastic matching in bipartite and general graphs, respectively.

Sequential Posted Pricing mechanisms were investigated first by Chawla et al. [9], followed by Yan [24], and Kleinberg and Weinberg [19]. Gupta and Nagarajan [18] were first to propose looking at SPM from the point of view of stochastic adaptive problems.

Asadpour et al. [4] were first to consider a stochastic adaptive problem with submodular objective function. In our terms, they considered only a single outer matroid constraint.

Work of Calinescu et al. [8] provides the tools for submodular functions we use in this paper. The method of [24] was based on "correlation gap" [3], something we address implicitly in Subsection 2.2.2.

2 Preliminaries

For set $S \subseteq E$ and element $e \in E$ we use $S + e$ to denote $S \cup \{e\}$, and $S - e$ to denote $S \setminus \{e\}$. For set $S \subseteq E$ we shall denote by $\mathbf{1}_S$ a characteristic vector of set S, and for a single element e we shall write $\mathbf{1}_e$ instead of $\mathbf{1}_{\{e\}}$. For random event $\mathcal{A}$ we shall denote by $\chi[\mathcal{A}]$ a 0-1 random variable that indicates whether $\mathcal{A}$ occurred. The optimal strategy will be denoted by OPT, and we shall denote the expected objective value of its outcome as $\mathbb{E}[OPT]$.

2.1 Matroids and polytopes

Let $\mathcal{M} = (E, \mathcal{I})$ be a matroid, where E is the universe of elements and $\mathcal{I} \subseteq 2^E$ is a family of independent sets. For element $e \in E$, we shall denote the matroid $\mathcal{M}$ with e contracted by $\mathcal{M}/e$, i.e. $\mathcal{M}/e = (E - e, \{S \subseteq E - e \,|\, S + e \in \mathcal{I}\})$.

The following lemma is a slightly modified[1] basis exchange lemma, which can be found in [20].

▶ **Lemma 3.** *Let $A, B \in \mathcal{I}$ and $|A| = |B|$. There exists a bijection $\phi : A \mapsto B$ such that: 1) $\phi(e) = e$ for every $e \in A \cap B$, 2) $B - \phi(e) + e \in \mathcal{I}$.*

We shall use the following corollary, where we consider independent sets of possibly different sizes.

▶ **Corollary 4.** *Let $A, B \in \mathcal{I}$. We can find assignment $\phi_{A,B} : A \mapsto B \cup \{\perp\}$ such that:*
1. *$\phi_{A,B}(e) = e$ for every $e \in A \cap B$,*
2. *for each $f \in B$ there exists at most one $e \in A$ for which $\phi_{A,B}(e) = f$,*
3. *for $e \in A \setminus B$, if $\phi_{A,B}(e) = \perp$ then $B + e \in \mathcal{I}$, otherwise $B - \phi_{A,B}(e) + e \in \mathcal{I}$.*

We consider optimization over *matroid polytopes* which have the general form $\mathcal{P}(\mathcal{M}) = \left\{x \in \mathbb{R}^E_{\geq 0} \,\middle|\, \forall_{A \in \mathcal{I}} \sum_{e \in A} x_e \leq r_{\mathcal{M}}(A)\right\}$, where $r_{\mathcal{M}}$ is the rank function of $\mathcal{M}$. We know [20]

[1] The difference is that we do not assume that A, B are bases, but independent sets of the same size.

that the matroid polytope $\mathcal{P}(\mathcal{M})$ is equivalent to the convex hull of $\{\mathbf{1}_A \,|\, A \in \mathcal{I}\}$, i.e. characteristic vectors of all independent sets of $\mathcal{M}$. Thus, we can represent any $x \in \mathcal{P}(\mathcal{M})$ as $x = \sum_{i=1}^{m} \beta_i \cdot \mathbf{1}_{B_i}$, where $B_1, \ldots, B_m \in \mathcal{I}$ and $\beta_1, \ldots, \beta_m$ are non-negative weights such that $\sum_{i=1}^{m} \beta_i = 1$. We shall call sets $B_1, \ldots, B_m$ a *support* of x in $\mathcal{P}(\mathcal{M})$.

2.2 Submodular functions

2.2.1 Multilinear extension

A set function $f : 2^E \mapsto \mathbb{R}_{\geq 0}$ is *submodular*, if for any two subsets $S, T \subseteq E$ we have $f(S \cup T) + f(S \cap T) \leq f(S) + f(T)$. We call function f *monotone*, if for any two subsets $S \subseteq T \subseteq E : f(S) \leq f(T)$. For a set $S \subseteq E$, we let $f_S(A) = f(A \cup S) - f(S)$ denote the marginal increase in f when the set A is added to S. Note that if f is monotone submodular, then so is f_S for all $S \subseteq E$. Moreover, we have $f_S(\emptyset) = 0$ for all $S \subseteq E$, so f_S is normalized. Without loss of generality, we assume also that $f(\emptyset) = 0$.

We consider the *multilinear extension* $F : [0,1]^E \mapsto \mathbb{R}_{\geq 0}$ of f, whose value at a point $y \in [0,1]^E$ is given by

$$F(y) = \sum_{A \subseteq E} f(A) \prod_{e \in A} y_e \prod_{e \notin A} (1 - y_e).$$

Note that $F(\mathbf{1}_A) = f(A)$ for any set $A \subseteq E$, so F is an extension of f from discrete domain 2^E into a real domain $[0,1]^E$. The value $F(y)$ can be interpreted as the expected value of f on a random subset $A \subseteq E$ that is constructed by taking each element $e \in E$ with probability y_e. Following this interpretation, Calinescu et al. [8] show that $F(y)$ can be estimated to any desired accuracy in polynomial time, using a sampling procedure.

Additionally, they show that F has the following properties, which we shall make use of in our analysis:

▶ **Lemma 5.** *The multilinear extension F is linear along the coordinates, i.e. for any point $x \in [0,1]^E$, any element $e \in E$, and any $\xi \in [-1,1]$ such that $x + \xi \cdot \mathbf{1}_e \in [0,1]^E$, it holds that $F(x + \xi \cdot \mathbf{1}_e) - F(x) = \xi \cdot \frac{\partial F}{\partial y_e}(x)$, where $\frac{\partial F}{\partial y_e}(x)$ is the partial derivative of F in direction y_e at point x.*

▶ **Lemma 6.** *If $F : [0,1]^E \mapsto \mathbb{R}$ is a multilinear extension of monotone submodular function $f : 2^E \mapsto \mathbb{R}$, then 1) function F has second partial derivatives everywhere; 2) for each $e \in E$, $\frac{\partial F}{\partial y_e} \geq 0$ everywhere; 3) for any $e_1, e_2 \in E$ (possibly equal), $\frac{\partial^2 F}{\partial y_{e_1} \partial y_{e_2}} \leq 0$, which means that $\frac{\partial F}{\partial y_{e_2}}$ is non-increasing with respect to y_{e_1}.*

2.2.2 Continuous greedy algorithm

In [8] the authors utilized the multilinear extension in order to maximize a submodular monotone function over a matroid constraint. They showed that a *continuous greedy algorithm* finds a $(1 - 1/e)$-approximate maximum of the above extension F over any downward closed polytope. In the special case of the matroid polytope, they show how to employ the pipage rounding [2] technique to the fractional solution to obtain an integral solution.

Another extension of f studied in [7] is given by:

$$f^+(y) = \max \left\{ \sum_{A \subseteq E} \alpha_A f(A) \,\middle|\, \sum_{A \subseteq E} \alpha_A \leq 1,\ \forall A \subseteq E : \alpha_A \geq 0,\ \forall j \in E : \sum_{A : j \in A} \alpha_A \leq y_j \right\}.$$

Intuitively, the solution $(\alpha_A)_{A\subseteq E}$ above represents the distribution over 2^E that maximizes the value $\mathbb{E}[f(A)]$ subject to the constraint that its marginal values satisfy $\mathbb{P}[i \in A] \leq y_i$. The value $f^+(y)$ is then the expected value of $\mathbb{E}[f(A)]$ under this distribution, while the value of $F(y)$ is the value of $\mathbb{E}[f(A)]$ under the particular distribution that places each element i in A independently. However, the following allows us to relate the value of F on the solution of the continuous greedy algorithm to the optimal value of the relaxation f^+.

▶ **Lemma 7.** *Let f be a submodular function with multilinear extension F, and let $\mathcal{P}$ be any downward closed polytope. Then, the solution $x \in \mathcal{P}$ produced by the continuous greedy algorithm satisfies $F(x) \geq (1-1/e)\max_{y\in\mathcal{P}} f^+(y)$.*

This follows from a simple modification of the continuous greedy analysis, given by Vondrák [22].

2.3 Overview of the iterative randomized rounding approach

We now give a description of the general rounding approach that we employ in both the linear and submodular case. In each case, we formulate a mathematical programming relaxation of the following general form

$$\max_{x\in[0,1]^E} \left\{ g(x) \,\middle|\, \forall j \in \left[k^{in}\right] : p\cdot x \in \mathcal{P}\left(\mathcal{M}_j^{in}\right); \forall j \in \left[k^{out}\right] : x \in \mathcal{P}\left(\mathcal{M}_j^{out}\right) \right\} \tag{1}$$

with $p \in [0,1]^E$ being the vector of probabilities. Here $g : [0,1]^E \mapsto \mathbb{R}_{\geq 0}$ is an objective function chosen so that the optimal value of (1) can be used to bound the expected value of an optimal policy for the given instance using the following lemma. Note that our program will always have constraints as given in (1), only the objective function g changes between the linear and monotone submodular cases.

▶ **Lemma 8.** *Let OPT be the optimal feasible strategy for some stochastic probing problem in our general setting, and define $x_e = \mathbb{P}[OPT \text{ probes } e]$. Then, $x = (x_e)_{e\in E}$ is a feasible solution to the related relaxation of the form* (1).

Proof. Since OPT is a feasible strategy, the set of elements Q probed by any execution of OPT is always an independent set of each outer matroid $\mathcal{M} = \left(E, \mathcal{I}_j^{out}\right)$, i.e. $\forall_{j\in[k^{out}]} Q \in \mathcal{I}_j^{out}$. Thus, for any $j \in [k^{out}]$, the vector $\mathbb{E}[\mathbf{1}_Q] = x$ may be represented as a convex combination of vectors from $\left\{\mathbf{1}_A \,\middle|\, A \in \mathcal{I}_j^{out}\right\}$, and hence $x \in \mathcal{P}\left(\mathcal{M}_j^{out}\right)$. Analogously, the set of elements S that were successfully probed by OPT satisfy $\forall_{j\in[k^{in}]} S \in \mathcal{I}_j^{in}$ for every possible execution of OPT. Hence, for any $j \in \left[k^{in}\right]$ the vector $\mathbb{E}[\mathbf{1}_S] = p\cdot x$ may be represented as a convex combination of vectors from $\left\{\mathbf{1}_A \,\middle|\, A \in \mathcal{I}_j^{in}\right\}$, and hence $x \in \mathcal{P}\left(\mathcal{M}_j^{in}\right)$. ◀

Suppose that f is the objective function for a given instance of stochastic probing over a universe E of elements. Our algorithm first obtains a solution x^0 to a relaxation of the form (1) using either linear programming or the continuous greedy algorithm. Our algorithm proceeds iteratively, maintaining a current set of constraints, a current fractional solution x, and a current set S of elements that have been successfully probed. Initially, the constraints are as given in (1), $x = x^0$, and $S = \emptyset$. At each step, the algorithm selects single element $\bar{e}$ to probe, then permanently sets $x_{\bar{e}}$ to 0. It then updates the outer constraints, replacing $\mathcal{M}_j^{out}$ with $\mathcal{M}_j^{out}/\bar{e}$ for each $j \in [k^{out}]$. If the probe succeeds, the algorithm adds $\bar{e}$ to S and then updates the inner constraints, replacing $\mathcal{M}_j^{in}$ with $\mathcal{M}_j^{in}/\bar{e}$ for each $j \in [k^{in}]$. Finally, we modify our fractional solution x so that it is feasible for the updated constraints. The algorithm terminates when the current solution $x = 0^E$.

In order to analyze the approximation performance of our algorithm, we keep track of a current potential value z, related to the value of the remaining fractional solution x. Let x^t, z^t, and S^t be the current value of x, z, and S at the beginning of step $t+1$. We show that initially we have $z^0 = g\left(x^0\right) \geq \beta \cdot \mathbb{E}\left[OPT\right]$ for some constant $\beta \in [0,1]$, and then analyze the expected decrease $z^t - z^{t+1}$ at an arbitrary step $t+1$. We show that for each step we have $\alpha \cdot \mathbb{E}\left[z^t - z^{t+1}\right] \leq \mathbb{E}\left[f(S^{t+1}) - f(S^t)\right]$, for some $\alpha < 1$. That is, the expected increase in the value of the current solution is at least α times the expected decrease in z. Then, we employ the following Lemma to conclude that the algorithm is an $\alpha\beta$-approximation in expectation. The proof is based on Doob's optional stopping theorem for martingales. Hence, we need to deploy language from martingale theory, such as stopping time and filtration. See [23] for extended background on martingale theory.

▶ **Lemma 9.** *Suppose the algorithm runs for τ steps and that $z^0 = g\left(x^0\right) \geq \beta \cdot \mathbb{E}\left[OPT\right]$, $z^\tau = 0$. Let $(\mathcal{F}_t)_{t\geq 0}$ be the filtration associated with our iterative algorithm, where $\mathcal{F}_i$ represents all information available after the ith iteration. Finally, suppose that in each step in our iterative rounding procedure, $\mathbb{E}\left[f(S^{t+1}) - f(S^t)\middle|\mathcal{F}_t\right] \geq \alpha \cdot \mathbb{E}\left[z^t - z^{t+1}\middle|\mathcal{F}_t\right]$. Then, the final solution S^τ produced by the algorithm satisfies $\mathbb{E}\left[f(S^\tau)\right] \geq \alpha\beta \cdot \mathbb{E}\left[OPT\right]$.*

Proof. Let G_{t+1} be the gain $f(S^{t+1}) - f(S^t)$ in f at step $t+1$, and let L_{t+1} be the corresponding loss $z^t - z^{t+1}$ in z at time $t+1$. We set $G_0 = L_0 = 0$. Define variable $D_t = G_t - \alpha \cdot L_t$. The sequence of random variables $(D_0 + D_1 + ... + D_t)_{t\geq 0}$ forms a sub-martingale, i.e.

$$\mathbb{E}\left[\sum_{i=0}^{t+1} D_i \middle| \mathcal{F}_t\right] = \sum_{i=0}^{t} D_i + \mathbb{E}\left[G_{t+1} - \alpha \cdot L_{t+1}\middle|\mathcal{F}_t\right] \geq \sum_{i=0}^{t} D_i.$$

Let τ be the step in which the algorithm terminates, i.e. $\tau = \min\left\{t \middle| x^t = 0^E\right\}$. Then, the event $\tau = t$ depends only on $\mathcal{F}_0, \ldots, \mathcal{F}_t$, so τ is a stopping time. Also, by the definition of the algorithm $x^\tau = 0^E$. It is easy to verify that all the assumptions of Doob's optional stopping theorem are satisfied, and from this theorem we get that $\mathbb{E}\left[\sum_{i=0}^{\tau} D_i\right] \geq \mathbb{E}\left[D_0\right]$. Since $D_0 = 0$, we have

$$0 \leq \mathbb{E}\left[\sum_{i=0}^{\tau} D_i\right] = \mathbb{E}\left[\sum_{i=0}^{\tau} G_i - \alpha \cdot \sum_{i=0}^{\tau} L_i\right] = \mathbb{E}\left[\sum_{i=0}^{\tau} G_i\right] - \alpha \cdot \mathbb{E}\left[\sum_{i=0}^{\tau} L_i\right].$$

It remains to note that $\sum_{i=0}^{\tau} G_i = f\left(S^\tau\right)$ is the total gain of the algorithm, so $\mathbb{E}\left[\sum_{i=0}^{\tau} G_i\right] = \mathbb{E}\left[f(S^\tau)\right]$. On the other hand, $\sum_{i=0}^{\tau} L_i = g(x^0) - g(x^\tau) = g(x^0) \geq \beta \cdot \mathbb{E}\left[OPT\right]$. ◀

Henceforth, we will implicitly condition on all information $\mathcal{F}_t$ available to the algorithm just before it makes step $t+1$. That is, when discussing step $t+1$ of the algorithm, we write shortly $\mathbb{E}\left[\cdot\right]$ instead of $\mathbb{E}\left[\cdot\middle|\mathcal{F}_t\right]$.

3 Linear stochastic probing

In this setting, we are given a weight w_e and a probability p_e for each element $e \in E$ and $f(S)$ is simply $\sum_{e\in S} w_e$. We consider the relaxation (1) in which $g(x) = f(x)$. Then, Lemma 8 shows that the optimal policy OPT must correspond to some feasible solution x^* of (1). Moreover, because f is linear, $\mathbb{E}\left[OPT\right] = \sum_{e\in S} \mathbb{P}\left[OPT \text{ probes e}\right] p_e w_e = \sum_{e\in S} x_e p_e w_e = f(x^*)$.

At each step, our algorithm randomly selects an element $\bar{e}$ to probe. Let $\Sigma = \sum_{e\in E} x_e$ Then, our algorithm chooses element e with probability x_e/Σ. As discussed in the previous

overview, it then sets $x_{\bar{e}} = 0$ and carries out the probe, updating the matroid constraints to reflect both the choice of $\bar{e}$ and the probe. Finally, it updates x to obtain a new fractional solution that is feasible in the updated constraints. Note that because x_e is set to 0 after probing e, we will never probe an element e twice.

Let us now describe how to update the current solution x to ensure feasibility in each of the updated matroid constraints. Let $\bar{e}$ be the element that we probed and let $\mathcal{M}_j^{out}$ be some outer matroid. Currently we have $x \in \mathcal{P}(\mathcal{M}_j^{out})$ and we must obtain a solution x' so that $x' \in \mathcal{P}(\mathcal{M}_j^{out}/\bar{e})$. We represent the vector x as a convex combination of independent sets $x = \sum_{i=1}^{m} \beta_i^{out} \mathbf{1}_{B_i^{out}}$, where $B_1^{out}, \ldots, B_m^{out}$ is the support of x with respect to matroid $\mathcal{M}_j^{out}$. We obtain $x' \in \mathcal{P}(\mathcal{M}_j^{out}/\bar{e})$ by replacing each independent set B_b^{out} for which $B_b^{out} + \bar{e} \notin \mathcal{M}_j^{out}$ with some other set B_c^{out} such that $B_c^{out} + \bar{e} \in \mathcal{M}_j^{out}$. We pick one set B_a^{out} with $\bar{e} \in B_a^{out}$ to guide the update process. We pick the set $B_a^{out} \ni \bar{e}$ at random with probability $\beta_a^{out}/x_{\bar{e}}$ (note that for any element e, $\sum_{a:e\in B_a^{out}} \beta_a^{out} = x_e$). For any set $B_b^{out} : \bar{e} \notin B_b^{out}$, let $\phi_{a,b}$ be the mapping from B_a^{out} into B_b^{out} from Corollary 4. If $\phi_{a,b}(\bar{e}) = \perp$, or $\phi_{a,b}(\bar{e}) = \bar{e}$, then in fact $B_b^{out} + \bar{e} \in \mathcal{M}_j^{out}$, and we can just include B_b^{out} in the support of $\mathcal{M}_j^{out}/\bar{e}$. Otherwise, we substitute B_b^{out} with $B_b^{out} - \phi_{a,b}(\bar{e})$ in the support of $(x_e)_{e\in E}$ in $\mathcal{P}\left(\mathcal{M}_j^{out}/\bar{e}\right)$, since we know that $B_b^{out} - \phi_{a,b}(\bar{e}) + \bar{e} \in \mathcal{M}_j^{out}$.

Similarly, if $\bar{e}$ is successfully probed we must perform a support update for each inner matroid. Here, we proceed as in the case of the outer matroids, except we have $p \cdot x \in \mathcal{M}_j^{in}$ and must obtain x' such that $p \cdot x' \in \mathcal{M}_j^{in}/e$. We write $p \cdot x$ as a combination independent sets $p \cdot x = \sum_{i=1}^{m} \beta_i^{in} \mathbf{1}_{B_i^{in}}$, and now choose a random set $B_a^{in} \ni \bar{e}$ to guide the support update with probability $\beta_a^{in}/p_{\bar{e}}x_{\bar{e}}$. (note that for any element e, we have $\sum_{a:e\in B_a^{in}} \beta_a^{in} = p_e x_e$). As in the previous case, we replace B_b^{in} with $B_b^{in} - \phi_{a,b}(\bar{e})$ for each base B_b^{in} such that $B_b^{in} + \bar{e} \notin \mathcal{M}_j^{in}$.

We now turn to the analysis of the probing algorithm. Suppose that the algorithm runs for τ steps and consider the quantity $z^t = f(x^t)$. Then, $z^0 = f(x^0) \geq \mathbb{E}[OPT]$ and $z^\tau = f(0^E) = 0$, so the conditions of Lemma 9 are satisfied with $\beta = 1$. It remains to bound the expected loss $\mathbb{E}\left[z^t - z^{t+1}\right]$ in step $t+1$. In order to do this, we consider the value $\delta_i = p_i(x_i^t - x_i^{t+1})$ for each $i \in E$. We consider arbitrary step $t+1$, but we are going to denote x^t by x and x^{t+1} by x'. The decrease δ_i may be caused both by the probing step, in which we set $x'_{\bar{e}}$ to 0, or by the matroid update step, in which we decrease several coordinates of x. Let us first consider the losses due to each matroid update.

▶ **Lemma 10.** *Let x and x' be the current fractional solution before and after one update for a given outer matroid $\mathcal{M}_j^{out}$. Then, for each $i \in E$, we have $\mathbb{E}[\delta_i^{out}] \triangleq \mathbb{E}[p_i(x_i - x'_i)] \leq \frac{1}{\Sigma}(1 - x_i)\, p_i x_i$.*

Proof. The expectation $\mathbb{E}[\delta_i^{out}]$ is over the random choice of an element $\bar{e}$ to probe and the random choice of an independent set to guide the update. Let $\mathcal{E}_a^{out}$ denote the event that some set B_a^{out} is chosen to guide a support update for $\mathcal{M}_j^{out}$.

In a given step the probability that the set B_a^{out} is chosen to guide the support update is equal to

$$\mathbb{P}\left[\mathcal{E}_a^{out}\right] = \sum_{e\in B_a^{out}} \frac{x_e}{\Sigma} \frac{\beta_a^{out}}{x_e} = \sum_{e\in B_a^{out}} \frac{\beta_a^{out}}{\Sigma} = \left|B_a^{out}\right| \frac{\beta_a^{out}}{\Sigma}. \tag{2}$$

Moreover, conditioned on the fact B_a^{out} was chosen, the probability that an element $e \in B_a^{out}$ was probed is uniform over the elements of B_a^{out}:

$$\mathbb{P}\left[e \text{ probed} \mid \mathcal{E}_a^{out}\right] = \mathbb{P}\left[e \text{ probed} \wedge \mathcal{E}_a^{out}\right] / \mathbb{P}\left[\mathcal{E}_a^{out}\right] = \frac{x_e}{\Sigma} \frac{\beta_a^{out}}{x_e} \Big/ \left|B_a^{out}\right| \frac{\beta_a^{out}}{\Sigma} = \frac{1}{|B_a^{out}|}. \tag{3}$$

We can write the expected decrease as $\mathbb{E}\left[\delta_i^{out}\right] = \sum_{a=1}^{m} \mathbb{P}\left[\mathcal{E}_a^{out}\right] \cdot \mathbb{E}\left[\delta_i^{out} \middle| B_a^{out}\right]$. Note that for all $i \in B_a^{out}$, we have $\phi_{a,b}(i) = i$ for every set $B_b^{out} \ni i$. Thus, the support update will not change x_i for any $i \in B_a^{out}$, and so $\sum_{a=1}^{m} \mathbb{P}\left[\mathcal{E}_a^{out}\right] \cdot \mathbb{E}\left[\delta_i^{out} \middle| \mathcal{E}_a^{out}\right] = \sum_{a:i \notin B_a^{out}} \mathbb{P}\left[\mathcal{E}_a^{out}\right] \cdot \mathbb{E}\left[\delta_i^{out} \middle| \mathcal{E}_a^{out}\right]$.

Now let us condition on taking B_a^{out} to guide the support update. Consider a set $B_b^{out} \ni e$. If we remove i from B_b^{out}, and hence decrease $p_i x_i$ by $p_i \beta_b^{out}$, it must be the case that we have chosen to probe the single element $\phi_{a,b}^{-1}(i) \in B_a^{out}$. The probability that we probe this element is $\frac{1}{|B_a^{out}|}$. Hence

$$\begin{aligned}
&\sum_{a:i \notin B_a^{out}} \mathbb{P}\left[\mathcal{E}_a^{out}\right] \cdot \mathbb{E}\left[\delta_i^{out} \middle| B_a^{out}\right] \\
&= \sum_{a:i \notin B_a^{out}} \mathbb{P}\left[\mathcal{E}_a^{out}\right] \cdot \left(\sum_{b:i \in B_b^{out}} p_i \beta_b^{out} \cdot \mathbb{P}\left[\phi_{a,b}^{-1}(i) \text{ is probed} \middle| \mathcal{E}_a^{out}\right] \right) \\
&\leq \sum_{a:i \notin B_a^{out}} \mathbb{P}\left[\mathcal{E}_a^{out}\right] \cdot \left(\sum_{b:i \in B_b^{out}} p_i \beta_b^{out} \cdot \frac{1}{|B_a^{out}|} \right) \\
&= \sum_{a:i \notin B_a^{out}} \mathbb{P}\left[\mathcal{E}_a^{out}\right] \cdot \frac{p_i x_i}{|B_a^{out}|} \\
&= \sum_{a:i \notin B_a^{out}} \left|B_a^{out}\right| \frac{1}{\Sigma} \beta_a^{out} \cdot \frac{p_i x_i}{|B_a^{out}|} \quad = \quad \frac{1}{\Sigma} \sum_{a:i \notin B_a^{out}} \beta_a^{out} p_i x_i \quad = \quad \frac{1}{\Sigma} (1 - x_i)\, p_i x_i. \qquad \blacktriangleleft
\end{aligned}$$

▶ **Lemma 11.** *Let x be the current fractional solution before and after one update for a given inner matroid $\mathcal{M}_j^{in}$. Then, for each $i \in E$, we have $\mathbb{E}\left[\delta_i^{in}\right] \triangleq \mathbb{E}\left[p_i(x_i - x_i')\right] \leq \frac{1}{\Sigma}(1 - p_i x_i)\, p_i x_i$.*

Proof. Because we only perform a support update when the probe of a chosen element is successful, the expectation $\mathbb{E}\left[\delta_i^{in}\right]$ is over the random result of the probe, as well as the random choice of element $\bar{e}$ to probe and the random choice of a base to guide the update. We proceed as in the case of Lemma 10, now letting $\mathcal{E}_a^{in}$ denote the event that the probe was successful and B_a^{in} is chosen to guide the support update. We have:

$$\mathbb{P}\left[\mathcal{E}_a^{in}\right] = \sum_{e \in B_a^{in}} p_e \frac{x_e}{\Sigma} \frac{\beta_a^{in}}{p_e x_e} = \sum_{e \in B_a^{in}} \frac{\beta_a^{in}}{\Sigma} = \left|B_a^{in}\right| \frac{\beta_a^{in}}{\Sigma},$$

$$\mathbb{P}\left[e \text{ probed} \mid \mathcal{E}_a^{in}\right] = \mathbb{P}\left[e \text{ probed} \wedge \mathcal{E}_a^{in}\right] / \mathbb{P}\left[\mathcal{E}_a^{in}\right] = p_e \frac{x_e}{\Sigma} \frac{\beta_a^{in}}{p_e x_e} \Big/ \left|B_a^{in}\right| \frac{\beta_a^{in}}{\Sigma} = \frac{1}{|B_a^{in}|}.$$

By a similar argument as in Lemma 10 we then have that $\mathbb{E}\left[\delta_i^{in}\right]$ is at most:

$$\begin{aligned}
\sum_{a:i \notin B_a^{in}} \mathbb{P}\left[\mathcal{E}_a^{in}\right] \cdot \left(\sum_{b:i \in B_b^{in}} \beta_b^{in} \cdot \frac{1}{|B_a^{in}|} \right) &= \sum_{a:i \notin B_a^{in}} \mathbb{P}\left[\mathcal{E}_a^{in}\right] \cdot \frac{p_i x_i}{|B_a^{in}|} \\
&= \sum_{a:i \notin B_a^{out}} \left|B_a^{in}\right| \frac{1}{\Sigma} \beta_a^{in} \cdot \frac{p_i x_i}{|B_a^{in}|} = \frac{1}{\Sigma} \sum_{a:i \notin B_a^{in}} \beta_a^{in} p_i x_i = \frac{1}{\Sigma}(1 - p_i x_i) p_i x_i.
\end{aligned}$$

◀

We perform the matroid updates sequentially for each of the k^{in} and k^{out} matroids. Note that once we decrease a coordinate x_i to 0, it cannot be altered in any further updates, so no coordinate is ever decreased below 0. Now, we consider the expected decrease

$\mathbb{E}[\delta_i] = \mathbb{E}[p_i(x_i - x'_i)]$ due to both the initial probing step, in which we decrease the probed element's coordinate to 0, and the following matroid updates. We have:

$$\begin{aligned}\mathbb{E}[\delta_i] &\le \mathbb{P}[i \text{ probed}]\, p_i x_i + k^{out}\mathbb{E}\left[\delta_i^{out}\right] + k^{in}\mathbb{E}\left[\delta_i^{in}\right] \\ &= \frac{1}{\Sigma}p_i x_i^2 + k^{out}\frac{1}{\Sigma}(1-x_i)p_i x_i + k^{in}\frac{1}{\Sigma}(1-p_i x_i)p_i x_i \\ &= \frac{1}{\Sigma}k^{out}p_i x_i - \frac{1}{\Sigma}(k^{out}-1)p_i x_i^2 + \frac{1}{\Sigma}k^{in}p_i x_i - \frac{1}{\Sigma}k^{in}p_i^2 x_i^2 \\ &\le \frac{k^{out}+k^{in}}{\Sigma}p_i x_i. \end{aligned} \tag{4}$$

Because z^t is a linear function of x^t, the expected total decrease of z in this step is then

$$\mathbb{E}\left[z^t - z^{t+1}\right] = \sum_i \mathbb{E}[\delta_i]\, w_i \le \frac{k^{out}+k^{in}}{\Sigma}\sum_i p_i x_i w_i.$$

On the other hand, the expected gain in $f(S)$ is $\sum_{e\in E}\mathbb{P}[e \text{ probed}]\, p_e w_e = \frac{1}{\Sigma}\sum_{e\in E} w_e p_e x_e$. Thus, by Lemma 9 the final solution S^τ produced by the algorithm satisfies $\mathbb{E}[f(S^\tau)] \ge \frac{1}{k^{out}+k^{in}}\mathbb{E}[OPT]$.

4 Submodular stochastic probing

We now consider the case in which we are given a set of elements E each becoming active with probability p_e, and we seek to maximize a given submodular function $f : 2^E \mapsto \mathbb{R}_{\ge 0}$. In this case, we consider the relaxation (1) in which $g(x) = f^+(p \cdot x)$. Then, Lemma 8 shows that the optimal policy OPT must correspond to some feasible solution x^* of (1), where $x^*_e = \mathbb{P}[OPT \text{ probes } e]$, and hence $\mathbb{P}[OPT \text{ takes } e] = p_e x^*_e$. The function $f^+(p \cdot x^*)$ gives the maximum value of $\mathbb{E}_{S\sim\mathcal{D}}[f(S)]$ over all distributions $\mathcal{D}$ satisfying $\mathbb{P}_{S\sim\mathcal{D}}[e \in S] = x^*_e p_e$. Thus, $f^+(p \cdot x^*) \ge \mathbb{E}[OPT]$.

In general, we cannot obtain an optimal solution to this relaxation. Instead, we apply the continuous greedy algorithm to a variant of (1) in which $g(x)$ is given by $F(p \cdot x)$ to obtain an initial solution x^0. From Lemma 7 we then have $F(p \cdot x^0) \ge (1-1/e)f^+(p \cdot x^*) \ge (1-1/e)\mathbb{E}[f(OPT)]$.

Given x^0, our algorithm is exactly the same as in the linear case. However, we must be more careful in our analysis. We define the quantity

$$z^t = F(\mathbf{1}_{S^t} + p \cdot x^t) - F(\mathbf{1}_{S^t})$$

where S^t and x^t are, respectively, the set of successfully probed elements and the current fractional solution at time t. Note that because after probing an element we set its variable to zero, for all elements $i \in S$ we have $x_i = 0$, and so indeed $\mathbf{1}_{S^t} + p \cdot x^t \in [0,1]^E$. Suppose that the algorithm runs for τ iterations, and note that $z^0 = F(p \cdot x^0) \ge (1-1/e)\mathbb{E}[f(OPT)]$ and $z^\tau = F(\mathbf{1}_{S^\tau + p\cdot 0^E}) - F(S^\tau) = 0$, so the conditions of Lemma 9 are satisfied with $\beta = (1-1/e)$.

We now analyze the expected decrease $z^t - z^{t+1}$ due to step $t+1$ of the algorithm. Suppose that the algorithm selects element i to probe. Then, we have $S^{t+1} = S^t + i$ with probability p_i and $S^{t+1} = S^t$ otherwise. Thus, we have

$$\begin{aligned}\mathbb{E}\left[z^t - z^{t+1}\right] &= \mathbb{E}\left[F(\mathbf{1}_{S^t} + x^t \cdot p) - F(\mathbf{1}_{S^t})\right] - \mathbb{E}\left[F(\mathbf{1}_{S^{t+1}} + x^{t+1} \cdot p) - F(\mathbf{1}_{S^{t+1}})\right] \\ &= \mathbb{E}\left[F(\mathbf{1}_{S^{t+1}}) - F(\mathbf{1}_{S^t})\right] + \mathbb{E}\left[F(\mathbf{1}_{S^t} + x^t \cdot p) - F(\mathbf{1}_{S^{t+1}} + x^{t+1} \cdot p)\right] \\ &\le \mathbb{E}\left[F(\mathbf{1}_{S^{t+1}}) - F(\mathbf{1}_{S^t})\right] + \mathbb{E}\left[F(\mathbf{1}_{S^t} + x^t \cdot p) - F(\mathbf{1}_{S^t} + x^{t+1} \cdot p)\right] \end{aligned} \tag{5}$$

where in the last line, we have used the fact that $S^{t+1} \geq S^t$ and F is increasing in all directions (Lemma 6). We shall first bound the second expectation in (5). We consider the vector δ of decreases in x, given by $\delta = (x^t - x^{t+1}) \cdot p$. For each $i \in E$, let $w_i = \frac{\partial F}{\partial x_i}(\mathbf{1}_{S^t}) = F(\mathbf{1}_{S^t+i}) - F(\mathbf{1}_{S^t})$. Let $y = \mathbf{1}_{S^t} + x^t \cdot p$, and suppose that we decrease the coordinates of y one at a time to obtain $y - \delta = \mathbf{1}_{S^t} + x^{t+1} \cdot p$, letting y^i be the value of y after the first $i-1$ coordinates have been decreased.[2] We then have:

$$F(y) - F(y-\delta) = \sum_i F(y^i) - F(y^{i+1}) = \sum_i F(y^i) - F(y^i - \delta_i \mathbf{1}_i)$$
$$= \sum_i \delta_i \frac{\partial F}{\partial x_i}(y^i - \delta_i \mathbf{1}_i) \leq \sum_i \delta_i \frac{\partial F}{\partial x_i}(\mathbf{1}_{S^t}) = \sum_i \delta_i w_i,$$

where the third equality follows from the fact that F is linear when one coordinate is changed (Lemma 5), while the inequality follows from the fact that the partial derivatives of F are coordinate-wise non-increasing (Lemma 6) and $y^i - \delta_i \mathbf{1}_i \geq \mathbf{1}_{S^t}$ for all i. Thus, we have:

$$\mathbb{E}\left[F(y) - F(y-\delta)\right] \leq \mathbb{E}\left[\sum_i \delta_i w_i\right] = \sum_i \mathbb{E}\left[\delta_i\right] \cdot w_i. \leq \frac{1}{\Sigma}(k^{out} + k^{in}) \sum_i p_i x_i^t w_i,$$

where the last inequality follows, as in the linear case, from inequality (4).

Returning to the first expectation in (5), we note that:

$$\mathbb{E}\left[F(\mathbf{1}_{S^{t+1}}) - F(\mathbf{1}_{S^t})\right] = \sum_i \mathbb{P}\left[i \text{ probed}\right] p_i (F(S^t + i) - F(S^t)) = \frac{1}{\Sigma} \sum_i p_i x_i^t w_i.$$

Thus, the total expected decrease $\mathbb{E}\left[z^t - z^{t+1}\right]$ from one step of our rounding procedure is at most:

$$\frac{1}{\Sigma} \sum_i p_i x_i^t w_i + \frac{1}{\Sigma} \sum_i (k^{out} + k^{in}) p_i x_i^t w_i = (k^{out} + k^{in} + 1) \frac{1}{\Sigma} \sum_i p_i x_i^t w_i.$$

On the other hand, the expected increase of $f(S^{t+1}) - f\left(S^t\right)$ in this step is:

$$\frac{1}{\Sigma} \sum_{e \in E} p_e x_e^t (f(S^t + e) - f(S^t)) = \frac{1}{\Sigma} \sum_{e \in E} p_e x_e^t (F(\mathbf{1}_{S^t + e}) - F(\mathbf{1}_{S^t})) = \frac{1}{\Sigma} \sum_{e \in E} p_e x_e^t w_e.$$

Thus, by Lemma 9, the final solution S^τ produced by the algorithm satisfies

$$\mathbb{E}\left[f(S^\tau)\right] \geq \left(1 - \frac{1}{e}\right) \left(\frac{1}{k^{out} + k^{in} + 1}\right) \mathbb{E}\left[OPT\right].$$

References

1 Marek Adamczyk. Improved analysis of the greedy algorithm for stochastic matching. *Inf. Process. Lett.*, 111:731–737, August 2011.

2 Alexander A. Ageev and Maxim Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *J. Comb. Optim.*, 8(3):307–328, 2004.

[2] With slight abuse of notation, we write x_i for the value of the ith decreased coordinate of x and $\mathbf{1}_i$ for the characteristic vector of this coordinate. That is, we identify an element with its index

3 Shipra Agrawal, Yichuan Ding, Amin Saberi, and Yinyu Ye. Correlation robust stochastic optimization. In *SODA*, pages 1087–1096, 2010.
4 Arash Asadpour, Hamid Nazerzadeh, and Amin Saberi. Stochastic submodular maximization. In *WINE*, pages 477–489, 2008.
5 Nikhil Bansal, Anupam Gupta, Jian Li, Julián Mestre, Viswanath Nagarajan, and Atri Rudra. When LP is the cure for your matching woes: Improved bounds for stochastic matchings. *Algorithmica*, 63(4):733–762, 2012.
6 E. M. L. Beale. Linear programming under uncertainty. *Journal of the Royal Statistical Society. Series B*, 17:173–184, 1955.
7 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint (extended abstract). In *IPCO*, pages 182–196, 2007.
8 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a Submodular Set Function Subject to a Matroid Constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.
9 Shuchi Chawla, Jason D. Hartline, David L. Malec, and Balasubramanian Sivan. Multi-parameter mechanism design and sequential posted pricing. In *STOC*, pages 311–320, 2010.
10 Ning Chen, Nicole Immorlica, Anna R. Karlin, Mohammad Mahdian, and Atri Rudra. Approximating matches made in heaven. In *ICALP*, pages 266–278, 2009.
11 G.B. Dantzig. Linear programming under uncertainty. *Management Science*, 1:197–206, 1955.
12 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Adaptivity and approximation for stochastic packing problems. In *SODA*, pages 395–404, 2005.
13 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Math. Oper. Res.*, 33(4):945–964, 2008.
14 Michel X. Goemans and Jan Vondrák. Stochastic covering and adaptivity. In *LATIN*, pages 532–543, 2006.
15 Sudipto Guha and Kamesh Munagala. Approximation algorithms for budgeted learning problems. In *STOC*, pages 104–113, 2007.
16 Sudipto Guha and Kamesh Munagala. Model-driven optimization using adaptive probes. In *SODA*, pages 308–317, 2007.
17 Anupam Gupta, Ravishankar Krishnaswamy, Marco Molinaro, and R. Ravi. Approximation algorithms for correlated knapsacks and non-martingale bandits. In *FOCS*, pages 827–836, 2011.
18 Anupam Gupta and Viswanath Nagarajan. A stochastic probing problem with applications. In *IPCO*, pages 205–216, 2013.
19 Robert Kleinberg and S. Matthew Weinberg. Matroid prophet inequalities. In *STOC*, pages 123–136, 2012.
20 A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.
21 Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.
22 Jan Vondrák. Personal correspondence.
23 David Williams. *Probability with Martingales*. Cambridge mathematical textbooks. Cambridge University Press, 1991.
24 Qiqi Yan. Mechanism design via correlation gap. In *SODA*, pages 710–719, 2011.

On Symmetric Circuits and Fixed-Point Logics

Matthew Anderson and Anuj Dawar

University of Cambridge Computer Laboratory, Cambridge, UK
`firstname.lastname@cl.cam.ac.uk`

Abstract

We study properties of relational structures such as graphs that are decided by families of Boolean circuits. Circuits that decide such properties are necessarily invariant to permutations of the elements of the input structures. We focus on families of circuits that are symmetric, i.e., circuits whose invariance is witnessed by automorphisms of the circuit induced by the permutation of the input structure. We show that the expressive power of such families is closely tied to definability in logic. In particular, we show that the queries defined on structures by uniform families of symmetric Boolean circuits with majority gates are exactly those definable in fixed-point logic with counting. This shows that inexpressibility results in the latter logic lead to lower bounds against polynomial-size families of symmetric circuits.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic

Keywords and phrases symmetric circuit, fixed-point logic, majority, counting, uniformity

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.41

1 Introduction

A property of graphs on n vertices can be seen as a Boolean function which takes as inputs the $\binom{n}{2}$ potential edges (each of which can be 0 or 1) and outputs either 0 or 1. For the function to determine a property of the graph, rather than of a particular presentation of the graph, it must be invariant under re-ordering the vertices of the graph. That is, permuting the $\binom{n}{2}$ inputs according to some permutation of $[n]$ leaves the value of the function unchanged. We call such Boolean functions *invariant*. Note that this does not require the function to be invariant under *all* permutations of its inputs, which would mean that it was entirely determined by the number of inputs that are set to 1.

It is a long-standing open problem in descriptive complexity to give a characterisation of the polynomial-time properties of finite relational structures (or, indeed, just graphs) as the classes of structures definable in some suitable logic (see, for instance, [7, Chapter 11]). It is known that fixed-point logic FP and its extension with counting FPC are strictly less expressive than deterministic polynomial time P [3]. It is easy to see that every polynomial-time property of graphs is decided by a P-uniform family of circuits that are *invariant* in the sense above. On the other hand, when a property of graphs is expressed in a formal logic, it gives rise to a family of circuits that are *explicitly invariant* or *symmetric*. By this we mean that their invariance is witnessed by the automorphisms of the circuits themselves. For instance, any sentence of FP translates into a polynomial-size family of symmetric Boolean circuits, while any sentence of FPC translates into a polynomial-size family of symmetric Boolean circuits with majority gates.

Concretely, a circuit C_n consists of a directed acyclic graph whose internal gates are marked by operations from a basis (e.g., the standard Boolean basis $\mathbb{B}_{\text{std}} := \{\text{AND, OR, NOT}\}$ or the majority basis $\mathbb{B}_{\text{maj}} = \mathbb{B}_{\text{std}} \cup \{\text{MAJ}\}$) and input gates which are marked with pairs of


© Matthew Anderson and Anuj Dawar;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 41–52

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


vertices representing potential edges of an n-vertex input graph. Such a circuit is *symmetric* if C_n has an automorphism π induced by each permutation σ of the n vertices, i.e., π moves the input gates of C_n according to σ and preserves operations and wiring of the internal gates of C_n. Clearly, any symmetric circuit is invariant.

Are symmetric circuits a weaker model of computation than invariant circuits? We aim at characterising the properties that can be decided by uniform families of symmetric circuits. Our main result shows that, indeed, any property that is decided by a uniform polynomial-size family of symmetric majority circuits can be expressed in FPC.

▶ **Theorem 1.** *A graph property is decided by a* P*-uniform family of symmetric majority circuits if, and only if, it is defined by a fixed-point with counting sentence.*

A consequence of this result is that inexpressibility results that have been proved for FPC can be translated into lower bound results for symmetric circuits. For instance, it follows (using [4]) that there is no polynomial-size family of symmetric majority circuits deciding 3-colourability or Hamiltonicity of graphs.

We also achieve a characterisation similar to Theorem 1 of symmetric Boolean circuits.

▶ **Theorem 2.** *A graph property is decided by a* P*-uniform family of symmetric Boolean circuits if, and only if, it is defined by a fixed-point sentence interpreted in $\mathcal{G} \oplus \langle [n], \leq \rangle$, i.e., the structure that is the disjoint union of an n-vertex graph $\mathcal{G}$ with a linear order of length n.*

Note that symmetric majority circuits can be transformed into symmetric Boolean circuits. But, since FP, even interpreted over $\mathcal{G} \oplus \langle [n], \leq \rangle$, is strictly less expressive than FPC, our results imply that any such translation must involve a super-polynomial blow-up in size. Similarly, our results imply with [3] that *invariant* Boolean circuits cannot be transformed into symmetric circuits (even with majority gates) without a super-polynomial blow-up in size. On the other hand, it is clear that symmetric majority circuits can still be translated into *invariant* Boolean circuits with only a polynomial blow-up.

Support. The main technical tool in establishing the translation from uniform families of symmetric circuits to sentences in fixed-point logics is a *support theorem* (stated informally below) that establishes properties of the stabiliser groups of gates in symmetric circuits.

We say that a set $X \subseteq [n]$ *supports* a gate g in a symmetric circuit C on an n-element input structure if every automorphism of C that is generated by a permutation of $[n]$ fixing X also fixes g. It is not difficult to see that for a family of symmetric circuits obtained from a given first-order formula ϕ there is a constant k such that all gates in all circuits of the family have a support of size at most k. To be precise, the gates in such a circuit correspond to subformulas ψ of ϕ along with an assignment of values from $[n]$ to the free variables of ψ. The set of elements of $[n]$ appearing in such an assignment forms a support of the gate and its size is bounded by the number of free variables ψ. Using the fact that any formula of FP is equivalent, on structures of size n, to a first-order formula with a constant bound k on the number of variables and similarly any formula of FPC is equivalent to a first-order formula *with majority quantifiers* (see [9]) and a constant bound on the number of variables, we see that the resulting circuits have supports of constant-bounded size. Our main technical result is that the existence of supports of bounded size holds, in fact, for all polynomial-size families of symmetric circuits. In its general form, we show the following theorem in Section 3 via an involved combinatorial argument.

▶ **Theorem 3** (Informal Support Thm)**.** *Let C be a symmetric circuit with s gates over a graph of size n. If n is sufficiently large and s is sub-exponential in n, then every gate in C has a support of size $O\Big(\frac{\log s}{\log n}\Big)$.*

In the typical instantiation of the Theorem 3 the circuit C contains a polynomial number of gates $s = \text{poly}(n)$ and hence the theorem implies that every gate has a support that is bounded in size by a constant. The proof of the Theorem 3 mainly relies on the structural properties of symmetric circuits and is largely independent of the semantics of such circuits; this means it may be of independent interest for other circuit bases and in other settings.

Symmetric Circuits and FP. In Section 4 we show that each polynomial-size family $\mathcal{C}$ of symmetric circuits can be translated into a formula of fixed-point logic. If the family $\mathcal{C}$ is P-uniform, by the Immerman-Vardi Theorem [12, 8] there is an FP-definable interpretation of the circuit C_n in the ordered structure $\langle [n], \leq \rangle$. We show that the support of a gate is computable in polynomial time, and hence we can also interpret the support of each gate in $\langle [n], \leq \rangle$. The circuit C_n can be evaluated on an input graph $\mathcal{G}$ by fixing a bijection between $[n]$ and the universe U of $\mathcal{G}$. We associate with each gate of g of C_n the set of those bijections that cause g to evaluate to 1 on $\mathcal{G}$. This set of bijections admits a compact (i.e., polynomial-size) representation as the set of injective maps from the support of g to U. We show that these compact representations can be inductively defined by formulas of FP, or FPC if the circuit also admits majority gates.

Thus, we obtain that P-uniform families of symmetric Boolean circuits can be translated into formulas of FP interpreted in $\mathcal{G}$ combined with a disjoint linear order $\langle [|\mathcal{G}|], \leq \rangle$, while families containing majority gates can be simulated by sentences of FPC. The reverse containment follows using classical techniques. As a consequence we obtain the equivalences of Theorems 1 & 2, and a number of more general results as this sequence of arguments naturally extends to: (i) inputs given as an arbitrary relational structure, (ii) outputs defining arbitrary relational queries, and (iii) non-uniform circuits, provided the logic is allowed additional advice on the disjoint linear order.

Related Work. The term "symmetric circuit" is used by Denenberg et al. in [6] to mean what we call invariant circuits. They give a characterisation of first-order definability in terms of a restricted invariance condition, namely circuits that are invariant and whose relativisation to subsets of the universe remains invariant. Our definition of symmetric circuits follows that in [10] where Otto describes it as the "natural and straightforward combinatorial condition to guarantee generic or isomorphism-invariant performance." He combines it with a size restriction on the orbits of gates along with a strong uniformity condition, which he calls "coherence", to give an exact characterisation of definability in infinitary logic. A key element is the proof that if the orbits of gates in such a circuit are polynomially bounded in size then they have supports of bounded size. We remove the assumption of coherence from this argument and show that constant-size supports exist in any polynomial-size symmetric circuit. This requires a generalisation of what Otto calls a "base" to supporting partitions. See Section 5 for more discussion of connections with prior work.

Due to space limitations, full proofs are omitted and may be found in [1].

2 Preliminaries

Let $[n]$ denote the set of positive integers $\{1, \ldots, n\}$. Let Sym_S denote the group of all permutations of the set S. When $S = [n]$, we write Sym_n for $\text{Sym}_{[n]}$.

2.1 Vocabularies, Structures, and Logics

A *relational vocabulary* (always denoted by τ) is a finite sequence of relation symbols $(R_1^{r_1}, \ldots, R_k^{r_k})$ where for each $i \in [k]$ the relation symbol R_i has an associated arity $r_i \in \mathbb{N}$.

A τ-*structure* $\mathcal{A}$ is a tuple $\langle A, R_1^{\mathcal{A}}, \ldots, R_k^{\mathcal{A}} \rangle$ consisting of (i) a non-empty set A called the *universe* of $\mathcal{A}$, and (ii) relations $R_i^{\mathcal{A}} \subseteq A^{r_i}$ for $i \in [k]$. Members of the universe A are called *elements* of $\mathcal{A}$. A *multi-sorted* structure is one whose universe is given as a disjoint union of several distinct *sorts*. Define the *size* of a structure $|\mathcal{A}|$ to be the cardinality of its universe. All structures considered in this paper are *finite*, i.e., their universes have finite cardinality. Let $\mathrm{fin}[\tau]$ denote the set of all finite τ-structures.

First-Order and Fixed-Point Logics. Let $\mathrm{FO}(\tau)$ denote *first-order logic* with respect to the vocabulary τ. The logic $\mathrm{FO}(\tau)$ is the set of formulas whose atoms are formed using the relation symbols in τ, an equality symbol $=$, an infinite sequence of variables $(x, y, z \ldots)$, and that are closed under the Boolean connectives ($\wedge$ and $\vee$), negation ($\neg$), and universal and existential quantification ($\forall$ and $\exists$). Let *fixed-point logic* $\mathrm{FP}(\tau)$ denote the extension of $\mathrm{FO}(\tau)$ to include an inflationary fixed-point operator ifp. Assume standard syntax and semantics for FO and FP (see the textbook [7] for more background). For a formula ϕ write $\phi(x)$ to indicate that x is the tuple of the free variables of ϕ. For a logic $\mathcal{L}$, a formula $\phi(x) \in \mathcal{L}(\tau)$ with k free variables, $\mathcal{A} \in \mathrm{fin}[\tau]$, and tuple $a \in A^k$ write $\mathcal{A} \models_{\mathcal{L}} \phi[a]$ to express that the tuple a makes the formula ϕ true in the structure $\mathcal{A}$ with respect to the logic $\mathcal{L}$. We usually drop the subscript $\mathcal{L}$ and write $\mathcal{A} \models \phi[a]$ when no confusion would arise.

Logics with Disjoint Advice. Let τ_{arb} be a relational vocabulary without a binary relation symbol $\leq$. Let $\Upsilon : \mathbb{N} \to \mathrm{fin}[\tau_{\mathrm{arb}} \uplus \{\leq^2\}]$ be an *advice function*, where for $n \in \mathbb{N}$, $\Upsilon(n)$ has universe $[n]$ naturally ordered by $\leq$. Let $(\mathrm{FP} + \Upsilon)(\tau)$ denote the set of formulas of $\mathrm{FP}(\tau')$ where $\tau' := \tau \uplus \tau_{\mathrm{arb}} \uplus \{\leq^2\}$ and τ is a vocabulary disjoint from $\tau_{\mathrm{arb}} \uplus \{\leq^2\}$. For a structure $\mathcal{A} \in \mathrm{fin}[\tau]$ define the semantics of $\phi \in (\mathrm{FP} + \Upsilon)(\tau)$ to be $\mathcal{A} \models_{(\mathrm{FP}+\Upsilon)} \phi$ iff $\mathcal{A}^{\Upsilon} \models_{\mathrm{FP}} \phi$, where $\mathcal{A}^{\Upsilon} := \mathcal{A} \oplus \Upsilon(|\mathcal{A}|)$ is the multi-sorted τ'-structure formed by taking the disjoint union of $\mathcal{A}$ with a structure coding a linear order of corresponding cardinality endowed with interpretations of the relations in τ_{arb}. The universe of the multi-sorted structure $\mathcal{A}^{\Upsilon}$ is written as $A \uplus [|A|]$; refer to A as the *point sort* of $\mathcal{A}^{\Upsilon}$ and to $[|A|]$ as the *number sort* of $\mathcal{A}^{\Upsilon}$.

We are primarily interested in the special case when τ_{arb} is empty and hence $\Upsilon(|\mathcal{A}|) = \langle [|A|], \leq \rangle$ is simply a linear order. Denote formulas of this logic by $(\mathrm{FP} + \leq)(\tau)$ and extended structures by $\mathcal{A}^{\leq}$ to emphasise the disjoint linear order. Let $\mathrm{FPC}(\tau)$ denote the extension of $(\mathrm{FP} + \leq)(\tau)$ with a counting operator $\#_x$ where x is a point or number variable. For a structure $\mathcal{A} \in \mathrm{fin}[\tau]$ and a formula $\phi(x) \in \mathrm{FPC}(\tau)$, $\#_x \phi(x)$ is a term denoting the element in the number sort corresponding to $|\{a \in \mathcal{A} \mid \mathcal{A} \models \phi[a]\}|$. See [7, Section 8.4.2] for more details. Finally, we consider the extension of fixed-point logic with both advice functions and counting quantifiers $(\mathrm{FPC} + \Upsilon)(\tau)$.

2.2 Symmetric and Uniform Circuits

A *Boolean basis* (always denoted by $\mathbb{B}$) is a finite set of Boolean functions from $\{0,1\}^*$ to $\{0,1\}$. We consider only bases containing symmetric functions, i.e., for all $f \in \mathbb{B}$, $f(x) = f(y)$ for all $n \in \mathbb{N}$ and $x, y \in \{0,1\}^n$ with the same number of ones. The *standard Boolean basis* $\mathbb{B}_{\mathrm{std}}$ consists of unbounded fan-in AND, OR, and unary NOT operators. The *majority basis* $\mathbb{B}_{\mathrm{maj}}$ extends the standard basis with an operator MAJ which is one iff the number of ones in the input is at least the number of zeroes.

▶ **Definition 4** (Circuits on Structures). A *Boolean* $(\mathbb{B}, \tau)$*-circuit* C *with universe* U *computing a* q*-ary query* Q is a structure $\langle G, W, \Omega, \Sigma, \Lambda \rangle$.

- G is a set called the *gates* of C. The *size* of C is $|C| := |G|$.

- $W \subseteq G \times G$ is a binary relation called the *wires* of the circuit. We require that (G, W) forms a *directed acyclic graph*. Call the gates with no incoming wires *input gates*, and all other gates *internal gates*. Gates h with $(h, g) \in W$ are called the *children* of g.
- Ω is an injective function from U^q to G. The gates in the image of Ω are called the *output gates*. When $q = 0$, Ω is a constant function mapping to a single output gate.
- Σ is a function from G to $\mathbb{B} \uplus \tau \uplus \{0, 1\}$ which maps input gates into $\tau \uplus \{0, 1\}$ with $|\Sigma^{-1}(0)|, |\Sigma^{-1}(1)| \leq 1$ and internal gates into $\mathbb{B}$. Call the input gates marked with a relation from τ *relational gates* and the input gates marked with 0 or 1 *constant gates*.
- Λ is a sequence of injective functions $(\Lambda_R)_{R \in \tau}$ where for each $R \in \tau$, Λ_R maps each relational gate g with $R = \Sigma(g)$ to $\Lambda_R(g) \in U^r$ where r is the arity of R. Where no ambiguity arises, we write $\Lambda(g)$ for $\Lambda_R(g)$.

Let C be a Boolean $(\mathbb{B}, \tau)$-circuit with universe U, $\mathcal{A} \in \text{fin}[\tau]$ with $|\mathcal{A}| = |U|$, and $\gamma : A \to U$ be a bijection. Let $\gamma\mathcal{A}$ denote the τ-structure over the universe U obtained by relabelling the universe of $\mathcal{A}$ according to γ. Recursively evaluate C on $\gamma\mathcal{A}$ by determining a value $C[\gamma\mathcal{A}](g)$ for each gate g: (i) a constant gate evaluates to the bit given by $\Sigma(g)$, (ii) a relational gate evaluates to 1 iff $\gamma\mathcal{A} \models \Sigma(g)(\Lambda_{\Sigma(g)}(g))$, and (iii) an internal gate evaluates to the result of applying the Boolean operation $\Sigma(g)$ to the values for g's children. C defines the q-ary query $Q \subseteq \mathcal{A}^q$ where $a \in Q$ iff $C[\gamma\mathcal{A}](\Omega(\gamma a)) = 1$.

▶ **Definition 5** (Invariant Circuit)**.** Let C be a $(\mathbb{B}, \tau)$-circuit with universe U computing a q-ary query. The circuit C is *invariant* if for every $\mathcal{A} \in \text{fin}[\tau]$ with $|\mathcal{A}| = |U|$, $a \in \mathcal{A}^q$, and bijections γ_1, γ_2 from A to U, $C[\gamma_1\mathcal{A}](\Omega(\gamma_1 a)) = C[\gamma_2\mathcal{A}](\Omega(\gamma_2 a))$.

Invariance indicates that C computes a property of τ-structures which is invariant to presentations of the structure. Moreover, for an invariant circuit C only the size of U matters and we often write $C = C_n$, for emphasis, when the universe is size n. A *family* $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ of invariant $(\mathbb{B}, \tau)$-circuits naturally computes a q-ary query on τ-structures. When $q = 0$ the family computes a Boolean property of structures. We now discuss an algebraic property of circuits called *symmetry* that implies invariance.

Symmetric Circuits. Permuting a circuit's universe may induce automorphisms of the circuit.

▶ **Definition 6** (Induced Automorphism)**.** Let $C = \langle G, W, \Omega, \Sigma, \Lambda \rangle$ be a $(\mathbb{B}, \tau)$-circuit with universe U computing a q-ary query. Let $\sigma \in \text{Sym}_U$. If there is a bijection π from G to G such that

- for all gates $g, h \in G$, $W(g, h)$ iff $W(\pi(g), \pi(h))$,
- for all output tuples $x \in U^q$, $\pi\Omega(x) = \Omega(\sigma(x))$,
- for all gates $g \in G$, $\Sigma(g) = \Sigma(\pi(g))$, and
- for each relational gate $g \in G$, $\sigma\Lambda(g) = \Lambda(\pi(g))$,

we say σ *induces the automorphism* π of C.

The principle goal of this paper is to understand the computational power of circuit classes with the following type of algebraic symmetry.

▶ **Definition 7** (Symmetric)**.** A circuit C with universe U is called *symmetric* if for every permutation $\sigma \in \text{Sym}_U$, σ induces an automorphism of C.

It is not difficult to see that, for a symmetric circuit C, there is a homomorphism $h : \text{Sym}_U \to \text{Aut}(C)$ (where $\text{Aut}(C)$ denotes the automorphism group of C) such that $h(\sigma)$ is an automorphism induced by σ.

To avoid certain trivialities we restrict ourselves to circuits which are *rigid*.

▶ **Definition 8** (Rigid). Let $C = \langle G, W, \Omega, \Sigma, \Lambda\rangle$ be a $(\mathbb{B}, \tau)$-circuit with universe U. Call C *rigid* if there do not exist distinct gates $g, g' \in G$ with $\Sigma(g) = \Sigma(g')$, $\Lambda(g) = \Lambda(g')$, $\Omega^{-1}(g) = \Omega^{-1}(g')$, and for every $g'' \in G$, $W(g'', g)$ iff $W(g'', g')$.

For a rigid symmetric circuit C it is easy to show that the group of automorphisms of C is exactly Sym_U acting faithfully. We shall therefore abuse notation and use these interchangeably. In particular, we shall write σg to denote the image of a gate g in C under the action of the automorphism induced by a permutation σ in Sym_U.

An examination of the definitions suffices to show that symmetry implies invariance. In symmetric circuits it is useful to consider those permutations which induce automorphisms that fix gates. Let $\mathcal{P}$ be a partition of a set U. Let the *pointwise stabiliser* of $\mathcal{P}$ be $\mathrm{Stab}_U(\mathcal{P}) := \{\sigma \in \mathrm{Sym}_U \mid \forall P \in \mathcal{P}, \sigma P = P\}$, and similarly define the *setwise stabiliser* $\mathrm{Stab}_U\{\mathcal{P}\} := \{\sigma \in \mathrm{Sym}_U \mid \forall P \in \mathcal{P}, \sigma P \in \mathcal{P}\}$. For a gate g in a rigid symmetric circuit C with universe U, let the *stabiliser* of g be $\mathrm{Stab}_U(g) := \{\sigma \in \mathrm{Sym}_U \mid \sigma g = g\}$, and let the *orbit* of g under the automorphism group $\mathrm{Aut}(C)$ of C be $\mathrm{Orb}(g) := \{\sigma g \mid \sigma \in \mathrm{Sym}_U\}$. In each case, when $U = [n]$, we write Stab_n instead of $\mathrm{Stab}_{[n]}$.

Uniform Circuits. One natural class of circuits are those with polynomial-size descriptions that can be generated by a deterministic polynomial-time machine.

▶ **Definition 9** (P and P/poly-Uniform). A $(\mathbb{B}, \tau)$-circuit family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ computing a q-ary query is P/poly-*uniform* if there exists an integer $t \geq q$ and function $\Upsilon : \mathbb{N} \to \{0,1\}^*$ which takes an integer n to a binary string $\Upsilon(n)$ such that $|\Upsilon(n)| = \mathrm{poly}(n)$, and $\Upsilon(n)$ describes[1] the circuit C_n whose gates are indexed by t-tuples of $[n]$, inputs are labelled by t-tuples of $[n]$, and outputs are labelled by q-tuples of $[n]$. Moreover, if there exists a deterministic Turing machine M that for each integer n computes $\Upsilon(n)$ from 1^n in time $\mathrm{poly}(n)$ call $\mathcal{C}$ P-*uniform*.

Note that such uniform families implicitly have polynomial size.

Over ordered structures neither P-uniform nor P/poly-uniform circuits need compute invariant queries as their computation may implicitly depend on the order associated with $[n]$. To obtain invariance for such circuits we assert symmetry. The next section proves a natural property of symmetric circuits that ultimately implies that *symmetric* P-uniform circuits coincide with FP definitions on the standard and majority bases.

3 Symmetry and Support

In this section we analyse the algebraic properties of symmetric circuits.

▶ **Definition 10** (Support). Let C be a rigid symmetric circuit with universe U and let g be a gate in C. A set $X \subseteq U$ *supports* g if $\mathrm{Stab}_U(X) \subseteq \mathrm{Stab}_U(g)$.

We now show how to associate supports of constant size in a canonical way to all gates in *any* rigid symmetric circuit of polynomial size. Indeed, our result is more general as it associates moderately growing supports to gates in circuits of sub-exponential size. We first introducing the more general notion of a *supporting partition* for a permutation group, which can be canonically associated with any permutation group G, and obtain bounds on the size of such a partition based on the index of G in the symmetric group. These results are then

[1] Formally one must define a particular way of encoding circuits via binary strings. However, since the details of the representation are largely irrelevant for our purposes we omit them.

used to bound the size of supports of stabiliser groups of gates in rigid symmetric circuits as a function of circuit size. This proves our main technical result—the Support Theorem.

A supporting partition generalises the notion of a support of a gate by replacing the set with a partition and the stabiliser group of the gate with an arbitrary permutation group.

▶ **Definition 11** (Supporting Partition). Let $G \subseteq \mathrm{Sym}_U$ be a group and $\mathcal{P}$ a partition of U. We say that $\mathcal{P}$ is a *supporting partition* of G if $\mathrm{Stab}_U(\mathcal{P}) \subseteq G$.

For intuition consider two extremes. When G has supporting partition $\mathcal{P} = \{U\}$, it indicates $G = \mathrm{Sym}_U$. Saying that G has supporting partition $\mathcal{P} = \{\{u_1\}, \{u_2\}, \ldots, \{u_{|U|}\}\}$ indicates only that G contains the identity permutation, which is always true.

A natural partial order on partitions is the coarseness relation, i.e., $\mathcal{P}'$ is as coarse as $\mathcal{P}$, denoted $\mathcal{P}' \supseteq \mathcal{P}$, if every part in $\mathcal{P}$ is contained in some part of $\mathcal{P}'$. A proof is similar to that of (*) on page 379 of [10] implies the following lemma.

▶ **Lemma 12.** *Each permutation group $G \subseteq \mathrm{Sym}_U$ has a unique coarsest supporting partition.*

We write $\mathsf{SP}(G)$ for *the unique coarsest partition supporting G*. By analysing how supporting partitions are affected by the conjugacy action of Sym_U it is easy to show that any group G is sandwiched between the pointwise and setwise stabilisers of $\mathsf{SP}(G)$.

▶ **Lemma 13.** *For any group $G \subseteq \mathrm{Sym}_U$, we have $\mathrm{Stab}_U(\mathsf{SP}(G)) \subseteq G \subseteq \mathrm{Stab}_U\{\mathsf{SP}(G)\}$.*

Note that these bounds need not be tight. For example, if G is the alternating group on U (or, indeed, any transitive, primitive subgroup of Sym_U), then $\mathsf{SP}(G)$ is the partition of U into singletons. In this case, $\mathrm{Stab}_U(\mathsf{SP}(G))$ is the trivial group while $\mathrm{Stab}_U\{\mathsf{SP}(G)\} = \mathrm{Sym}_U$.

We now use the bounds given by Lemma 13, in conjunction with bounds on G to obtain size bounds on $\mathsf{SP}(G)$. Recall that the index of G in Sym_U, denoted $[\mathrm{Sym}_U : G]$ is the number of cosets of G in Sym_U or, equivalently, $\frac{|\mathrm{Sym}_U|}{|G|}$. The next lemma, proved via a involved combinatorial argument, says that if $\mathcal{P}$ is a partition of $[n]$ where the index of $\mathrm{Stab}_n\{\mathcal{P}\}$ in Sym_n is sufficiently small then (i) the number of parts in $\mathcal{P}$ is either very small or very big, and (ii) if the number of parts in $\mathcal{P}$ is small, then it must have a large part.

▶ **Lemma 14.** *Let ϵ and n be such that $0 \leq \epsilon < 1$ and $\log n \geq \frac{8}{\epsilon^2}$. Let $\mathcal{P}$ be a partition of $[n]$, $s := [\mathrm{Sym}_n : \mathrm{Stab}_n\{\mathcal{P}\}]$ and $n \leq s \leq 2^{n^{1-\epsilon}}$.*

1. *Let $k := |\mathcal{P}|$, then $\min\{k, n-k\} \leq \frac{8}{\epsilon}\frac{\log s}{\log n}$.*
2. *If $|\mathcal{P}| \leq \frac{n}{2}$, then $\mathcal{P}$ contains a part with at least $n - \frac{33}{\epsilon} \cdot \frac{\log s}{\log n}$ elements.*

We leverage the above combinatorial lemmas to show that in symmetric circuits of polynomial size, each gate has a small supporting partition, and hence has a small support. Let g be a gate in a rigid symmetric circuit C over universe U, we abuse notation and write $\mathsf{SP}(g)$ for $\mathsf{SP}(\mathrm{Stab}_U(g))$. Note that, if P is any part in $\mathsf{SP}(g)$, then $U \setminus P$ is a support of g in the sense of Definition 10. We write $\|\mathsf{SP}(g)\|$ to denote the smallest value of $|U \setminus P|$ over all parts P in $\mathsf{SP}(g)$. Also, let $\mathsf{SP}(C)$ denote the maximum of $\|\mathsf{SP}(g)\|$ over all gates g in C.

By the orbit-stabiliser theorem, $|\mathrm{Orb}(g)| = [\mathrm{Sym}_U : \mathrm{Stab}_U(g)]$. By Lemma 13, we have that $\mathrm{Stab}_U(g) \subseteq \mathrm{Stab}_U\{\mathsf{SP}(g)\}$ and thus, if s is an upper bound on $|\mathrm{Orb}(g)|$, $s \geq [\mathrm{Sym}_U : \mathrm{Stab}_U(g)] \geq [\mathrm{Sym}_U : \mathrm{Stab}_U\{\mathsf{SP}(g)\}]$. Then, by Part 2 of Lemma 14, g has a support of small size provided that (i) s is sub-exponential, and (ii) $\mathsf{SP}(g)$ has fewer than $n/2$ parts. Thus, to prove our main technical theorem, which formalises Theorem 3 from the introduction, it suffices to show that if s is sufficiently sub-exponential, (ii) holds.

▶ **Theorem 15** (Support Theorem). *For any ϵ and n with $\frac{2}{3} \leq \epsilon \leq 1$ and $n > 2^{\frac{56}{\epsilon^2}}$, if C is a rigid symmetric circuit over universe U with $|U| = n$ and $s := \max_{g \in C} |\mathrm{Orb}(g)| \leq 2^{n^{1-\epsilon}}$, then, $\mathsf{SP}(C) \leq \frac{33}{\epsilon} \frac{\log s}{\log n}$.*

Proof. Suppose $1 \leq s < n$. C cannot have relational inputs, because each relational gate must have an orbit of size at least n, so each gate of C computes a constant Boolean function. The support of every gate g in C must be $\{U\}$, and hence $0 = \|\mathsf{SP}(g)\| = \mathsf{SP}(C)$. Therefore assume $s \geq n$.

To conclude the theorem from Part 2 of Lemma 14 it suffices to argue that for all gates g, $|\mathsf{SP}(g)| \leq \frac{n}{2}$. Suppose g is a constant gate, then, because g is the only gate with its label, it is fixed under all permutations and hence $|\mathsf{SP}(g)| = |\{U\}| = 1 < \frac{n}{2}$. If g is a relational gate, then it is fixed by any permutation that fixes all elements appearing in $\Lambda(g)$ and moved by all others. Thus, $\mathsf{SP}(g)$ must contain singleton parts for each element of U in $\Lambda(g)$ and a part containing everything else. Thus, if $|\mathsf{SP}(g)| > \frac{n}{2}$, $\mathsf{SP}(g)$ contains at least $\frac{n}{2}$ singleton parts, there is a contradiction using the bounds on s, n, and ϵ, $s \geq |\mathrm{Orb}(g)| \geq \|\mathsf{SP}(g)\|! \cdot \binom{n}{\|\mathsf{SP}(g)\|} \geq \lfloor \frac{n}{2} \rfloor ! \geq 2^{\lfloor \frac{n}{4} \rfloor} > 2^{n^{1-\epsilon}}$.

It remains to consider internal gates. For the sake of contradiction let g be a topologically first internal gate such that $\mathsf{SP}(g)$ has more than $\frac{n}{2}$ parts. Part 1 of Lemma 14 implies, along with the assumptions on s, n, and ϵ, that $n - |\mathsf{SP}(g)| \leq k' := \left\lceil \frac{8}{\epsilon} \frac{\log s}{\log n} \right\rceil \leq \frac{1}{4} n^{1-\epsilon} < \frac{n}{2}$.

Let H denote the children of g. Because g is a topologically first gate with $|\mathsf{SP}(g)| > \frac{n}{2}$, for all $h \in H$, $\mathsf{SP}(h)$ has at most $\frac{n}{2}$ parts. As before, we argue a contradiction with the upper bound on s. This is done by demonstrating that there is a set of gate-automorphism pairs $S = \{(h, \sigma) \mid h \in H, \sigma \in \mathrm{Sym}_U\}$ that are: (i) *useful* – the automorphism moves the gate out of the set of g's children, i.e., $\sigma h \notin H$, and (ii) *independent* – each child and its image under the automorphism are fixed points of the other automorphisms in the set, i.e., for all $(h, \sigma), (h', \sigma') \in S$, $\sigma' h = h$ and $\sigma' \sigma h = \sigma h$. Note that sets which are useful and independent contain tuples whose gate and automorphism parts are all distinct. The set S describes elements in the orbit of H with respect to Sym_U.

▶ **Claim 16.** *Let S be useful and independent, then $|\mathrm{Orb}(H)| \geq 2^{|S|}$.*

Proof. Let R be any subset of S. Derive an automorphism from R: $\sigma_R := \prod_{(h,\sigma) \in R} \sigma$ (since automorphisms need not commute, fix an arbitrary ordering of S).

Let R and Q be distinct subsets of S where without loss of generality $|R| \geq |Q|$. Pick any $(h, \sigma) \in R \backslash Q \neq \emptyset$. Because S is independent $\sigma_R h = \sigma h$ and $\sigma_Q \sigma h = \sigma h$. Since S is useful, $\sigma h \notin H$. Thus $\sigma h \in \sigma_R H$, but $\sigma h \notin \sigma_Q H$. Hence $\sigma_R H \neq \sigma_Q H$. Therefore each subset of S can be identified with a distinct element in $\mathrm{Orb}(H)$ and hence $|\mathrm{Orb}(H)| \geq 2^{|S|}$. ◀

Thus to reach a contradiction it suffices to construct a sufficiently large set S of gate-automorphism pairs. To this end, divide U into $\lfloor \frac{|U|}{k'+2} \rfloor$ disjoint sets S_i of size $k' + 2$ and ignore the elements left over. Observe that for each i there is a permutation σ_i which fixes $U \backslash S_i$ but σ_i moves g, because otherwise the supporting partition of g could be smaller $(n - (k' + 2) + 1)$. Since g is moved by each σ_i and C is rigid, there must be an associated child $h_i \in H$ with $\sigma_i h_i \notin H$. Thus let (h_i, σ_i) be the gate-automorphism pair for S_i, these pairs are *useful*. Let Q_i be the union of all but the largest part of $\mathsf{SP}(h_i)$. Observe that for any σ which fixes Q_i pointwise σ also fixes both h_i and $\sigma_i h_i$, by the definition of support.

Define a directed graph K on the sets S_i as follows. Include an edge from S_i to S_j, with $i \neq j$, if $Q_i \cap S_j \neq \emptyset$. An edge in K indicates a potential lack of independence between (h_i, σ_i) and (h_j, σ_j), and on the other hand if there are no edges between S_i and S_j, the associated pairs are independent. Thus it remains to argue that K has a large independent set. This is

possible because the out-degree of S_i in K is bounded by $|Q_i| = \|\mathsf{SP}(h_i)\| \leq \frac{33}{\epsilon}\frac{\log s}{\log n}$ as the sets S_i are disjoint and Part 2 of Lemma 14 can be applied to h_i. Thus the average total degree (in-degree + out-degree) of K is at most $9k'$. Greedily select a maximal independent set in K by repeatedly selecting the S_i with the lowest total degree and eliminating it and its neighbours. This action does not effect the bound on the average total degree of K and hence determines an independent set I in K of size at least

$$\frac{\lfloor \frac{|U|}{k'+2} \rfloor}{9k'+1} \geq \frac{n-(k'+2)}{(9k'+1)(k'+2)} \geq \frac{\frac{n}{2}-1}{9k'^2+10k'+2} \geq \frac{\frac{7}{16}n}{9k'^2+10k'+2} \geq \frac{n}{(7k')^2}$$

where the first inequality follows by expanding the floored expression, the second follows because $k' < \frac{n}{2}$, the third follows from the lower bound on n, and the last follows because $k' \geq 1$ as it is the ceiling of a positive non-zero quantity by definition.

Take $S := \{(h_i, \sigma_i) \mid S_i \in I\}$. By the argument above S is useful and independent. By Claim 16, conclude that $s \geq |\mathrm{Orb}(g)| \geq |\mathrm{Orb}(H)| \geq 2^{|S|} \geq 2^{\frac{n}{(7k')^2}}$. For $\epsilon \geq \frac{2}{3}$, $s \leq 2^{n^{1-\epsilon}}$, and $\frac{\epsilon}{56}\log n > 1$ the following is a contradiction $\log s \geq n \cdot (\frac{56}{\epsilon}\frac{\log s}{\log n})^{-2} > n \cdot (n^{1-\epsilon})^{-2} = n^{2\epsilon-1} \geq n^{1-\epsilon}$. Thus $|\mathsf{SP}(g)| \leq \frac{n}{2}$ for all $g \in C$ and the proof is complete by Part 2 of Lemma 14. ◀

Observe that when s is polynomial in n the support of a rigid symmetric circuit family is asymptotically constant. This is the case for polynomial-size families.

▶ **Corollary 17.** *Let $\mathcal{C}$ be a polynomial-size rigid symmetric circuit family, then $\mathsf{SP}(\mathcal{C}) = O(1)$.*

4 Translating Symmetric Circuits to Formulas

In this section, we deploy the Support Theorem to show that P-uniform families of symmetric circuits can be translated into formulas of fixed-point logic. We can show that there is a polynomial-time algorithm that takes a symmetric circuit and outputs an equivalent rigid symmetric circuit together with the supporting partitions of each gate.

▶ **Lemma 18.** *Let C be a symmetric $(\mathbb{B}, \tau)$-circuit with universe U. There is a deterministic algorithm which runs in time $\mathrm{poly}(|C|)$ and outputs a rigid symmetric $(\mathbb{B}, \tau)$-circuit C' computing the same query as C along with coarsest supporting partitions for every gate of C'.*

Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a family of P-uniform symmetric $(\mathbb{B}, \tau)$-circuits computing a q-ary query. Let $\mathcal{A} \in \mathrm{fin}[\tau]$ be an input structure with universe U of size n. By Lemma 18 and the Immerman-Vardi theorem, we have an FP interpretation defining a rigid symmetric circuit equivalent to C_n over the number sort of $\mathcal{A}^{\leq}$, i.e., a tuple of formulas of $\mathrm{FP}(\leq)$ that define the circuit when interpreted in $\langle [n], \leq \rangle$. Moreover, the interpretation provides the coarsest supporting partitions of the gates in C_n. Note that C_n is defined over the universe $[n]$.

By Theorem 15, there is a constant bound k so that for each gate g in C_n the union of all but the largest part of the coarsest partition supporting g, $\mathsf{SP}(g)$, has at most k elements. Moreover, this union is a *support* of g in the sense of Definition 10. We call it the *canonical support* of g and denote it by $\mathrm{sp}(g)$. To describe the evaluation of the circuit C_n with a formula of fixed-point logic, we show that the evaluation of a gate g in C_n with respect to the structure $\mathcal{A}$ depends only on how its universe U is mapped to the canonical support of g.

For any set $X \subseteq [n]$, let U^X denote the set of *injective* functions from X to U. For $X, Y \subseteq [n]$ and $\alpha \in U^X, \beta \in U^Y$, we say α and β are *consistent*, denoted $\alpha \sim \beta$, if for all $z \in X \cap Y, \alpha(z) = \beta(z)$, and for all $x \in X \backslash Y$ and $y \in Y \backslash X$, $\alpha(x) \neq \beta(y)$. Recall that any bijection $\gamma : U \to [n]$ determines an evaluation of the circuit C_n on the input structure $\mathcal{A}$ which assigns to each gate g the Boolean value $C_n[\gamma\mathcal{A}](g)$. Let g be a gate and let

$\Gamma(g) := \{\gamma \mid C_n[\gamma\mathcal{A}](g) = 1\}$. The following claim establishes that the membership of γ in $\Gamma(g)$ (moreover, the number of 1s input to g) depends only on what γ maps to $\mathrm{sp}(g)$.

▶ **Claim 19.** *Let g be a gate in C_n with children H. Let $\alpha \in U^{\mathrm{sp}(g)}$, then for all $\gamma_1, \gamma_2 : U \to [n]$ with $\gamma_1^{-1} \sim \alpha$ and $\gamma_2^{-1} \sim \alpha$,*

1. *$\gamma_1 \in \Gamma(g)$ iff $\gamma_2 \in \Gamma(g)$.*
2. *$|\{h \in H \mid \gamma_1 \in \Gamma(h)\}| = \sum_{h\in H} \frac{|A_h \cap \mathrm{EV}_h|}{|A_h|}$, where for $h \in H$, $A_h := \{\beta \in U^{\mathrm{sp}(h)} \mid \alpha \sim \beta\}$.*

We associate with each gate g a set of injective functions $\mathrm{EV}_g \subseteq U^{\mathrm{sp}(g)}$ defined by $\mathrm{EV}_g := \{\alpha \in U^{\mathrm{sp}(g)} \mid \exists \gamma \in \Gamma(g) \wedge \alpha \sim \gamma^{-1}\}$ and note that, by Claim 19, this completely determines $\Gamma(g)$. Since $[n]$ is linearly ordered, $X \subseteq [n]$ inherits this order and we write $\vec{X}$ for the ordered $|X|$-tuple consisting of the elements of X in the inherited order. For $\alpha \in U^X$ write $\vec{\alpha} \in U^{\vec{X}}$ for the tuple $\alpha(\vec{X})$. This allows us to encode injective functions as tuples over U e.g., $\vec{\mathrm{EV}}_g := \{\vec{\alpha} \mid \alpha \in \mathrm{EV}_g\}$. Using Claim 19 we can construct $\vec{\mathrm{EV}}_g$ inductively over C_n.

- Let g be a constant input gate, then $\mathrm{sp}(g)$ is empty. If $\Sigma(g) = 0$, then $\Gamma(g) = \emptyset$ and $\vec{\mathrm{EV}}_g = \emptyset$. Otherwise $\Sigma(g) = 1$, then $\Gamma(g)$ is all bijections and $\vec{\mathrm{EV}}_g = \{\langle\rangle\}$, i.e., the set containing the empty tuple.
- Let g be a relational gate with $\Sigma(g) = R \in \tau$, then $\mathrm{sp}(g)$ is the set of elements in the tuple $\Lambda_R(g)$. By definition we have $\vec{\mathrm{EV}}_g = \{\vec{\alpha} \in U^{\vec{\mathrm{sp}}(g)} \mid \alpha(\Lambda_R(g)) \in R^{\mathcal{A}}\}$.
- Let $\Sigma(g) = \mathrm{AND}$ and consider $\vec{\alpha} \in U^{\vec{\mathrm{sp}}(g)}$. By Claim 19, $\vec{\alpha} \in \vec{\mathrm{EV}}_g$ iff $\vec{A_h} = \vec{\mathrm{EV}}_h$ for *every* child h of g, i.e., for *every* child h and *every* $\beta \in U^{\mathrm{sp}(h)}$ with $\alpha \sim \beta$, we have $\vec{\beta} \in \vec{\mathrm{EV}}_h$.
- Let $\Sigma(g) = \mathrm{OR}$ and consider $\vec{\alpha} \in U^{\vec{\mathrm{sp}}(g)}$. By Claim 19, $\vec{\alpha} \in \vec{\mathrm{EV}}_g$ iff there is a child h of g where $\vec{A_h} \cap \vec{\mathrm{EV}}_h$ is non-empty, i.e., for *some* child h of g and *some* $\beta \in U^{\mathrm{sp}(h)}$ with $\alpha \sim \beta$, we have $\vec{\beta} \in \vec{\mathrm{EV}}_h$.
- Let $\Sigma(g) = \mathrm{NOT}$ and consider $\vec{\alpha} \in U^{\vec{\mathrm{sp}}(g)}$. g has exactly one child h. Claim 19 implies that $\vec{\alpha} \in \vec{\mathrm{EV}}_g$ iff $\vec{A_h} \neq \vec{\mathrm{EV}}_h$, i.e., for *some* $\beta \in U^{\mathrm{sp}(h)}$ with $\alpha \sim \beta$, we have $\vec{\beta} \notin \vec{\mathrm{EV}}_h$.
- Let $\Sigma(g) = \mathrm{MAJ}$ and consider $\vec{\alpha} \in U^{\vec{\mathrm{sp}}(g)}$. Let H be the set of children of g and let $A_h := \{\beta \in U^{\mathrm{sp}(h)} \mid \beta \sim \alpha\}$. Then Claim 19 implies that $\vec{\alpha} \in \vec{\mathrm{EV}}_g$ if, and only if,

$$\sum_{h\in H} \frac{|\vec{A_h} \cap \vec{\mathrm{EV}}_h|}{|\vec{A_h}|} \geq \frac{|H|}{2}. \tag{1}$$

From $\vec{\mathrm{EV}}$ we can recover the query Q computed by C_n on the input structure $\mathcal{A}$ because the support of an output gate g is exactly the set of elements in the marking of g by Λ_Ω. In particular: $Q = \{\overline{a} \in U^q \mid \exists g \in G, \vec{\alpha} \in \vec{\mathrm{EV}}_g \text{ such that } \Lambda_\Omega(\alpha^{-1}(\overline{a})) = g\}$.

It is then straightforward (if laborious) to turn the inductive construction of $\vec{\mathrm{EV}}_g$ given above to a fixed-point formula defining the relation $V \subseteq [n]^t \times U^k$ by $V(g, \overline{a})$ if, and only if, $\tilde{a} \in \vec{\mathrm{EV}}_g$ in the structure $\mathcal{A}^{\leq}$, where $\tilde{a}$ is the restriction of the tuple $\overline{a}$ to $|\mathrm{sp}(g)|$ elements. Here t is the arity of the FP-interpretation of the circuit C_n in the structure $\mathcal{A}^{\leq}$. From this we get a formula defining the query Q given by the circuit family $\mathcal{C}$.

The only use of counting operators in the construction of the formula is in translating the inductive step corresponding to majority gates. Thus, the formula we obtain is one of $\mathrm{FP} + \leq$ if $\mathbb{B}$ is the standard basis and of FPC if $\mathbb{B}$ is the majority basis. Moreover, if the family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ is not P-uniform, but given by an advice function Υ, we get an equivalent formula of $\mathrm{FP} + \Upsilon$ (for the standard basis) or $\mathrm{FPC} + \Upsilon$ (for the majority basis).

On the other hand, formulas of $\mathrm{FP} + \leq$ can be translated into P-uniform families of symmetric Boolean circuits by standard methods and similar translations hold for FPC and $\mathrm{FP} + \Upsilon$. Putting this all together gives us our main theorem.

▶ **Theorem 20** (Main). *The following pairs of classes define the same queries on structures:*

1. *Symmetric* P*-uniform Boolean circuits and* $\mathrm{FP} + \leq$.
2. *Symmetric* P*-uniform majority circuits and* FPC.
3. *Symmetric* P/poly*-uniform Boolean circuits and* $\mathrm{FP} + \Upsilon$.
4. *Symmetric* P/poly*-uniform majority circuits and* $\mathrm{FPC} + \Upsilon$.

One consequence is that properties of graphs which we know not to be definable in FPC are also not decidable by P-uniform families of symmetric circuits. The results of Cai-Fürer-Immerman [3] give graph properties that are polynomial-time decidable, but not definable in FPC. Furthermore, there are a number of natural NP-complete graph problems known not to be definable in FPC, including Hamiltonicity and 3-colourability (see [4]). Indeed, all these proofs actually show that these properties are not even definable in the infinitary logic with a bounded number of variables and counting ($C^{\omega}_{\infty\omega}$—see [9]). Since it is not difficult to show that formulas of $\mathrm{FPC} + \Upsilon$ can be translated into $C^{\omega}_{\infty\omega}$, we have the following.

▶ **Corollary 21.** *Hamiltonicity and 3-colourability of graphs are not decidable by families of* P/poly*-uniform symmetric majority circuits.*

5 Coherent and Locally Polynomial Circuits

Otto [10] studies families of rigid symmetric Boolean circuits deciding properties of structures where the families satisfy two uniformity properties. Informally, a circuit family $\mathcal{C} := (C_n)_{n\in\mathbb{N}}$ is *coherent* if C_n appears as a subcircuit consisting of exactly the gates fixed by $\mathrm{Sym}_{[m]\setminus[n]}$ of all but finitely many of the circuits C_m at input length $m > n$. Second, $\mathcal{C}$ is *locally polynomial of degree* k if the size of the orbit of every wire in C_n is at most n^k. The main result [10, Theorem 6] is that coherent locally-polynomial of degree k families of symmetric $(\mathbb{B}_{\mathrm{std}}, \tau)$-circuits computing Boolean properties of $\mathrm{fin}[\tau]$ correspond to infinitary FO with k variables. In Otto's definition, individual circuits in the family may themselves be infinite, as the only size restriction is on the orbits of wires. The theorem also shows that if the circuit families are constant depth they correspond to the fragment of FO with k variables.

In all notions of uniformity we consider the circuits are of polynomial size. The Support Theorem can be used to establish a direct connection between polynomial-size symmetric circuit families and the locally-polynomial coherent symmetric families.

▶ **Proposition 22** (Informal). *Let* $\mathcal{C} := (C_n)_{n\in\mathbb{N}}$ *be a family of rigid symmetric Boolean circuits.*

1. *If* $\mathcal{C}$ *is locally-polynomial and coherent, then* $\mathcal{C}$ *is polynomial size.*
2. *If* $\mathcal{C}$ *is polynomial size, then* $\mathcal{C}$ *is locally polynomial.*

Since there are properties definable in an infinitary logic with finitely many variables that are not decidable by polynomial-size circuits, it follows from the above proposition that the use of infinite circuits is essential in Otto's result.

Proposition 22 implies that all uniform circuit families we consider are locally polynomial. However, they are not necessarily coherent. Indeed there are Boolean circuit families uniformly definable in $\mathrm{FO} + \leq$ that are not coherent. To see this observe that such circuit families may include gates that are completely indexed by the number sort and hence are fixed under all automorphisms induced by permutations of the point sort. Moreover the number of such gates may increase as a function of input length. However, under the definition of coherence, the number of gates in each circuit of a coherent family that are not moved by any automorphism must be identical. Thus there are uniform circuits that are not coherent.

6 Future Directions

One of the original motivations for studying symmetric majority circuits was the hope that they had the power of choiceless polynomial time with counting (CPTC) [2], and that, perhaps, techniques from circuit complexity could improve our understanding of the relationship between CPTC and the invariant queries definable in polynomial time. However, because FPC $\subsetneq$ CPTC [5], our results indicate that symmetry is too much of a restriction on P-uniform circuit families to recover CPTC.

A natural way to weaken the concept of symmetry is to require that induced automorphisms exist only for a certain subgroup of the symmetric group. This interpolates between our notion of symmetric circuits and circuits on linearly-ordered structures, with the latter case occurring when the subgroup is the identity.

The Support Theorem is a fairly general statement about the structure of symmetric circuits and is largely agnostic to the particular semantics of the basis. To that end the Support Theorem may find application to circuits over bases not consider here. The Support Theorem can be applied to arithmetic circuits computing invariant properties of matrices over a field; e.g., the Permanent polynomial is invariant and one standard way to compute it is as a symmetric arithmetic circuit, i.e., Ryser's formula [11]. Finally, the form of the Support Theorem can, perhaps, be improved as the particular upper bound required on the orbit size does not appear to be fundamental to the conclusion it reaches.

Acknowledgments. The authors thank Dieter van Melkebeek for looking at an early draft of this paper. This research was supported by EPSRC grant EP/H026835.

References

1 M. Anderson and A. Dawar. On symmetric circuits and fixed-point logics. *arXiv 1401.1125*, 2014.
2 A. Blass, Y. Gurevich, and S. Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100:141–187, 1999.
3 J-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.
4 A. Dawar. A restricted second order logic for finite structures. *Information and Computation*, 143:154–174, 1998.
5 A. Dawar, D. Richerby, and B. Rossman. Choiceless polynomial time, counting and the Cai–Fürer–Immerman graphs. *Annals of Pure and Applied Logic*, 152(1):31–50, 2008.
6 L. Denenberg, Y. Gurevich, and S. Shelah. Definability by constant-depth polynomial-size circuits. *Information and Control*, 70(2):216–240, 1986.
7 H.D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 2006.
8 N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1-3):86–104, 1986.
9 M. Otto. *Bounded Variable Logics and Counting: A Study in Finite Models*, volume 9 of *Lecture Notes in Logic*. Springer-Verlag, 1997.
10 M. Otto. The logic of explicitly presentation-invariant circuits. In Dirk van Dalen and Marc Bezem, editors, *Computer Science Logic*, volume 1258 of *Lecture Notes in Computer Science*, pages 369–384. Springer Berlin Heidelberg, 1997.
11 H.J. Ryser. *Combinatorial Mathematics*. Mathematical Association of America, 1963.
12 M. Vardi. The complexity of relational query languages. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 137–146. ACM, 1982.

Throughput Maximization in the Speed-Scaling Setting*

Eric Angel[1], Evripidis Bampis[2], and Vincent Chau[1]

1 **IBISC, Université d'Evry Val d'Essonne, Evry, France**
{Eric.Angel,Vincent.Chau}@ibisc.univ-evry.fr
2 **Sorbonne Universités, UPMC Univ Paris 06, LIP6, Paris, France**
Evripidis.Bampis@lip6.fr

Abstract

We are given a set of n jobs and a single processor that can vary its speed dynamically. Each job J_j is characterized by its processing requirement (work) p_j, its release date r_j and its deadline d_j. We are also given a budget of energy E and we study the scheduling problem of maximizing the throughput (i.e. the number of jobs that are completed on time). While the preemptive energy minimization problem has been solved in polynomial time [Yao et al., FOCS'95], the complexity of the problem of maximizing the throughput remained open until now. We answer partially this question by providing a dynamic programming algorithm that solves the problem in pseudo-polynomial time. While our result shows that the problem is not strongly NP-hard, the question of whether the problem can be solved in polynomial time remains a challenging open question. Our algorithm can also be adapted for solving the weighted version of the problem where every job is associated with a weight w_j and the objective is the maximization of the sum of the weights of the jobs that are completed on time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problem

Keywords and phrases energy efficiency, dynamic speed scaling, offline algorithm, throughput, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.53

1 Introduction

The problem of scheduling n jobs with release dates and deadlines on a single processor that can vary its speed dynamically with the objective of minimizing the energy consumption has been first studied in the seminal paper by Yao et al. [12]. In this paper, we consider the problem of maximizing the throughput for a given budget of energy. Throughput is one of the most popular objectives in scheduling literature [5, 10]. Its maximization in the context of energy-related scheduling is very natural since mobile devices, such as mobile phones or computers, have a limited energy capacity depending on the quality of their battery. The maximization of the number of jobs or of the total weight of the jobs executed on time for a given budget of energy is of great importance. Different variants of the throughput maximization problem in the online setting have been studied in the literature, but surprisingly the status of the offline problem remained open.

Formally, we are given a set of n jobs $J = \{J_1, J_2, \ldots, J_n\}$, where each job J_j is characterized by its processing requirement (work) p_j, its release date r_j and its deadline d_j.

* Supported by the French Agency for Research under the DEFIS program TODO (ANR-09-EMER-010) and by the project PHC CAI YUANPEI (27927VE)



© Eric Angel, Evripidis Bampis, and Vincent Chau;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 53–62



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



We consider integer release dates, deadlines and processing requirements. (For simplicity, we suppose that the earliest released job is released at $t = 0$.) We assume that the jobs have to be executed by a single speed-scalable processor, i.e. a processor which can vary its speed over time (at a given time, the processor's speed can be any non-negative value). The processor can execute at most one job at each time. We measure the processor's speed in units of executed work per unit of time. If $s(t)$ denotes the speed of the processor at time t, then the total amount of work executed by the processor during an interval of time $[t, t')$ is equal to $\int_t^{t'} s(u)du$. Moreover, we assume that the processor's power consumption is a convex function of its speed. Specifically, at any time t, the power consumption of the processor is $P(t) = s(t)^\alpha$, where $\alpha > 1$ is a constant. Since the power is defined as the rate of change of the energy consumption, the total energy consumption of the processor during an interval $[t, t')$ is $\int_t^{t'} s(u)^\alpha du$. Note that if the processor runs at a constant speed s during an interval of time $[t, t')$, then it executes $(t' - t) \cdot s$ units of work and it consumes $(t' - t) \cdot s^\alpha$ units of energy.

Each job J_j can start being executed after or at its release date r_j. Moreover, the execution of a job may be suspended and continued later from the point of suspension. Given a budget of energy E, our objective is to find a schedule of maximum throughput whose energy does not exceed the budget E, where the throughput of a schedule is defined as the number of jobs which are completed on time, i.e. before their deadline. Observe that a job is completed on time if it is entirely executed during the interval $[r_j, d_j)$. By extending the well-known 3-field notation, this problem can be denoted as $1|pmtn, r_j, E|\sum U_j$. We also consider the weighted version of the problem where every job J_j is also associated with a weight w_j and the objective is no more the maximization of the cardinality of the jobs that are completed on time, but the maximization of the sum of their weights. We denote this problem as $1|pmtn, r_j, E|\sum w_j U_j$. In what follows, we consider the problem in the case where all jobs have arbitrary integer release dates, deadlines and processing requirement.

1.1 Related Works and our Contribution

A series of papers appeared for some online variants of throughput maximization: the first work that considered throughput maximization and speed scaling in the online setting has been presented by Chan et al. [6]. They considered the single processor case with release dates and deadlines and they assumed that there is an upper bound on the processor's speed. They are interested in maximizing the throughput, and minimizing the energy among all the schedules of maximum throughput. They presented an algorithm which is $O(1)$-competitive with respect to both objectives. Li [11] has also considered the maximum throughput when there is an upper bound in the processor's speed and he proposed a 3-approximation greedy algorithm for the throughput and a constant approximation ratio for the energy consumption. In [3], Bansal et al. improved the results of [6], while in [9], Lam et al. studied the 2-processors environment. In [8], Chan et al. defined the energy efficiency of a schedule to be the total amount of work completed in time divided by the total energy usage. Given an efficiency threshold, they considered the problem of finding a schedule of maximum throughput. They showed that no deterministic algorithm can have competitive ratio less than Δ, the ratio of the maximum to the minimum jobs' processing requirement. However, by decreasing the energy efficiency of the online algorithm the competitive ratio of the problem becomes constant. Finally, in [7], Chan et al. studied the problem of minimizing the energy plus a rejection penalty. The rejection penalty is a cost incurred for each job which is not completed on time and each job is associated with a value which is its importance. The authors proposed an $O(1)$-competitive algorithm for the case where the speed is unbounded and they showed

that no $O(1)$-competitive algorithm exists for the case where the speed is bounded. In what follows, we focus on the complexity status of the offline case for general instances. Angel et al. [1] were the first to consider the throughput maximization problem in the energy setting for the offline case. They studied the problem for a particular family of instances where the jobs have agreeable deadlines, i.e. for every pair of jobs J_i and J_j, $r_i \leq r_j$ if and only if $d_i \leq d_j$. They provided a polynomial time algorithm to solve the problem for agreeable instances. However, to the best of our knowledge, the complexity of the unweighted preemptive problem for arbitrary instances remained unknown until now. In this paper, we prove that there is a pseudo-polynomial time algorithm for solving the problem optimally. For the weighted version, the problem is $\mathcal{NP}$-hard even for instances in which all the jobs have common release dates and deadlines. Angel et al. [1] showed that the problem admits a pseudo-polynomial time algorithm for agreeable instances. Our algorithm for the unweighted case can be adapted for the weighted throughput problem with arbitrary release dates and deadlines solving the problem in pseudo-polynomial time. More recently, Antoniadis et al. [2] considered a generalization of the classical knapsack problem where the objective is to maximize the total profit of the chosen items minus the cost incurred by their total weight. The case where the cost functions are convex can be translated in terms of a weighted throughput problem where the objective is to select the most profitable set of jobs taking into account the energy costs. Antoniadis et al. presented a FPTAS and a fast 2-approximation algorithm for the non-preemptive problem where the jobs have no release dates or deadlines.

We present in this paper an optimal algorithm for throughput maximization when the preemption of jobs is allowed.

2 Preliminaries

Among the schedules of maximum throughput, we try to find the one of minimum energy consumption. Therefore, if we knew by an oracle the set of jobs J^*, $J^* \subseteq J$, which are completed on time in an optimal solution, we would simply have to apply an optimal algorithm for $1|pmtn, r_j, d_j|E$ for the jobs in J^* in order to determine a minimum energy schedule of maximum throughput for our problem. Such an algorithm has been proposed in [12]. Based on this observation, we can use in our analysis some properties of an optimal schedule for $1|pmtn, r_j, d_j|E$.

Let $t_1, t_2, \ldots, t_K$ be the time points which correspond to release dates and deadlines of the jobs so that for each release date and deadline there is a t_i value that corresponds to it. We number the t_i values in increasing order, i.e. $t_1 < t_2 < \ldots < t_K$. The following theorem is a consequence of the algorithm of Yao et al. [12] and was proved in [4].

▶ **Theorem 1.** *A feasible schedule for $1|pmtn, r_j, d_j|E$ is optimal if and only if all the following hold:*

1. *Each job J_j is executed at a constant speed s_j.*
2. *The processor is not idle at any time t such that $t \in (r_j, d_j]$, for all $J_j \in J$.*
3. *The processor runs at a constant speed during any interval $(t_i, t_{i+1}]$, for $1 \leq i \leq K-1$.*
4. *If others jobs are scheduled in the span $[r_j, d_j]$ of J_j, then their speed is necessarily greater or equal to the speed of J_j.*

Theorem 1 is also satisfied by the optimal schedule of $1|pmtn, r_j, E|\sum U_j$ for the jobs in J^*. In the following, we suppose that the jobs are sorted in non-decreasing order of their deadlines (EDF order), i.e. $d_1 \leq d_2 \leq \ldots \leq d_n$. Moreover, we suppose that the release dates, the deadlines and the processing requirements are integer.

▶ **Definition 2.** Let $J(k,s,t) = \{J_j \mid j \leq k \text{ and } s \leq r_j < t\}$ be the set of jobs, among the k first ones w.r.t. the EDF order, whose release dates are within s and t.

▶ **Lemma 3.** *The total period in which the processor runs at a same speed in an optimal solution for $1|pmtn, r_j, d_j|E$ has an integer length.*

Proof. The total period is defined by a set of intervals $(t_i, t_{i+1}]$ for $1 \leq i \leq K-1$ thanks to the property 3) in Theorem 1. Since each t_i corresponds to some release date or some deadline, then $t_i \in \mathbb{N},\ 1 \leq i \leq K$. Thus every such period has necessarily an integer length. ◀

▶ **Definition 4.** Let $L = d_{max} - r_{min}$ be the span of the whole schedule. To simplify the notation, we assume that $r_{min} = 0$.

▶ **Definition 5.** Let $P = \sum_j p_j$ be the total processing requirement of all the jobs.

▶ **Definition 6.** We call an EDF schedule, a schedule in which at any time, the processor schedules the job that has the smallest deadline among the set of available jobs at this time.

In the sequel, all the considered schedules are EDF schedules.

3 The Dynamic Program and its Correctness

In this part, we propose an optimal algorithm which is based on dynamic programming depending on the span length L and the total processing requirement P. As mentioned previously, among the schedules of maximum throughput, our algorithm constructs a schedule with the minimum energy consumption.

For a subset of jobs $S \subseteq J$, a schedule which involves only the jobs in S will be called a S-schedule.

▶ **Definition 7.** Let $G_k(s,t,u)$ be the minimum energy consumption of a S-schedule with $S \subseteq J(k,s,t)$ such that $|S| = u$ and such that the jobs in S are entirely scheduled in $[s,t]$.

Given a budget of energy E that we cannot exceed, the objective function is $\max\{u \mid G_n(0, d_{max}, u) \leq E;\ 0 \leq u \leq n\}$.

▶ **Definition 8.** Let $F_{k-1}(x,y,u,\ell,i,a,h)$ be the minimum energy consumption of a S-schedule with $S \subseteq J(k-1,x,y)$ such that $|S| = u$ and such that the jobs in S are entirely scheduled in $[x,y]$ during at most $a + h \times \frac{\ell}{i}$ unit times. Moreover, we assume that each maximal block of consecutive jobs of S starts at a release date and has a length equal to $a' + h' \times \frac{\ell}{i}$ with $a', h' \in \mathbb{N}$.

Next, we define the set of all important dates of an optimal schedule in which every job can start and end, and we show that the size of this set is pseudo-polynomial.

▶ **Definition 9.** Let $\Omega = \{r_j \mid j = 1, \ldots, n\} \cup \{d_j \mid j = 1, \ldots, n\}$.

▶ **Definition 10.** Let $\Phi = \{s + h \times \frac{\ell}{i} \leq L \mid i = 1, \ldots, P;\ h = 0, \ldots, i;\ s = 0, \ldots, L;\ \ell = 1, \ldots, L\}$

▶ **Proposition 11.** *There exists an optimal schedule $\mathcal{O}$ in which for each job, its starting times and finish times belong to the set Φ, and such that each job is entirely executed with a speed $\frac{i}{\ell}$ for some $i = 1, \ldots, P$ and $\ell = 1, \ldots, L$.*

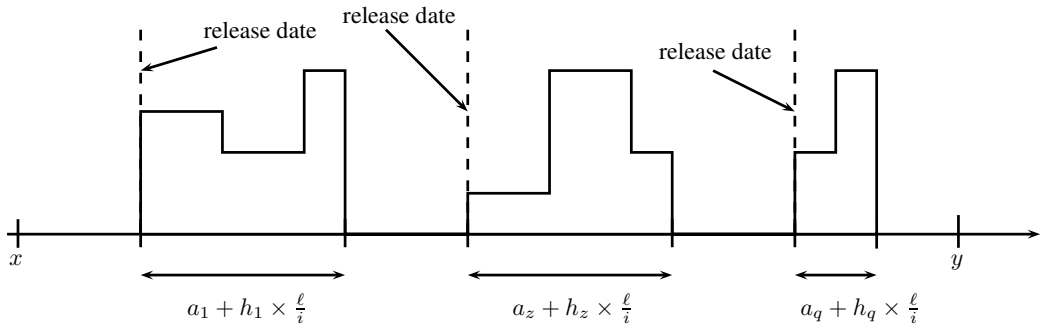


Figure 1 Illustration of $F_{k-1}(x, y, u, \ell, i, a, h)$ in Definition 8 with respect to $\sum_{z=1}^{q} a_z + h_z \times \frac{\ell}{i} = a + h \times \frac{\ell}{i}$.

Proof. W.l.o.g. we can consider that each job has a unit processing requirement. If it is not the case, we can split a job J_j into p_j jobs, each one with a unit processing requirement.

We briefly explain the algorithm proposed in [12] which gives an optimal schedule. At each step, it selects the (critical) interval $I = [s, t]$ with s and $t > s$ in $\Omega = \{r_j \mid j = 1, \ldots, n\} \cup \{d_j \mid j = 1, \ldots, n\}$, such that $s_I = \frac{|\{J_j \mid s \leq r_j \leq d_j \leq t\}|}{t-s}$ is maximum. All the jobs inside this interval are executed at the speed s_I, which is of the form $\frac{i}{\ell}$ for some $i = 1, \ldots, P$ and $\ell = 1, \ldots, L$, and according to the EDF order. This interval cannot be used any more, and we recompute a new critical interval without considering the jobs and the previous critical intervals, until all the jobs have been scheduled.

We can remark that the length of each critical interval (at each step) $I = [s, t]$ is an integer. This follows from the fact that $s = r_z \in \mathbb{N}$ for some job J_z, and $t = d_j \in \mathbb{N}$ for some job J_j, moreover we remove integer lengths at each step (the length of previous critical intervals which intersect the current one), so the new considered critical interval has always an integer length.

Then we can define every potential starting time or completion time of each job in this interval. We first prove that the completion time of a job in a continuous critical interval, i.e. a critical interval which has an empty intersection with all other critical intervals, belongs to Φ. Let J_k be any job in a continuous critical interval and let x and y be respectively its starting and completion times. Then there is no idle time between $s = r_f$ (for some J_f) and y since it is a critical interval. Let $v = \frac{i}{\ell}$ be the processor speed in this interval and $p = \frac{\ell}{i}$ be the processing time of a job (Recall that each job has the same processing requirement). The jobs that are executed (even partially) between x and y are not executed neither before x nor after y since we consider an EDF schedule. Thus $y - x$ is a multiple of p. Two cases may occur:

- Either J_k causes a preemption and hence $x = r_k$,
- or J_k does not cause any preemption and hence the jobs that are executed between s and x, are fully scheduled in this interval. Consequently, $x - s$ is a multiple of p.

In both cases, there is a release date r_g (either r_k or r_f) such that between r_g and y, the processor is never idle and such that y is equal to r_g modulo p. On top of that, the distance between r_g and t is not greater than $n \times p$. Hence, $y \in \Phi$. Now consider the starting time of any job. This time point is either the release date of the job or is equal to the completion time of the "previous" one. Thus, starting times also belong to Φ.

Now we consider the starting and the completion times of a job in a critical interval I in which there is at least another critical interval (with greater speeds) included in I or

intersecting I. Let A be the union of those critical intervals. Since the jobs of I cannot be scheduled during the intervals A, the starting time and completion time of these jobs have to be (right)-shifted by an integer value (since each previously critical interval has an integer length). Thus the starting time and completion time of all the jobs still belong to Φ. ◀

▶ **Proposition 12.** *One has*

$$G_k(s,t,u) = \min \begin{cases} G_{k-1}(s,t,u) \\ \min\limits_{\substack{x\in\Phi \\ 0\le u_1\le u \\ 0\le u_2\le u \\ 0\le u_1+u_2\le u-1 \\ 0\le a\le L;\ 1\le \ell\le L \\ 1\le i\le P;\ 0\le h\le P \\ y-x=a+(p_k+h)\frac{\ell}{i} \\ r_k\le x\le y\le d_k}} \left\{ \begin{array}{l} G_{k-1}(s,x,u_1) + F_{k-1}(x,y,u_2,\ell,i,a,h) \\ +\left(\frac{i}{\ell}\right)^{\alpha-1} p_k + G_{k-1}(y,t,u-u_1-u_2-1) \end{array} \right\} \end{cases}$$

$G_0(s,t,0) = 0\ \forall s,t \in \Phi$
$G_0(s,t,u) = +\infty\ \forall s,t \in \Phi$ *and* $u > 0$

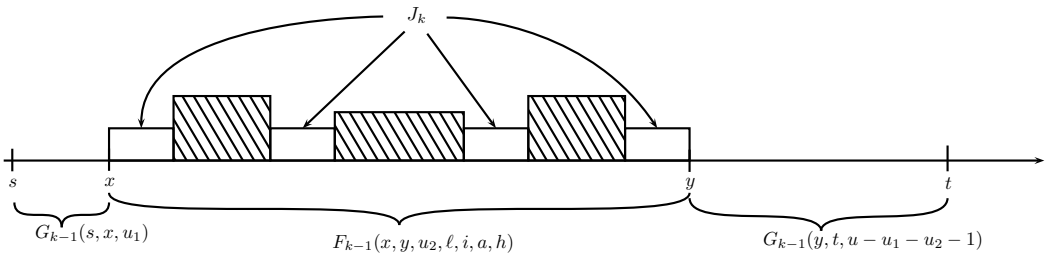


Figure 2 Illustration of Proposition 12 where x is the first starting time of J_k and y is the last completion time of J_k.

Proof. Let G' be the right hand side of the formula, G'_1 be the first line of G' and G'_2 be the second line of G'.

We first prove that $G_k(s,t,u) \le G'$.

Since $J(k-1,s,t) \subseteq J(k,s,t)$, then $G_k(s,t,u) \le G_{k-1}(s,t,u) = G'_1$.

Now consider a schedule $\mathcal{S}_1$ that realizes $G_{k-1}(s,x,u_1)$, a schedule $\mathcal{S}_2$ that realizes $F_{k-1}(x,y,u_2,\ell,i,a,h)$ such that $y - x = a + (p_k + h) \times \frac{\ell}{i}$ and a schedule $\mathcal{S}_3$ that realizes $G_{k-1}(y,t,u-u_1-u_2-1)$. We build a schedule with $\mathcal{S}_1$ from s to x, with $\mathcal{S}_2$ from x to y and with $\mathcal{S}_3$ from y to t.

Since $F_{k-1}(x,y,u_2,\ell,i,a,h)$ is a schedule where the processor executes the jobs during at most $a + h \times \frac{l}{i}$ unit times and we have $y - x = a + (p_k + h) \times \frac{\ell}{i}$, then there is at least $p_k \times \frac{\ell}{i}$ units time for J_k. Thus J_k can be scheduled with speed $\frac{i}{\ell}$ during $[x,y]$.

Obviously, the subsets $J(k-1,s,x), J(k-1,x,y)$ and $J(k-1,y,t)$ do not intersect, so this is a feasible schedule, and its cost is G'_2. Hence $G_k(s,t,u) \le G'_2$.

We now prove that $G' \le G_k(s,t,u)$.

If $J_k \notin \mathcal{O}$ such that $\mathcal{O}$ realizes $G_k(s,t,u)$, then $G'_1 = G_k(s,t,u)$.

Now, let us consider the case $J_k \in \mathcal{O}$.

We denote by $\mathcal{X}$ the schedule that realizes $G_k(s,t,u)$ in which the first starting time x of J_k is maximal, and in which y is the last completion time of J_k is also maximal. According to Proposition 11, we assume that $x, y \in \Phi$. We split $\mathcal{X}$ (which is an EDF schedule) into three sub-schedules $\mathcal{S}_1 \subseteq J(k-1,s,x)$, $\mathcal{S}_2 \subseteq J(k-1,x,y) \cup \{J_k\}$ and $\mathcal{S}_3 \subseteq J(k-1,y,t)$.

We claim that we have the following properties:

P1) all the jobs of $\mathcal{S}_1$ are released in $[s,x]$ and are completed before x,
P2) all the jobs of $\mathcal{S}_2$ are released in $[x,y]$ and are completed before y,
P3) all the jobs of $\mathcal{S}_3$ are released in $[y,t]$ and are completed before t.

We prove P1

Suppose that there is a job $J_j \in \mathcal{S}_1$ which is not completed before x. Then we can swap some part of J_j of length ε which is scheduled after x with some part of J_k of length ε at time x. This can be done since we have $d_j \leq d_k$. Thus we have a contradiction with the fact that x was maximal.

We prove P2

Similarly, suppose that there is a job $J_j \in \mathcal{S}_2$ which is not completed before y. Then we can swap some part of J_j of length ℓ which is scheduled after y with some part of J_k of length ℓ in $[x,y]$. This can be done since we have $d_j \leq d_k$. Thus we have a contradiction with the fact that y was maximal.

We prove P3

If there exists a job in $\mathcal{S}_3$ which is not entirely executed at time t, then the removal of this job would lead to a lower energy consumption schedule for $\mathcal{S}_3$ with the same throughput value. This contradicts the definition of $G_{k-1}(y,t,|\mathcal{S}_3|)$.

Let us now consider the schedule $\mathcal{S}'_2 = \mathcal{S}_2 \setminus J_k$ in $[x,y]$. Since $[x,y] \subseteq [r_k,d_k]$, thanks to property 4) of Theorem 1, the speeds of jobs in $\mathcal{S}'_2$ are necessarily greater than or equal to the speed of J_k. Let us consider any maximal block b of consecutive jobs in $\mathcal{S}'_2$. This block can be partitioned into two sub-blocks b_1 and b_2 such that b_1 (resp. b_2) contains all the jobs of b which are scheduled with a speed equal to (resp. strictly greater than) the speed of J_k. All the jobs scheduled in block b are also totally completed in b (this comes from the EDF property and because J_k has the biggest deadline). Notice that the speed of J_k is equal to $\frac{i}{\ell}$ for some value $i = 1,\ldots,P$ and $\ell = 1,\ldots,L$ thanks to Proposition 11. Thus the total processing time of b_1 is necessarily $h' \times \frac{\ell}{i}$. Moreover since from property 3 of Theorem 1, all the speed changes occur at time $t_i \in \mathbb{N}$, the block b_2 has an integer length. Therefore, every block b has a length equal to $a' + h' \times \frac{\ell}{i}$ and the total processing time of $\mathcal{S}'_2$ is $a + h \times \frac{\ell}{i}$. Furthermore, every block b in $\mathcal{S}'_2$ starts at a release date (this comes from the EDF property). On top of that, we have $y - x = a + (h + p_k) \times \frac{\ell}{i}$ with $a = 0,1,\ldots,L$ and $h = 0,\ldots,i$. Moreover, every block b in $\mathcal{S}'_2$ starts at a release date (this comes from the EDF property). Hence the cost of the schedule $\mathcal{S}'_2$ is greater than $F_{k-1}(x,y,|\mathcal{S}'_2|,\ell,i,a,h)$. The energy consumption of J_k is exactly $p_k \times (\frac{i}{\ell})^{\alpha-1}$.

Similarly, the cost of the schedule $\mathcal{S}_1$ is greater than $G_{k-1}(s,x,|\mathcal{S}_1|)$ and the cost of $\mathcal{S}_3$ is greater than $G_{k-1}(y,t,|\mathcal{S}_3|)$.

Therefore, $G_k(s,t,u) \geq G_{k-1}(s,x,|\mathcal{S}_1|) + F_{k-1}(x,y,|\mathcal{S}_2|,\ell,i,a,h) + G_{k-1}(y,t,|\mathcal{S}_3|) + p_k\left(\frac{i}{\ell}\right)^{\alpha-1} = G'_2$ and $G_k(s,t,u) \geq G'$. ◀

▶ **Proposition 13.** *One has*

$$F_{k-1}(x,y,u,\ell,i,a,h) = \min_{\substack{0\le a'\le a;\ 0\le h'\le h\\ x\le x'=r_j\le y;\ j\le k\\ 1\le\beta\le u\\ y'=x'+a'+h'\times\frac{\ell}{i}\le y}} \{G_{k-1}(x',y',\beta)+F_{k-1}(y',y,u-\beta,\ell,i,a-a',h-h')\}$$

$$F_{k-1}(x,y,0,\ell,i,a,h) = 0$$
$$F_{k-1}(x,y,u,\ell,i,0,0) = +\infty$$

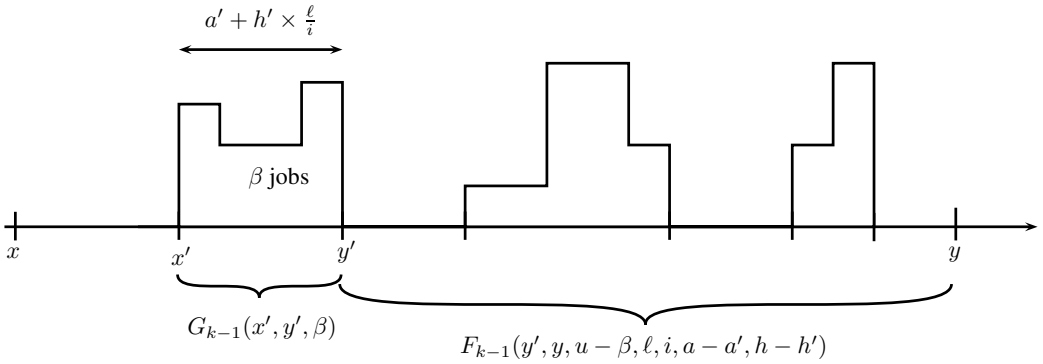


Figure 3 Illustration of Proposition 13.

Proof. Let F' be the right hand side of the equation.

We first prove that $F_{k-1}(x,y,u,\ell,i,a,h) \le F'$.

Let us consider a schedule $\mathcal{S}_1$ that realizes $G_{k-1}(x',y',\beta)$ and a schedule $\mathcal{S}_2$ that realizes $F_{k-1}(y',y,u-\beta,\ell,i,a',h')$. We suppose that the processor is idle during $[x,x']$. We build a schedule with an empty set from x to x', with $\mathcal{S}_1$ from x' to y' and with $\mathcal{S}_2$ from y' to y.

Obviously, the subsets $J(k-1,x,z)$ and $J(k-1,z,y)$ do not intersect, so this is a feasible schedule, and its cost is F', thus $F_{k-1}(x,y,u,f,\ell,i) \le F'$.

We now prove that $F' \le F_{k-1}(x,y,u,\ell,i,a,h)$.

Let $\mathcal{O}$ be an optimal schedule that realizes $F_{k-1}(x,y,u,\ell,i,a,h)$ such that x' is the first starting time of the schedule and y' is the completion time of the first block of jobs in $\mathcal{O}$. We split it into two sub-schedules $\mathcal{S}_1 \subseteq J(k-1,x',y')$ and $\mathcal{S}_2 \subseteq J(k-1,y',y)$ such that the value of x' is maximal and the value of y' is also maximal.

Then $y'-x' = a'+h'\times\frac{\ell}{i}$ for some value $a' = 0,\dots,a$ and $h' = 0,\dots,h$ by definition. Thus we can assume that the jobs in $\mathcal{S}_2$ have to be scheduled during at most $(a-a')+(h-h')\times\frac{\ell}{i}$ units of time in $[y',y]$. We claim that x' is a release date by definition.

Moreover, we claim that all the jobs of $\mathcal{S}_2$ are released in $[y',y]$ and are completed before y. If there exists a job in $\mathcal{S}_2$ which is not completed at time t, then the removal of this job would lead to a lower energy consumption schedule for $\mathcal{S}_2$ which contradicts the definition of $F_{k-1}(y',y,|\mathcal{S}_2|,\ell,i,a-a',h-h')$.

Then the restriction $\mathcal{S}_1$ of $\mathcal{O}$ in $[x',y']$ is a schedule that meets all constraints related to $G_{k-1}(x',y',|\mathcal{S}_1|)$. Hence its cost is greater than $G_{k-1}(x',y',|\mathcal{S}_1|)$. Similarly, the restriction $\mathcal{S}_2$ of $\mathcal{O}$ to $[y',y]$ is a schedule that meets all constraints related to $F_{k-1}(y',y,|\mathcal{S}_2|,\ell,i,a-a',h-h')$.

Thus $F' \le F_{k-1}(x,y,u,\ell,i,a,h)$. ◀

▶ **Theorem 14.** *The preemptive throughput maximization problem can be solved in* $O(n^6L^9P^9)$ *time and in* $O(nL^6P^6)$ *space.*

Proof. The values of $G_k(s,t,u)$ are stored in a multi-dimensional array of size $O(|\Phi|^2n^2)$. Each value need $O(|\Phi|n^2L^2P^2r(F))$ time to be computed where $r(F)$ is the running time for computing $F_{k-1}(x,y,u,\ell,i,a,h)$. Since we fix every value of x,y,u,ℓ,i,a,h in the minimization step, the table F does not need to be pre-computed. Then the running time is $O(n^2LP)$ for each value of F. Therefore, the total running time of the dynamic programming is $O(n^6L^9P^9)$. Moreover, the values of $F_{k-1}(x,y,u,\ell,i,a,h)$ are stored in a multi-dimensional array (since we don't need to remember the F_i values for $i < k-1$) of size $O(n|\Phi|^2L^2P^2) = O(nL^6P^6)$. ◀

The dynamic program can be adapted for the weighted version of the problem and has a running time of $O(n^2W^4L^9P^9)$ where W is the sum of the weight of all jobs. This can be done by considering the total weight of completed jobs of a schedule instead of considering the number of completed jobs. More formally, this can be done by modifying the definitions of G_k and F_{k-1} in the following way:

- $G_k(s,t,w)$ is the minimum energy consumption of a S-schedule with $S \subseteq J(k,s,t)$ such that $\sum_{J_j \in S} w_j \geq w$ and such that the jobs in S are entirely scheduled in $[s,t]$, and
- $F_{k-1}(x,y,w,\ell,i,a,h)$ is the minimum energy consumption of a S-schedule with $S \subseteq J(k-1,x,y)$ such that $\sum_{J_j \in S} w_j \geq w$ and such that the jobs in S are entirely scheduled in $[x,y]$ during at most $a + h \times \frac{\ell}{i}$ unit times. As for the cardinality case, we assume that each maximal block of consecutive jobs of S starts at a release date and has a length equal to $a' + h' \times \frac{\ell}{i}$ with $a', h' \in \mathbb{N}$.

4 Conclusion

In this paper, we proved that there is a pseudo-polynomial time algorithm for solving the problem optimally. This result is a first (partial) answer to the complexity status of the throughput maximization problem in the offline setting. Our result shows that the problem is not strongly NP-hard, but the question of whether there is a polynomial time algorithm for it remains a challenging open question.

References

1 Eric Angel, Evripidis Bampis, Vincent Chau, and Dimitrios Letsios. Throughput maximization for speed-scaling with agreeable deadlines. In T.-H. Hubert Chan, Lap Chi Lau, and Luca Trevisan, editors, *TAMC*, volume 7876 of *Lecture Notes in Computer Science*, pages 10–19. Springer, 2013.

2 Antonios Antoniadis, Chien-Chung Huang, Sebastian Ott, and José Verschae. How to pack your items when you have to buy your knapsack. In Krishnendu Chatterjee and Jiri Sgall, editors, *MFCS*, volume 8087 of *Lecture Notes in Computer Science*, pages 62–73. Springer, 2013.

3 Nikhil Bansal, Ho-Leung Chan, Tak Wah Lam, and Lap-Kei Lee. Scheduling for speed bounded processors. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 409–420. Springer, 2008.

4 Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), 2007.

5 Peter Brucker. *Scheduling Algorithms*. Springer Publishing Company, Incorporated, 5th edition, 2010.

6 Ho-Leung Chan, Wun-Tat Chan, Tak Wah Lam, Lap-Kei Lee, Kin-Sum Mak, and Prudence W. H. Wong. Energy efficient online deadline scheduling. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *SODA*, pages 795–804. SIAM, 2007.

7 Ho-Leung Chan, Tak Wah Lam, and Rongbin Li. Tradeoff between energy and throughput for online deadline scheduling. In Klaus Jansen and Roberto Solis-Oba, editors, *WAOA*, volume 6534 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 2010.

8 Joseph Wun-Tat Chan, Tak Wah Lam, Kin-Sum Mak, and Prudence W. H. Wong. Online deadline scheduling with bounded energy efficiency. In Jin yi Cai, S. Barry Cooper, and Hong Zhu, editors, *TAMC*, volume 4484 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 2007.

9 Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Energy efficient deadline scheduling in two processor systems. In Takeshi Tokuyama, editor, *ISAAC*, volume 4835 of *Lecture Notes in Computer Science*, pages 476–487. Springer, 2007.

10 E. L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26:125–133, 1990.

11 Minming Li. Approximation algorithms for variable voltage processors: Min energy, max throughput and online heuristics. *Theor. Comput. Sci.*, 412(32):4074–4080, 2011.

12 F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382. IEEE Computer Society, 1995.

Efficient Computation of Optimal Energy and Fractional Weighted Flow Trade-off Schedules

Antonios Antoniadis[*1], Neal Barcelo[†2], Mario Consuegra[‡3], Peter Kling[§4], Michael Nugent[5], Kirk Pruhs[¶6], and Michele Scquizzato[‖7]

1,2,5,6,7 University of Pittsburgh, Pittsburgh, USA
3 Florida International University, Miami, USA
4 University of Paderborn, Paderborn, Germany

Abstract

We give a polynomial time algorithm to compute an optimal energy and fractional weighted flow trade-off schedule for a speed-scalable processor with discrete speeds. Our algorithm uses a geometric approach that is based on structural properties obtained from a primal-dual formulation of the problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases scheduling, flow time, energy efficiency, speed scaling, primal-dual

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.63

1 Introduction

It seems to be a universal law of technology in general, and information technology in particular, that higher performance comes at the cost of energy efficiency. Thus a common theme of green computing research is how to manage information technologies so as to obtain the proper balance between these conflicting goals of performance and energy efficiency. Here the technology we consider is a speed-scalable processor, as manufactured by the likes of Intel and AMD, that can operate in different modes, where each mode has a different speed and power consumption, and the higher speed modes are less energy-efficient in that they consume more energy per unit of computation. The management problem that we consider is how to schedule jobs on such a speed-scalable processor in order to obtain an optimal trade-off between a natural performance measure (fractional weighted flow) and the energy used. Our main result is a polynomial time algorithm to compute such an optimal trade-off schedule.

We want to informally elaborate on the statement of our main result. Fully formal definitions can be found in Section 3. We need to explain how we model the processors, the jobs, a schedule, our performance measure, and the energy-performance trade-off:

* Supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).
† This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1247842.
‡ Supported by NSF Graduate Research Fellowship DGE-1038321.
§ Supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901) and by the Graduate School on Applied Network Science (GSANS).
¶ Supported in part by NSF grants CCF-1115575, CNS-1253218 and an IBM Faculty Award.
‖ Supported in part by a fellowship of "Fondazione Ing. Aldo Gini", University of Padova, Italy.



© Antonios Antoniadis, Neal Barcelo, Mario Consuegra, Peter Kling, Michael Nugent, Kirk Pruhs, and Michele Scquizzato;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 63–74



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



The Speed-Scalable Processor: We assume that the processor can operate in any of a discrete set of modes, each with a specified speed and power consumption.

The Jobs: Each job has a release time when the job arrives in the system, a volume of work (think of a unit of work as being an infinitesimally small instruction to be executed), and a total importance or weight. The ratio of the weight to the volume of work specifies the density of the job, which is the importance per unit of work of that job.

A Schedule: A schedule specifies, for each real time, the job that is being processed and the mode of the processor.

Our Performance Measure of Fractional Weighted Flow: The fractional weighted flow of a schedule is the total over all units of work (instructions) of how much time that work had to wait from its release time until that work was executed on the processor, times the weight (aggregate importance) of that unit of work. So work with higher weight is considered to be more important. Presumably the weights are specified by higher-level applications that have knowledge of the relative importance of various jobs.

Optimal Trade-off Schedule: An optimal trade-off schedule minimizes the fractional weighted flow plus the energy used by the processor (energy is just power integrated over time). To gain intuition, assume that at time zero a volume p of work of weight w is released. Intuitively/Heuristically one might think that the processor should operate in the mode i that minimizes $w\frac{p}{2s_i} + P_i\frac{p}{s_i}$, where s_i and P_i are the speed and power of mode i respectively, until all the work is completed; In this schedule the time to finish all the work is $\frac{p}{s_i}$, the fractional weighted flow is $w\frac{p}{2s_i}$, and the total energy usage is $P_i\frac{p}{s_i}$. So the larger the weight w, the faster the mode that the processor will operate in. Thus intuitively the application-provided weights inform the system scheduler as to which mode to operate in so as to obtain the best trade-off between energy and performance. (The true optimal trade-off schedule for the above instance is more complicated as the speed will decrease as the work is completed.)

In Section 2 we explain the relationship of our result to related results in the literature. Unfortunately both the design and analysis of our algorithm are complicated, so in Section 4 we give an overview of the main conceptual ideas before launching into details in the subsequent sections. In Section 5 we present the obvious linear programming formulation of the problem, and discuss our interpretation of information that can be gained about optimal schedules from both the primal and dual linear programs. In Section 6 we use this information to develop our algorithm. Finally in Section 7 we analyze the running time of our algorithm. Due to space limitations, many of the details are left for the full version of the paper.

2 Related Results

To the best of our knowledge there are three papers in the algorithmic literature that study computing optimal energy trade-off schedules. All of these papers assume that the processor can run at any non-negative real speed, and that the power used by the processor is some nice function of the speed, most commonly the power is equal to the speed raised to some constant α. Essentially both [2, 13] give polynomial time algorithms for the special case of our problem where the densities of all units of work are the same. The algorithm in [13] is a homotopic optimization algorithm that intuitively traces out all schedules that are Pareto-optimal with respect to energy and fractional flow, one of which must obviously be the optimal energy trade-off schedule. The algorithm in [2] is a dynamic programming algorithm. [2] also deserves credit for introducing the notion of trade-off schedules. [7] gave

a polynomial-time algorithm for recognizing an optimal schedule. [7] also showed that the optimal schedule evolves continuously as a function of the importance of energy, implying that a continuous homotopic algorithm is, at least in principle, possible. However, [7] was not able to provide any bound, even exponential, on the time of this algorithm, nor was [7] able to provide any way to discretize this algorithm.

To reemphasize, the prior literature [2, 13, 7] on our problem assumes that the set of allowable speeds is continuous. Our setting of discrete speeds both more closely models the current technology, and seems to be algorithmically more challenging. In [7] the recognition of an optimal trade-off schedule in the continuous setting is essentially a direct consequence of the KKT conditions of the natural convex program, as it is observed that there is essentially only one degree of freedom for each job in any plausibly optimal schedule, and this degree of freedom can be recovered from the candidate schedule by looking at the speed that the job is run at any time that the job is run. In the discrete setting, we shall see that there is again essentially only one degree of freedom for each job, but unfortunately one cannot easily recover the value of this degree of freedom by examining the candidate schedule. Thus we do not know of any simple way to even recognize an optimal trade-off schedule in the discrete setting.

One might also reasonably consider the performance measure of the aggregate weighted flow over jobs (instead of work), where the flow of a job is the amount of time between when the job is released and when the last bit of work of that job is finished. In the context that the jobs are flight queries to a travel site, aggregating over the delay of jobs is probably more appropriate in the case of Orbitz, as Orbitz does not present the querier with any information until all the possible flights are available, while aggregating over the delay of work may be more appropriate in the case of Kayak, as Kayak presents the querier with flight options as they are found. Also, often the aggregate flow of work is used as a surrogate measure for the aggregate flow of jobs as it tends to be more mathematically tractable. In particular, for the trade-off problem that we consider here, the problem is NP-hard if we were to consider the performance measure of the aggregate weighted flow of jobs, instead of the aggregate weighted flow of work. The hardness follows immediately from the well known fact that minimizing the weighted flow time of jobs on a unit speed processor is NP-hard [10], or from the fact that minimizing total weighted flow, without release times, subject to an energy budget is NP-hard [12].

There is a fair number of papers that study approximately computing optimal trade-off schedules, both offline and online. [12] also gives PTAS's for minimizing total flow without release times subject to an energy budget in both the continuous and discrete speed settings. [2, 6, 11, 4, 3, 5, 8, 9] consider online algorithms for optimal total flow and energy, [4, 5] consider online algorithms for fractional flow and energy. For a survey on energy-efficient algorithms, see [1].

3 Model & Preliminaries

We consider the problem of scheduling a set $\mathcal{J} \coloneqq \{1, 2, \ldots, n\}$ of n jobs on a single processor featuring k different speeds $0 < s_1 < s_2 < \ldots < s_k$. The power consumption of the processor while running at speed s_i is $P_i \geq 0$. We use $\mathcal{S} \coloneqq \{s_1, \ldots, s_k\}$ to denote the set of speeds and $\mathcal{P} \coloneqq \{P_1, \ldots, P_k\}$ to denote the set of powers. While running at speed s_i, the processor performs s_i units of work per time unit and consumes energy at a rate of P_i.

Each job $j \in \mathcal{J}$ has a release time r_j, a processing volume (or work) p_j, and a weight w_j. Moreover, we denote the value $d_j \coloneqq \frac{w_j}{p_j}$ as the density of job j. All densities are distinct;

details about this assumption are left for the full version. For each time t, a schedule S must decide which job to process at what speed. We allow preemption, that is, a job may be suspended at any point in time and resumed later on. We model a schedule S by a speed function $V\colon \mathbb{R}_{\geq 0} \to \mathcal{S}$ and a scheduling policy $J\colon \mathbb{R}_{\geq 0} \to \mathcal{J}$. Here, $V(t)$ denotes the speed at time t, and $J(t)$ the job that is scheduled at time t. Jobs can be processed only after they have been released. For job j let $I_j = J^{-1}(j) \cap [r_j, \infty)$ be the set of times during which it is processed. A feasible schedule must finish the work of all jobs. That is, the inequality $\int_{I_j} S(t)\,\mathrm{d}t \geq p_j$ must hold for all jobs j.

We measure the quality of a given schedule S by means of its energy consumption and its fractional flow. The speed function V induces a power function $P\colon \mathbb{R}_{\geq 0} \to \mathcal{P}$, such that $P(t)$ is the power consumed at time t. The energy consumption of schedule S is $E(S) \coloneqq \int_0^\infty P(t)\,\mathrm{d}t$. The flow time (also called response time) of a job j is the difference between its completion time and release time. If F_j denotes the flow time of job j, the weighted flow of schedule S is $\sum_{j\in J} w_j F_j$. However, we are interested in the fractional flow, which takes into account that different parts of a job j finish at different times. More formally, if $p_j(t)$ denotes the work of job j that is processed at time t (i.e., $p_j(t) = V(t)$ if $J(t) = j$, and $p_j(t) = 0$ otherwise), the fractional flow time of job j is $\widetilde{F}_j \coloneqq \int_{r_j}^\infty (t - r_j)\frac{p_j(t)}{p_j}\,\mathrm{d}t$. The fractional weighted flow of schedule S is $\widetilde{F}(S) \coloneqq \sum_{j\in\mathcal{J}} w_j \widetilde{F}_j$. The objective function is $E(S) + \widetilde{F}(S)$. Our goal is to find a feasible schedule that minimizes this objective.

We define $s_0 \coloneqq 0$, $P_0 \coloneqq 0$, $s_{k+1} \coloneqq s_k$, and $P_{k+1} \coloneqq \infty$ to simplify notation. Note that, without loss of generality, we can assume $\frac{P_i - P_{i-1}}{s_i - s_{i-1}} < \frac{P_{i+1} - P_i}{s_{i+1} - s_i}$; Otherwise, any schedule using s_i could be improved by linearly interpolating the speeds s_{i-1} and s_{i+1}.

4 Overview

In this section we give an overview of our algorithm design and analysis. We start by considering a natural linear programming formulation of the problem. We then consider the dual linear program. Using complementary slackness we find necessary and sufficient conditions for a candidate schedule to be optimal. Reminiscent of the approach used in the case of continuous speeds in [7], we then interpret these conditions in the following geometric manner. Each job j is associated with a linear function $D_j^{\alpha_j}(t)$, which we call *dual line*. This dual line has a slope of $-d_j$ and passes through point (r_j, α_j), for some $\alpha_j > 0$. Here t is time, α_j is the dual variable associated with the primal constraint that all the work from job j must be completed, r_j is the release time of job j, and d_j is the density of job j. Given such an α_j for each job j, one can obtain an associated schedule as follows: At every time t, the job j being processed is the one whose dual line is the highest at that time, and the speed of the processor depends solely on the height of this dual line at that time.

The left picture in Figure 1 shows the dual lines for four different jobs on a processor with three modes. The horizontal axis is time. The two horizontal dashed lines labeled by C_2 and C_3 represent the heights where the speed will transition between the lowest speed mode and the middle speed mode, and the middle speed mode and the highest speed mode, respectively (these lines only depend on the speeds and powers of the modes and not on the jobs). The right picture in Figure 1 shows the associated schedule.

By complementary slackness, a schedule corresponding to a collection of α_j's is optimal if and only if it processes exactly p_j units of work for each job j. Thus we can reduce finding an optimal schedule to finding values for these dual variables with this property.

Our algorithm is a primal-dual algorithm that raises the dual α_j variables in an organized way. We iteratively consider the jobs by decreasing density. In iteration i, we construct the

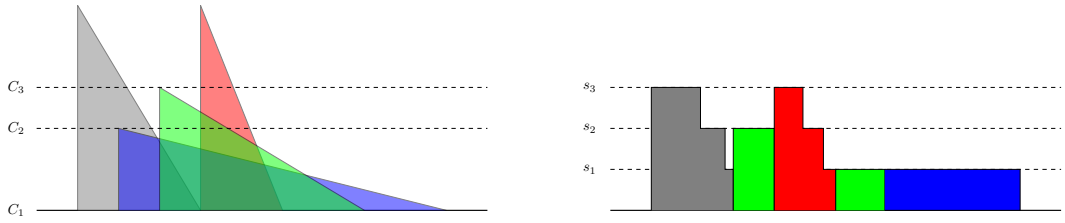


Figure 1 The dual lines for a 4-job instance, and the associated schedule.

optimal schedule S_i for the i most dense jobs from the optimal schedule S_{i-1} for the $i-1$ most dense jobs. We raise the new dual variable α_i from 0 until the associated schedule processes p_i units of work from job i. At some point raising the dual variable α_i may cause the dual line for i to "affect" the dual line for a previous job j in the sense that α_j must be raised as α_i is raised in order to maintain the invariant that the right amount of work is processed on job j. Intuitively one might think of "affection" as meaning that the dual lines intersect (this is not strictly correct, but might be a useful initial geometric interpretation to gain intuition). More generally this affection relation can be transitive in the sense that raising the dual variable α_j may in turn affect another job, etc.

The algorithm maintains an affection tree rooted at i that describes the affection relationship between jobs, and maintains for each edge in the tree a variable describing the relative rates that the two incident jobs must be raised in order to maintain the invariant that the proper amount of work is processed for each job. Thus this tree describes the rates that the dual variables of old jobs must be raised as the new dual variable α_i is raised at a unit rate.

In order to discretize the raising of the dual lines, we define four types of events that cause a modification to the affection tree:

- a pair of jobs either begin or cease to affect each other,
- a job either starts using a new mode or stops using some mode,
- the rightmost point on a dual line crosses the release time of another job, or
- enough work is processed on the new job i.

During an iteration, the algorithm repeatedly computes when the next such event will occur, raises the dual lines until this event, and then computes the new affection tree. Iteration i completes when job i has processed enough work. Its correctness follows from the facts that (i) the affection graph is a tree, (ii) this affection tree is correctly computed, (iii) the four aforementioned events are exactly the ones that change the affection tree, and (iv) the next such event is correctly computed by the algorithm. We bound the running time by bounding the number of events that can occur, the time required to calculate the next event of each type, and the time required to recompute the affection tree after each event.

5 Structural Properties via Primal-Dual Formulation

In the following, we give an integer linear programming (ILP) description of our problem. To this end, let us assume that time is divided into discrete time slots such that, in each time slot, the processor runs at constant speed and processes at most one job. Note that these time slots may be arbitrarily small, yielding an ILP with many variables and, thus, rendering a direct solution approach less attractive. However, we are actually not interested in solving this ILP directly. Instead, we merely strive to use it and its dual in order to obtain some simple structural properties of an optimal schedule.

$$\begin{aligned}\min \quad & \sum_{j\in\mathcal{J}}\sum_{t=r_j}^{T}\sum_{i=1}^{k} x_{jti}\big(P_i + s_i d_j(t - r_j + 1/2)\big) \\ \text{s.t.} \quad & \sum_{t=r_j}^{T}\sum_{i=1}^{k} x_{jti}\cdot s_i \geq p_j && \forall j \\ & \sum_{j\in\mathcal{J}}\sum_{i=1}^{k} x_{jti} \leq 1 && \forall t \\ & x_{jti} \in \{0,1\} && \forall j,t,i\end{aligned}$$

(a) ILP formulation of our scheduling problem.

$$\begin{aligned}\max \quad & \sum_{j\in\mathcal{J}} p_j\alpha_j - \sum_{t=1}^{T}\beta_t \\ \text{s.t.} \quad & \beta_t \geq \alpha_j s_i - P_i \\ & \qquad -s_i d_j(t - r_j + 1/2) \\ & \qquad \forall j,t,i : t \geq r_j \\ & \alpha_j \geq 0 \quad \forall j \\ & \beta_t \geq 0 \quad \forall t\end{aligned}$$

(b) Dual program of the ILP's relaxation.

Figure 2

ILP & Dual Program. Let the indicator variable x_{jti} denote whether job j is processed in slot t at speed s_i. Moreover, let T be some upper bound on the total number of time slots. This allows us to model our scheduling problem via the ILP given in Figure 2a. The first set of constraints ensures that all jobs are completed, while the second set of constraints ensures that the processor runs at constant speed and processes at most one job in each time slot.

In order to use properties of duality, we consider the relaxation of the above ILP. It can easily be shown that any optimal schedule will always use highest density first as its scheduling policy, and therefore there is no advantage to scheduling partial jobs in any time slot. It follows that by considering small enough time slots, the value of an optimal solution to the LP will be no less than the value of the optimal solution to the ILP. After considering this relaxation and taking the dual, we get the dual program shown in Figure 2b.

The complementary slackness conditions of our primal-dual program are

$$\alpha_j > 0 \quad \Rightarrow \quad \sum_{t=r_j}^{T}\sum_{i=1}^{k} x_{jti}\cdot s_i = p_j, \tag{1}$$

$$\beta_t > 0 \quad \Rightarrow \quad \sum_{j\in\mathcal{J}}\sum_{i=1}^{k} x_{jti} = 1, \tag{2}$$

$$x_{jti} > 0 \quad \Rightarrow \quad \beta_t = \alpha_j s_i - P_i - s_i d_j(t - r_j + 1/2) \quad . \tag{3}$$

By complementary slackness, any pair of feasible primal-dual solutions that fulfills these conditions is optimal. We will use this in the following to find a simple way to characterize optimal schedules.

A simple but important observation is that we can write the last complementary slackness condition as $\beta_t = s_i\big(\alpha_j - d_j(t - r_j + \frac{1}{2})\big) - P_i$. Using the complementary slackness conditions, the function $t \mapsto \alpha_j - d_j(t - r_j)$ can be used to characterize optimal schedules. The following definitions capture a parametrized version of these job-dependent functions and state how they imply a corresponding (not necessarily feasible) schedule.

▶ **Definition 1** (Dual Lines and Upper Envelope)**.** For a value $a \geq 0$ and a job j we denote the linear function $D_j^a\colon [r_j,\infty) \to \mathbb{R}, t \mapsto a - d_j(t - r_j)$ as the *dual line* of j with offset a.

Given a job set $H \subseteq \mathcal{J}$ and corresponding dual lines $D_j^{a_j}$, we define the *upper envelope* of H by the upper envelope of its dual lines. That is, the upper envelope of H is a function

$\mathrm{UE}_H\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}, t \mapsto \max_{j\in H}\big(D_j^{a_j}(t), 0\big)$. We omit the job set from the index if it is clear from the context.

For technical reasons, we will have to consider the discontinuities in the upper envelope separately.

▶ **Definition 2** (Left Upper Envelope and Discontinuity)**.** Given a job set $H \subseteq \mathcal{J}$ and upper envelope of H, UE_H, we define the *left upper envelope* at a point t as the limit of UE_H as we approach t from the left. That is, the left upper envelope of H is a function $\mathrm{LUE}_H\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}, t \mapsto \lim_{t'\to t^-} \mathrm{UE}_H(t')$. Note that an equivalent definition of the left upper envelope is $\mathrm{LUE}_H(t) = \max_{j\in H: r_j < t}\big(D_j^{a_j}(t), 0\big)$.

We say that a point t is a *discontinuity* if UE has a discontinuity at t. Note that this implies that $\mathrm{UE}(t) \neq \mathrm{LUE}(t)$.

For the following definition, let us denote $C_i \coloneqq \frac{P_i - P_{i-1}}{s_i - s_{i-1}}$ for $i \in [k+1]$ as the i-th *speed threshold*. We use it to define the speeds at which jobs are to be scheduled. It will also be useful to define $\hat{C}(x) = \min_{i\in[k+1]}\{\, C_i \mid C_i > x \,\}$ and $\check{C}(x) = \max_{i\in[k+1]}\{\, C_i \mid C_i \leq x \,\}$.

▶ **Definition 3** (Line Schedule)**.** Consider dual lines $D_j^{a_j}$ for all jobs. The corresponding *line schedule* schedules job j in all intervals $I \subseteq [r_j, \infty)$ of maximal length in which j's dual line is on the upper envelope of all jobs (i.e., $\forall t \in I\colon D_j^{a_j}(t) = \mathrm{UE}(t)$). The speed of a job j scheduled at time t is s_i, with i such that $C_i = \check{C}(D_j^{a_j}(t))$.

See Figure 1 for an example of a line schedule. Together with the complementary slackness conditions, we can now easily characterize optimal line schedules.

▶ **Lemma 4.** *Consider dual lines $D_j^{a_j}$ for all jobs. The corresponding line schedule is optimal with respect to fractional weighted flow plus energy if it schedules exactly p_j units of work for each job j.*

Proof. Consider the solution x to the ILP induced by the line schedule. We use the offsets a_j of the dual lines to define the dual variables $\alpha_j \coloneqq a_j + \frac{1}{2}d_j$. For $t \in \mathbb{N}$, set $\beta_t \coloneqq 0$ if no job is scheduled in the t-th slot and $\beta_t \coloneqq s_i D_j^{\alpha_j}(t) - P_i$ if job j is scheduled at speed s_i during slot t. It is easy to check that x, α, and β are feasible and that they satisfy the complementary slackness conditions. Thus, the line schedule must be optimal. ◀

6 Computing an Optimal Schedule

In this section, we describe and analyze the algorithm for computing an optimal schedule. We introduce the necessary notation and provide a formal definition of the algorithm in Subsection 6.1. Then, in Subsection 6.2, we prove the correctness of the algorithm.

6.1 Preliminaries and Formal Algorithm Description

Before formally defining the algorithm, we have to introduce some more notation.

▶ **Definition 5** (Interval Notation)**.** Let $\hat{r}_1, \ldots, \hat{r}_n$ denote the n release times in non-decreasing order. We define Ψ_j as a set of indices with $q \in \Psi_j$ if and only if job j is run between $\hat{r}_q$ and $\hat{r}_{q+1}$ (or after $\hat{r}_n$ for $q = n$). Further, let $x_{\ell,q,j}$ denote the time that the interval corresponding to q begins and $x_{r,q,j}$ denote the time that the interval ends. Let $s_{\ell,q,j}$ denote the speed at which j is running at the left endpoint corresponding to q and $s_{r,q,j}$ denote the speed j is running at the right endpoint. Let $q_{\ell,j}$ be the smallest and $q_{r,j}$ be the largest indices of Ψ_j, i.e., the indices of the first and last execution intervals of j.

Let the indicator variable $y_{r,j}(q)$ denote whether $x_{r,q,j}$ occurs at a release point. Similarly, $y_{\ell,j}(q) = 1$ if $x_{\ell,q,j}$ occurs at r_j, and 0 otherwise. Lastly, $\chi_j(q)$ is 1 if q is not the last interval in which j is run, and 0 otherwise.

We define $\rho_j(q)$ to be the last interval of the uninterrupted block of intervals starting at q, i.e., for all $q' \in \{ q+1, \ldots, \rho_j(q) \}$, we have that $q' \in \Psi_j$ and $x_{r,q'-1,j} = x_{\ell,q',j}$, and either $\rho_j(q) + 1 \notin \Psi_j$ or $x_{r,\rho_j(q),j} \neq x_{\ell,\rho_j(q)+1,j}$.

Within iteration i of the algorithm, τ will represent how much we have raised α_i. We can think of τ as the time parameter for this iteration of the algorithm (not time as described in the original problem description, but time with respect to raising dual-lines). To simplify notation, we do not index variables by the current iteration of the algorithm. In fact, note that every variable in our description of the algorithm may be different at each iteration of the algorithm, e.g., for some job j, $\alpha_j(\tau)$ may be different at the i-th iteration than at the $(i+1)$-st iteration. To further simplify notation, we use D_j^τ to denote the dual line of job j with offset $\alpha_j(\tau)$. Similarly, we use UE^τ to denote the upper envelope of all dual lines D_j^τ for $j \in [i]$ and S_i^τ to denote the corresponding line schedule. As the line schedule changes with τ, so does the set of intervals corresponding to it, therefore we consider variables relating to intervals to be functions of τ as well (e.g., $\Psi_j(\tau)$, $x_{\ell,q,j}(\tau)$, etc.). Prime notation generally refers to the rate of change of a variable with respect to τ, e.g., $\alpha_j'(\tau_0)$ is the rate of change of α_j with respect to τ at τ_0. To lighten notation, we drop τ from variables when its value is clear from the context.

We start by formally defining a relation capturing the idea of jobs affecting each other while being raised.

▶ **Definition 6** (Affection)**.** Consider two different jobs j and j'. We say job j *affects* job j' at time τ if raising (only) the dual line D_j^τ would decrease the processing time of j' in the corresponding line schedule.

We write $j \to j'$ to indicate that j affects j' (and refer to the parameter τ separately, if not clear from the context). Similarly, we write $j \not\to j'$ to state that j does not affect j'.

The affection relation naturally defines a graph on the jobs, which we define below. The following definition assumes that we are in iteration i of the algorithm.

▶ **Definition 7** (Affection Tree)**.** Let $G_i(\tau)$ be the directed graph induced by the affection relation on jobs $1, \ldots, i$. Then the *affection tree* is an undirected graph $A_i(\tau) = (V_i(\tau), E_i(\tau))$ where $j \in V_i(\tau)$ if and and only if j is reachable from i in $G_i(\tau)$, and for $j_1, j_2 \in V_i(\tau)$ we have $(j_1, j_2) \in E_i(\tau)$ if and only if $j_1 \to j_2$ or $j_2 \to j_1$.

Lemma 9 states that the affection tree is indeed a tree. We will assume that $A_i(\tau)$ is rooted at i and use the notation $(j, j') \in A_i(\tau)$ to indicate that j' is a child of j.

Given this notation, we now define four different types of events which intuitively represent the situations in which we must change the rate at which we are raising the dual line. We assume that from τ until an event we raise each dual line at a constant rate. More formally, we fix τ and for $j \in [i]$ and $u \geq \tau$ let $\alpha_j(u) = \alpha_j(\tau) + (u - \tau)\alpha_j'(\tau)$.

▶ **Definition 8** (Event)**.** For $\tau_0 > \tau$, we say that an *event occurs at* τ_0 if there exists $\epsilon > 0$ such that at least one of the following holds for all $u \in (\tau, \tau_0)$ and $v \in (\tau_0, \tau_0 + \epsilon)$:

- The affection tree changes, i.e., $A_i(u) \neq A_i(v)$. This is called an *affection change event.*
- The speed at the border of some interval of some job changes. That is, there exists $j \in [i]$ and $q \in \Psi_j(\tau)$ such that either $s_{\ell,q,j}(u) \neq s_{\ell,q,j}(v)$ or $s_{r,q,j}(u) \neq s_{r,q,j}(v)$. This is called a *speed change event.*

- The last interval in which job i is run changes from ending before the release time of some other job to ending at the release time of that job. That is, there exists a $j \in [i-1]$ and a $q \in \Psi_i(\tau)$ such that $x_{r,q,i}(u) < r_j$ and $x_{r,q,i}(v) = r_j$. This is called a *simple rate change event*.
- Job i completes enough work, i.e., $p_i(u) < p_i < p_i(v)$. This is called a *job completion event*.

A formal description of the algorithm can be found in Algorithm 1.

```
1  for each job i from 1 to n:
2      while p_i(τ) < p_i:                    {job i not yet fully processed in current schedule}
3          for each job j ∈ A_i(τ):
4              calculate δ_{j,i}(τ)           {see Equation (5)}
5          let Δτ be the smallest Δτ returned by any of the subroutines below:
6              (a) JobCompletion(S(τ), i, [α'_1, α'_2, ..., α'_i])      {time to next job completion}
7              (b) AffectionChange(S(τ), A_i(τ), [α'_1, α'_2, ..., α'_i])   {time to next affection change}
8              (c) SpeedChange(S(τ), [α'_1, α'_2, ..., α'_i])           {time to next speed change}
9              (d) RateChange(S(τ), i, [α'_1, α'_2, ..., α'_i])         {time to next rate change}
10         for each job j ∈ A_i(τ):
11             raise α_j by Δτ · δ_{j,i}
12         set τ = τ + Δτ
13         update A_i(τ) if needed            {only if Case (b) returns the smallest Δτ}
```

Algorithm 1 The algorithm for computing an optimal schedule.

6.2 Correctness of the Algorithm

In this subsection we focus on proving the correctness of the algorithm. Throughout this subsection, we assume that the iteration and value of τ are fixed. The following lemma states that A_i is indeed a tree. This structure will allow us to easily compute how fast to raise the different dual lines of jobs in A_i (as long as the connected component does not change).

▶ **Lemma 9.** *Let A_i be the (affection) graph of Definition 7. Then A_i is a tree, and if we root A_i at i, then for any parent and child pair $(\iota_j, j) \in G$ there holds that $d_{\iota_j} < d_j$.*

Recall that we have to raise the dual lines such that the total work done for any job $j \in [i-1]$ is preserved. To calculate the work processed for j in an interval, we must take into account the different speeds at which j is run in that interval. Note that the intersection of j's dual line with the i-th speed threshold C_i occurs at $t = \frac{\alpha_j - C_i}{d_j} + r_j$. Therefore, the work done by a job $j \in [i]$ is given by

$$\begin{aligned} p_j = & \sum_{q \in \Psi_j} s_{\ell,q,j} \left(\frac{\alpha_j - \check{C}(D_j^\tau(x_{\ell,q,j}))}{d_j} + r_j - x_{\ell,q,j} \right) \\ & + \sum_{k : s_{\ell,q,j} > s_k > s_{r,q,j}} s_k \left(\frac{\alpha_j - C_k}{d_j} + r_j - \left(\frac{\alpha_j - C_{k+1}}{d_j} + r_j \right) \right) \\ & + s_{r,q,j} \left(x_{r,q,j} - \left(\frac{\alpha_j - \hat{C}(D_j^\tau(x_{r,q,j}))}{d_j} + r_j \right) \right). \end{aligned}$$

It follows that the change in the work of job j with respect to τ is

$$p'_j = \sum_{q \in \Psi_j} \left[s_{\ell,q,j} \left(\frac{\alpha'_j}{d_j} - x'_{\ell,q,j} \right) + s_{r,q,j} \left(x'_{r,q,j} - \frac{\alpha'_j}{d_j} \right) \right]. \tag{4}$$

For some child j' of j in A_i, let $q_{j,j'}$ be the index of the interval of Ψ_j that begins with the completion of j'. Recall that D_i^τ is raised at a rate of 1 with respect to τ, and for a parent and child (ι_j, j) in the affection tree, the rate of change for α_j with respect to α_{ι_j} used by the algorithm is:

$$\delta_{j,\iota_j} := \Bigg(1 + y_{\ell,j}(q_{\ell,j})\frac{d_j - d_{\iota_j}}{d_j}\frac{s_{\ell,q_{\ell,j},j} - s_{r,\rho_j(q_{\ell,j}),j}}{s_{r,q_{r,j},j}} + \sum_{(j,j')\in A_i}\left((1-\delta_{j',j})\frac{d_j - d_{\iota_j}}{d_{j'} - d_j}\frac{s_{\ell,q_{j,j'},j}}{s_{r,q_{r,j},j}} + \frac{d_j - d_{\iota_j}}{d_j}\frac{s_{\ell,q_{j,j'},j} - s_{r,\rho(q_{j,j'}),j}}{s_{r,q_{r,j},j}}\right)\Bigg)^{-1}. \tag{5}$$

Lemma 12 states that these rates are work-preserving for all jobs $j \in [i-1]$. Note that the algorithm actually uses $\delta_{j,i}$ which we can compute by taking the product of the $\delta_{k,k'}$ over all edges (k,k') on the path from j to i. Similarly we can compute $\delta_{j,j'}$ for all $j, j' \in A_i$.

▶ **Observation 10.** *Since, by Lemma 9, parents in the affection tree are always of lower-density than their children, and since dual lines are monotonically decreasing, we have that $\delta_{\iota_j,j} \leq 1$. Therefore, intersection points on the upper envelope can never move towards the right as τ gets increased.*

The following lemma states how fast the borders of the various intervals change with respect to the change in τ.

▶ **Lemma 11.** *Consider any job $j \in A_i$ whose dual line gets raised at a rate of $\delta_{j,i}$.*

(a) *For an interval $q \in \Psi_j$, if $y_{\ell,j}(q) = 1$, then $x'_{\ell,q,j} = 0$.*

(b) *For an interval $q \in \Psi_j$, if $\chi_j(q) = 1$, then $x'_{r,q,j} = 0$.*

(c) *Let (j,j') be an edge in the affection tree and let q_j and $q_{j'}$ denote the corresponding intervals for j and j'. Then, $x'_{\ell,q_j,j} = x'_{r,q_{j'},j'} = -\frac{\alpha'_j - \alpha'_{j'}}{d_{j'} - d_j}$. Note that this captures the case $q \in \Psi_{j'}$ with $\chi_{j'}(q) = 0$ and $j' \neq i$.*

(d) *For an interval $q \in \Psi_i$, if $\chi_i(q) = 0$, then $x'_{r,q,i} = 0$ or $x'_{r,q,i} = 1/d_i$.*

Equation (4) defines a system of differential equations. In the following, we first show how to compute a work-preserving solution for this system (in which $p'_j = 0$ for all $j \in [i-1]$) if $\alpha'_i = 1$, and then show that there is only a polynomial number of events and that the corresponding τ values can be easily computed.

▶ **Lemma 12.** *For a parent and child $(\iota_j, j) \in A_i$, set $\alpha'_j = \delta_{j,\iota_j}\alpha'_{\iota_j}$, and for $j' \notin A_i$ set $\alpha_{j'} = 0$. Then $p'_j = 0$ for $j \in [i-1]$.*

Although it is simple to identify the next occurrence of job completion, speed change, or simple rate change events, it is more involved to identify the next affection change event. Therefore, we provide the following lemma to account for this case.

▶ **Lemma 13.** *An affection change event occurs at time τ_0 if and only if at least one of the following occurs.*

(a) *An intersection point t between a parent and child $(j,j') \in A_i$ becomes equal to r_j. That is, at $\tau_0 > \tau$ such that $D_j^{\tau_0}(r_j) = D_{j'}^{\tau_0}(r_j) = \mathrm{UE}^{\tau_0}(r_j)$.*

(b) *Two intersection points t_1 and t_2 on the upper envelope become equal. That is, for $(j_1,j_2) \in A_i$ and $(j_2,j_3) \in A_i$, at $\tau_0 > \tau$ such that there is a t with $D_{j_1}^{\tau_0}(t) = D_{j_2}^{\tau_0}(t) = D_{j_3}^{\tau_0}(t) = \mathrm{UE}^{\tau_0}(t)$.*

(c) *An intersection point between j and j' meets the (left) upper envelope at the right endpoint of an interval in which j' was being run. Furthermore, there exists $\epsilon > 0$ so that for all $\tau \in (\tau_0 - \epsilon, \tau_0)$, j' was not in the affection tree.*

6.2.1 The Subroutines

Recall that there are four types of events that cause the algorithm to recalculate the rates at which it is raising the dual lines. In Lemma 13 we gave necessary and sufficient conditions for affection change events to occur. The conditions for the remaining event types to occur follow easily from Lemma 11 and Observation 10. Given the rates at which the algorithm is raising the dual lines, we can then easily calculate the time until each of these events will occur next. The subroutines describing these calculations are left for the full version.

6.2.2 Completing the Correctness Proof

We are now ready to prove the correctness of the algorithm. Note that we handle termination in Theorem 15, where we prove a polynomial running time for our algorithm.

▶ **Theorem 14.** *Assuming that Algorithm 1 terminates, it computes an optimal schedule.*

Proof. The algorithm outputs a line schedule S, so by Lemma 4, S is optimal if for all jobs j the schedule does exactly p_j work on j. We now show that this is indeed the case.

For a fixed iteration i, we argue that a change in the rate at which work is increasing for j (i.e., a change in p'_j) may occur only when an event occurs. This follows from Equation (4), since the rate only changes when there is a change in the rate at which the endpoints of intervals move, when there is a change in the speed levels employed in each interval, or when there is an affection change (and hence a change in the intervals of a job or a change in α'_j). These are exactly the events we have defined. It can be shown that the algorithm recalculates the rates at any event (proofs deferred to the full version), and by Lemma 12 it calculates the correct rates such that $p'_j(\tau) = 0$ for $j \in [i-1]$ and for every τ until some τ_0 such that $p_i(\tau_0) = p_i$, which the algorithm calculates correctly (proof also deferred to the full version). Thus we get the invariant that after iteration i we have a line schedule for the first i jobs that does p_j work for every job $j \in [i]$. The theorem follows. ◀

7 The Running Time

The purpose of this section is to prove the following theorem.

▶ **Theorem 15.** *Algorithm 1 takes $O(n^4k)$ time.*

We do this by upper bounding the number of events that can occur. This is relatively straightforward for job completion, simple rate change, and speed change events, which can occur $O(n)$, $O(n^2)$, and $O(n^2k)$ times, respectively. However, bounding the number of times an affection change event can occur is more involved: One can show that whenever an edge is removed from the affection tree, there exists an edge which will never again be in the affection tree. This implies that the total number of affection change events is upper bounded by $O(n^2)$ as well. It can be shown that the next event can always be calculated in $O(n^2)$ time, and that the affection tree can be updated in $O(n)$ time after each affection change event. By combining these results it follows that our algorithm has a running time of $O(n^4k)$.

Due to space constraints, the missing proofs in the statements above are left for the full version.

References

1 Susanne Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

2 Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4), 2007.

3 Lachlan L. H. Andrew, Adam Wierman, and Ao Tang. Optimal speed scaling under arbitrary power functions. *SIGMETRICS Performance Evaluation Review*, 37(2):39–41, 2009.

4 Nikhil Bansal, Ho-Leung Chan, Tak Wah Lam, and Lap-Kei Lee. Scheduling for speed bounded processors. In *Proceedings of the 35th International Conference on Automata, Languages, and Programming (ICALP)*, pages 409–420, 2008.

5 Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. *ACM Transactions on Algorithms*, 9(2), 2013.

6 Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4):1294–1308, 2009.

7 Neal Barcelo, Daniel Cole, Dimitrios Letsios, Michael Nugent, and Kirk Pruhs. Optimal energy trade-off schedules. *Sustainable Computing: Informatics and Systems*, 3:207–217, 2013.

8 Sze-Hang Chan, Tak Wah Lam, and Lap-Kei Lee. Non-clairvoyant speed scaling for weighted flow time. In *Proceedings of the 18th annual European Symposium on Algorithms (ESA), Part I*, pages 23–35, 2010.

9 Nikhil R. Devanur and Zhiyi Huang. Primal dual gives almost optimal energy efficient online algorithms. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1123–1140, 2014.

10 Jacques Labetoulle, Eugene L. Lawler, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In Pulleyblank H. R., editor, *Progress in combinatorial optimization*, pages 245–261. Academic Press, 1984.

11 Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *Proceedings of the 16th annual European Symposium on Algorithms (ESA)*, pages 647–659, 2008.

12 Nicole Megow and José Verschae. Dual techniques for scheduling on a machine with varying speed. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming (ICALP) - Volume Part I*, pages 745–756, 2013.

13 Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3), 2008.

Weighted Coloring in Trees*

Julio Araujo[1,2,3], Nicolas Nisse[2,3], and Stéphane Pérennes[3]

1 **ParGO, Universidade Federal do Ceará, Fortaleza, Brazil**
2 **Inria, France**
3 **Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, Sophia Antipolis, France**

Abstract

A proper coloring of a graph is a partition of its vertex set into stable sets, where each part corresponds to a *color*. For a vertex-weighted graph, the *weight of a color* is the maximum weight of its vertices. The *weight of a coloring* is the sum of the weights of its colors. Guan and Zhu defined the *weighted chromatic number* of a vertex-weighted graph G as the smallest weight of a proper coloring of G (1997). If vertices of a graph have weight 1, its weighted chromatic number coincides with its chromatic number. Thus, the problem of computing the weighted chromatic number, a.k.a. Max Coloring Problem, is NP-hard in general graphs. It remains NP-hard in some graph classes as bipartite graphs. Approximation algorithms have been designed in several graph classes, in particular, there exists a PTAS for trees. Surprisingly, the time-complexity of computing this parameter in trees is still open.

The Exponential Time Hypothesis (ETH) states that 3-SAT cannot be solved in sub-exponential time. We show that, assuming ETH, the best algorithm to compute the weighted chromatic number of n-node trees has time-complexity $n^{\Theta(\log n)}$. Our result mainly relies on proving that, when computing an optimal proper weighted coloring of a graph G, it is hard to combine colorings of its connected components.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases Weighted Coloring, Max Coloring, Exponential Time Hypothesis, 3-SAT

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.75

1 Introduction

Given a loop-less graph $G = (V, E)$, a *(proper) k-coloring* of G is a surjective function $c : V \to \{1, \ldots, k\}$ that assigns to each vertex $v \in V$ a *color* $c(v) \in \{1, \ldots, k\}$, such that, for any $\{u, v\} \in E$, $c(u) \neq c(v)$. Equivalently, a k-coloring of G is a partition $c = (S_1, \ldots, S_k)$ of V such that, for any $1 \leq i \leq k$, S_i is a non-empty independent set of vertices that have the same color i. One of the most studied problems in Graph Theory consists in minimizing the number of colors of a proper coloring of a graph. Namely, Graph Coloring aims at computing the *chromatic number* of a graph G, denoted by $\chi(G)$, which is the minimum k for which G has a k-coloring. This is one of the Karp's NP-hard problems [8].

In [6], Guan and Zhu generalized Graph Coloring to vertex-weighted graphs. A *(vertex) weighted graph* (G, w) consists of a loop-less graph $G = (V, E)$ and a weight function $w : V \to \mathbb{R}_+$ over the vertices of G. Given a k-coloring $c = (S_1, \ldots, S_k)$ of a weighted graph (G, w), the *weight of color* i $(1 \leq i \leq k)$ is defined by $w(i) = \max_{v \in S_i} w(v)$. The *weight of coloring* c is $w(c) = \sum_{i=1}^{k} w(i)$. The *weighted chromatic number* of (G, w), denoted by

* This work was partly funded by the ANR projects AGAPE and GRATEL, and promoted by the Inria/FUNCAP project ALERTE and the Inria associate-team AlDyNet and CNPq–Brazil (contract PDE 202049/2012-4).



© Julio Araujo, Nicolas Nisse, and Stéphane Pérennes;
licensed under Creative Commons License CC-BY





$\chi_w(G)$, is the minimum weight of a proper coloring of (G, w). The Weighted Coloring Problem (also known as Max-coloring [15, 12, 13, 14, 11]) takes a weighted graph (G, w) as input and asks whether $\chi_w(G)$ is bounded [6].

Observe that if the weight of each of the vertices of a graph (G, w) is equal to one, then the weight of a coloring is the number of its colors and thus, $\chi_w(G) = \chi(G)$. Therefore, Weighted Coloring generalizes Graph Coloring to weighted graphs, and, as a consequence, this problem is NP-hard in general graphs. Moreover, Weighted Coloring has been shown NP-hard in bipartite graphs [3], where Graph Coloring is trivial. In the last years, the Weighted Coloring Problem has been addressed several times, however the complexity of this problem is surprisingly still unknown in the class of trees.

Here, we show that, if 3-SAT cannot be solved in sub-exponential time (Exponential Time Hypothesis), then Weighted Coloring in trees is not in P.

Related work. Weighted Coloring has been shown to be NP-hard in the classes of split graphs, interval graphs, triangle-free planar graphs with bounded degree, and bipartite graphs [3, 14, 2, 5, 15]. On the other hand, the weighted chromatic number of cographs and of some subclasses of bipartite graphs can be found in polynomial-time [3, 2]. Constant-factor approximation algorithms have been designed for various graph classes such as interval graphs, perfect graphs, etc. [14, 11, 12, 13, 4]. In particular, it is known that Weighted Coloring can be approximated by a factor $\frac{8}{7}$ in bipartite graphs and cannot be approximated by a factor $\frac{8}{7} - \epsilon$ for any $\epsilon > 0$ in this graph class unless $P = NP$ [13].

Guan and Zhu showed that, given a fixed parameter $r \in \mathbb{N}$, the minimum weight of a coloring using at most r colors can be computed in polynomial-time[1] in the class of bounded treewidth graphs (a.k.a. partial k-trees) [6]. They left open the question of the time-complexity of the Weighted Coloring Problem in this class (partial k-trees) and, in particular, in trees. In [13], a sub-exponential algorithm and a polynomial-time approximation scheme to compute the weighted chromatic number of trees are presented. Later on, Escoffier et al. proposed a polynomial-time approximation scheme to compute the weighted chromatic number of bounded treewidth graphs [5]. Kavitha and Mestre recently presented polynomial-time algorithms for subclasses of trees [9]. They show that computing the weighted chromatic number can be done in linear time in the class of trees where nodes with degree at least three induce a stable set [9].

In the last years, many studies have been done on the Weighted Coloring Problem, however the complexity of this problem was still unknown on trees. Indeed, Weighted Coloring in trees has some intriguing properties: on the one hand, a reduction to another NP-hard problem was unlikely to exist due to the existence of a sub-exponential algorithm [13] (see also Section 2); on the other hand, all the classical methods to derive polynomial-time algorithms on trees failed [5, 9]. We provide here some explanation for these facts.

Our results. We show that, under the Exponential Time Hypothesis (ETH) (see Section 2), the best algorithm to compute the weighted chromatic number of trees has time-complexity $n^{\Theta(\log n)}$, where n is the number of vertices of the input tree. The existence of an algorithm that solves the Weighted Coloring Problem in time $n^{\Theta(\log n)}$ in bounded treewidth graphs follows easily from previous results. The difficulty is to prove that it is optimal under ETH. For this, we show that computing the weighted chromatic number of an n-node tree is as hard as deciding whether a 3-SAT formula with size $log^2 n$ can be satisfied, where the size of

[1] We emphasize that this algorithm is exponential in r

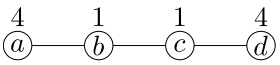


Figure 1 The unique optimal weighted coloring of P_4 uses strictly more than $\chi(P_4)$ colors.

a formula is η if it has η variables and its number of clauses is a polynomial in η. So, our reduction is rather technical, but we hope that it contains ideas that may be used in other contexts. Along the line of our reduction, one will discover another surprising aspect: the difficulty of the problem not only comes from the graph structure, but rather relies on the way weights are structured. This implies that choosing the right color for a node is hard. We indeed use non-binary constraint satisfaction formulae (i.e., constraint satisfaction formulae over positive integers) as main tool. Lastly, our reduction also proves that computing an optimal weighted coloring of a disconnected graph may be hard even if optimal colorings of each of its components can be done in polynomial-time.

Organization of the paper. The remainder of the paper is organized as follows. In Section 2, we formally state the main results of the paper: in Section 2.1, an $n^{\mathcal{O}(\log n)}$-time algorithm is derived from previous works, and in Section 2.2 we prove our main result assuming a technical reduction (Proposition 2). The remaining part of the paper is devoted to the proof of Proposition 2. In Section 3, we give the main ideas of its proof. Finally, in Section 4, we prove a technical result (Proposition 3) which allows us to prove Proposition 2.

2 Preliminaries

2.1 Sub-exponential algorithm

In this section, we show that there exists a sub-exponential algorithm to solve the Weighted Coloring Problem in the class of bounded treewidth graphs (including trees). This is an almost trivial consequence of previous works that mainly relies on the number of colors used by weighted colorings in these graphs.

There exist weighted graphs G for which any optimal weighted coloring uses strictly more than $\chi(G)$ colors: let us consider the 4-node path P_4 with $V(P_4) = \{a, b, c, d\}$, $w(a) = w(d) = 4$ and $w(b) = w(c) = 1$ (see Figure 1). Any coloring of P_4 with $2 = \chi(P_4)$ colors has weight 8, and the optimal coloring $\{\{a, d\}, \{b\}, \{c\}\}$ of P_4 has weight $\chi_w(P_4) = 6$ but uses 3 colors.

Luckily, the number of colors used by optimal weighted colorings can be bounded by $\mathcal{O}(\log n)$ in the class of bounded treewidth graphs with n nodes. Indeed, Guan and Zhu studied the number of colors used by an optimal weighted coloring [6]. More precisely, they proved that the maximum number of colors of an optimal weighted coloring of a weighted graph (G, w) is its first-fit chromatic number $\chi_{FF}(G)$ (a.k.a., *Grundy number*) [6]. This is tight since, for any graph G, there exists a weight function w such that an optimal weighted coloring of (G, w) uses $\chi_{FF}(G)$ colors. On the other hand, for any n-node graph G with tree-width at most k, $\chi_{FF}(G) = \mathcal{O}(k \log n)$ [10]. In particular, this implies that, for any n-node tree, there is an optimal weighted coloring using $\mathcal{O}(\log n)$ colors. Finally, in the class of bounded treewidth graphs and when the number $r \in \mathbb{N}$ of colors is fixed, there is an algorithm (using dynamic programming on the tree-decomposition) that computes the minimum weight of a coloring using at most r colors in time polynomial in $\mathcal{O}(n^r)$ where n is the number of vertices of the input graph [6].

By combining these results, the following proposition is straightforward:

▶ Proposition 1. There exists an algorithm that solves the Weighted Coloring Problem in time $n^{\mathcal{O}(\log n)}$ in the class of bounded treewidth graphs (including trees), where n is the number of vertices of the input graph.

2.2 Main Result

We now formalize our main result. Recall that an instance of the 3-SAT Problem is any Boolean formula $\Phi(v_1, \ldots, v_\eta)$ over the variables $v_1, \ldots, v_\eta$ in the conjunctive normal form (CNF) where each clause involves three variables. The *size* of Φ is η if it depends on η variables and its number of clauses is polynomial in η. The 3-SAT Problem asks whether there exists a truth assignment to the variables such that $\Phi(v_1, \ldots, v_\eta)$ is true. It is well known that the 3-SAT Problem is NP-complete [1]. A fundamental question is to know whether it can be solved in sub-exponential time. Note that, otherwise, $P \neq NP$.

▶ Conjecture 1. **Exponential Time Hypothesis (ETH)** [7].
3-SAT cannot be solved in time $2^{o(\eta)}$ where η is the size of the instance.

The main part of this paper is devoted to proving the following result.

▶ Proposition 2. For any Boolean formula Φ of size η, there exist a weighted tree (T, w) with $n = 2^{\mathcal{O}(\sqrt{\eta})}$ vertices and $M \in \mathbb{R}$ such that Φ is satisfiable if and only if $\chi_w(T) \leq M$. Moreover, (T, w) and M are computable in time polynomial in n.

Proposition 2 allows us to prove that there is no polynomial-time algorithm to solve the Weighted Coloring Problem in trees, unless ETH fails.

▶ **Theorem 1.** *If ETH is true, then the best algorithm to compute the weighted chromatic number of an n-node tree T has time-complexity $n^{\Theta(\log n)}$.*

Proof. The existence of such an algorithm directly follows from Proposition 1. For purpose of contradiction, let us assume that there exists an algorithm $\mathcal{A}$ that solves the Weighted Coloring Problem in time $n^{o(\log n)}$ in the class of trees, where n is the number of vertices of the input tree. Let Φ be any Boolean formula of size η. By Proposition 2, there exists a weighted tree (T, w) with $n = 2^{\mathcal{O}(\sqrt{\eta})} = 2^{o(\eta)}$ vertices and $M \in \mathbb{R}$ such that Φ is satisfiable if and only if $\chi_w(T) \leq M$. Consider the following algorithm to solve 3-SAT. For any Boolean formula Φ of size η, first compute (T, w) and M in time $2^{o(\eta)}$, then use Algorithm $\mathcal{A}$ to compute $\chi_w(T)$ in time $n^{o(\log n)} = 2^{o(\eta)}$. By definition, Φ is satisfiable if and only if $\chi_w(T) \leq M$. Therefore, the above algorithm solves the 3-SAT Problem in time $2^{o(\eta)}$ where η is the size of the instance, contradicting ETH. ◀

The remaining part of the paper is devoted to the proof of Proposition 2.

3 From boolean variables to integral variables

Proposition 2 establishes a link between the Weighted Coloring Problem and 3-SAT. Informally, to evaluate the time-complexity of the Weighted Coloring Problem, the ideal way would be to reduce any 3-SAT formula Φ to a weighted tree (T, w) such that (1) there is a correspondence between truth assignments of the variables of Φ and the optimal colorings of T, and (2) Φ is satisfiable if and only if $\chi_w(T)$ is at most some pre-defined value M (depending on Φ). To do such a reduction, we would like to proceed as follows: given a boolean formula Φ of size η, we build a weighted tree T such that any truth assignment of Φ for which Φ is satisfied, we have a coloring of T of bounded weight, where the weight

of a color reflects the truth assignment of a variable. Hence, the desired weighted tree T must be such that any optimal coloring of T uses η colors. However, proceeding that way, since the number of colors in an optimal weighted coloring of an n-node tree is at most $\mathcal{O}(\log n)$, T must have at least $n = 2^\eta$ nodes. Hence, a polynomial-time algorithm to solve the Weighted Coloring Problem in T would only lead to an exponential-time algorithm for deciding whether Φ is satisfiable.

3.1 From 3-SAT to INT-SAT

To bypass the above problem, we will use an auxiliary formula. Intuitively, given a 3-SAT formula with η boolean variables, we will translate it into another logical formula with $\sqrt{\eta}$ *integral variables*. Using this new formula, we build a tree with $2^{\sqrt{\eta}}$ nodes, where the weights of the colors in coloring of bounded weight will correspond to the integral values of the variables. Note that our method is close to the *Split and List* method of [16]. More formally,

▶ **Definition 2.** Given a set of $n \times m$ boolean variables $(y^i_j)_{i<n,j<m}$, an *integral assignment* of these variables is a truth assignment such that, for any $0 \leq i < n$, at most one variable y^i_j, $0 \leq j < m$, receives value 1.

A boolean formula Φ with $n \times m$ boolean variables $(y^i_j)_{i<n,j<m}$ is *integrally satisfiable w.r.t.* $(y^i_j)_{i<n,j<m}$ if there is an integral assignment of its variables that satisfies Φ.

The *INT-SAT Problem* takes a formula Φ with variables $(y^i_j)_{i<n,j<m}$ as input and asks whether Φ is integrally satisfiable w.r.t. $(y^i_j)_{i<n,j<m}$.

In what follows, we widely use the fact that there is a one-to-one mapping between any integral assignment of a set of $n \times m$ boolean variables $(y^i_j)_{i<n,j<m}$ and the set of n-tuples $(x_1, \ldots, x_n)$ of integers in $\{0, \ldots, m\}$. Indeed, for any $i < n$, $x_i = j$ if and only if $y^i_j = 1$, and $x_i = 0$ if $y^i_j = 0$ for all $j < m$.

We now show that 3-SAT can be sub-exponentially reduced to INT-SAT. This is an important ingredient of the proof of Proposition 2. We also think this result has its own interest and could be used in other contexts.

▶ **Theorem 3.** *For any instance Φ of 3-SAT with size η, there is a Boolean formula Φ_{int} of size $n = 2^{\mathcal{O}(\sqrt{\eta})}$, with variables $(y^i_j)_{i<\sqrt{\eta},j<2^{\sqrt{\eta}}}$, s.t. Φ is satisfiable if and only if Φ_{int} is integrally satisfiable w.r.t. $(y^i_j)_{i,j}$. Φ_{int} can be computed in time $\mathcal{O}(n)$ and it is a CNF formula where all variables appear positively.*

Proof. Let $\Phi(u_1, \ldots, u_\eta)$ be an instance of 3-SAT of size $\eta = N^2$ (if $\eta \neq N^2$, we can add dummy variables). For any two integers $a < N$ and $b < 2^N$, let $bit(a, b)$ correspond to the a-th bit of the binary representation of b.

Let Φ_{int} be the formula obtained from Φ by replacing each literal u_{iN+j}, $0 \leq i < N$ and $0 \leq j < N$, by $\bigvee_{\{\ell | bit(j,\ell)=1,\ 0\leq\ell<2^N\}} v^i_\ell$. Then, each literal $\bar{u}_{iN+j}$, $0 \leq i < N$ and $0 \leq j < N$ is replaced by $\bigvee_{\{\ell | bit(j,\ell)=0,\ 0\leq\ell<2^N\}} v^i_\ell$. Hence, Φ_{int} has $N \cdot 2^N$ variables

$$(v^1_0, \ldots, v^1_{2^N-1}, v^2_0, \ldots, v^2_{2^N-1}, \ldots, v^N_0, \ldots, v^N_{2^N-1})$$

and $poly(N)$ clauses of size $\mathcal{O}(2^N)$. Because Φ is in CNF, it is also the case for Φ_{int}. Moreover, all variables appear positively in Φ_{int}.

It remains to show that Φ_{int} is integrally satisfiable if and only if Φ is satisfiable.

First, let us assume that Φ is satisfiable. Let $u_1, \ldots, u_\eta$ be a valid assignment of its variables and, for any $0 < i < N$, let x_i be the integer with $(u_{N(i-1)+1}, \ldots, u_{N(i-1)+N})$ as binary representation. Finally, for any $i < N$ and $j < 2^N$, let us define $v^i_j = 1$ if $x_i = j$ and

$v^i_j = 0$ otherwise. By definition of Φ_{int}, $(v^i_j)_{0\leq i<N,\ 0\leq j<2^N}$ is a valid assignment and Φ_{int} is therefore integrally satisfiable.

Conversely, let us assume that Φ_{int} is integrally satisfiable and let $(x_1, \ldots, x_N)$ be N integers representing a valid assignment for it. Let $u_1, \ldots, u_\eta$ be defined such that, for any $0 \leq i < N$, $(u_{N(i-1)+1}, \ldots, u_{N(i-1)+N})$ is the binary representation of x_i. Then, $u_1, \ldots, u_\eta$ is a satisfying assignment for Φ which is satisfiable. ◀

3.2 Proof of Proposition 2

Theorem 3 allows us to reduce any 3-SAT instance Φ of size η into an INT-SAT instance Φ_{int} with size $2^{\mathcal{O}(\sqrt{\eta})}$. The key point is that this reduction allows us to turn the choice of η boolean variables into the choice of $\sqrt{\eta}$ integers in $\{0, \ldots, 2^{\sqrt{\eta}}\}$. Then, in further sections, we build a tree T with $2^{\mathcal{O}(\sqrt{\eta})}$ vertices from the formula Φ_{int}, such that there is a one to one mapping between any optimal weighted coloring of T and the $\sqrt{\eta}$-tuples of integers in $\{0, \ldots, 2^{\sqrt{\eta}}\}$. Finally, our reduction ensures that the value of $\chi_w(T)$ depends on the integral satisfiability of Φ_{int} and therefore, on the satisfiability of Φ. More formally, the next section is devoted to proving the following result:

▶ Proposition 3. For any CNF Boolean formula Φ_{int} of size n where all variables $(y^i_j)_{i,j}$ appear positively, there exist a weighted tree $(T(\Phi_{int}), w)$ with size polynomial in n and $M \in \mathbb{R}$ s.t. Φ_{int} is integrally satisfiable w.r.t. $(y^i_j)_{i,j}$ if and only if $\chi_w(T(\Phi_{int})) \leq M$. The pair $(T(\Phi_{int}), w)$ and M are computable in time polynomial in n.

The proof of Proposition 2 is straightforward from Theorem 3 and Proposition 3.

4 Proof of Proposition 3

This section is devoted to the proof of Proposition 3.

Let us introduce some notations. Let $n \in \mathbb{N}$ and let $m = 2^n$. Let Φ_{int} be a Boolean formula with $n \times m$ variables $\{y^j_i \mid 0 \leq i < n, 0 \leq j < m\}$ and L clauses, where L is polynomial in n. We assume that Φ_{int} is in the Conjunctive Normal Form and that each variable appears positively. Moreover, we may assume that each variable appears at least once. That is, $\Phi_{int} = \bigwedge_{\ell \leq L} Q_\ell$ and, for any $\ell \leq L$, Q_ℓ is the disjunction of $p_\ell \geq 1$ positive variables.

Let $\epsilon > 0$ such that $nm\epsilon = o(\frac{1}{2^{4n}})$ and let

$$M = \sum_{i=0}^{4n+2} \frac{1}{2^i} + n(m-1)\epsilon < 2.$$

Let $w^j_i = 1/2^i + j\epsilon$, for any $0 \leq i \leq 4n+3$ and $0 \leq j \leq m$. Let $\mathcal{W} = \{w^j_i \mid 0 \leq i \leq 4n+3, 0 \leq j \leq m\}$ denote a set of weights. Note that the length of the encoding of these weights is polynomially bounded. For any $0 \leq k \leq 3$, let $W_k = w^0_{4n+k} = 1/2^{4n+k}$. Finally, for any rooted tree T, let $r(T)$ denote its root. A rooted tree S is a *(proper) subtree* of a rooted tree T if there is an edge e of T such that S is the connected component of $T \setminus \{e\}$ that does not contain $r(T)$. We now define various subtrees required to build $(T(\Phi_{int}), w)$.

4.1 Binomial trees

We first define a particular family of *binomial trees* B_i, $0 \leq i \leq 4n+2$. They will be used in the construction of $T(\Phi_{int})$. Their role is to force the color of most of the nodes in any coloring c of $T(\Phi_{int})$ with $w(c) \leq M$.

▶ **Definition 4.** For any $0 \leq i \leq 4n+2$, let B_i be the weighted rooted tree defined recursively as follows (see Figure 2).

- if $i = 0$, then B_0 has a unique node with weight w_0^0;
- otherwise, B_i has a root of weight w_i^0, with the roots of copies of $B_0, B_1, \ldots, B_{i-1}$ as children.

Note that B_i has 2^i nodes and that it just contains nodes of weight w_j^0, for $0 \leq j \leq i \leq 4n+2$. We will use these binomial trees with two main goals in our reduction:

- enforce the number of used colors and the weights of these colors (up to an additive constant $c\epsilon$) in any optimal weighted coloring of the tree we build from the 3-SAT formula;
- forbid the color i to appear in any vertex that is adjacent to a root of a binomial tree B_i.

We get these properties according to the following lemmas:

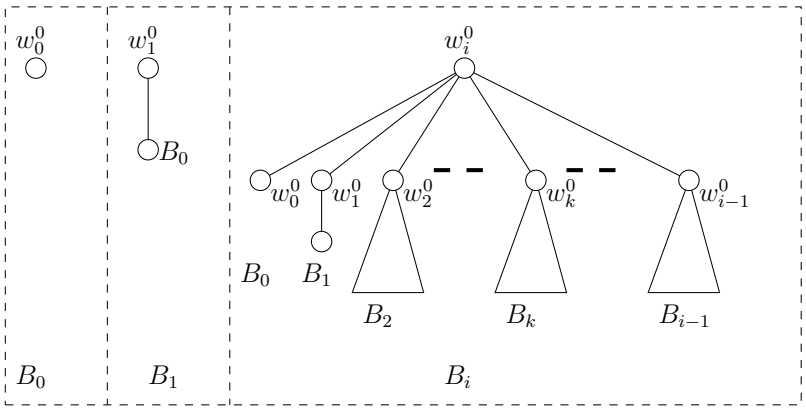


Figure 2 The construction of the binomial tree B_i.

▶ **Lemma 5.** *Let $0 \leq i \leq 4n+2$. Let (T, w) be a weighted tree having B_i as subtree. If there exists a coloring c of (T, w) with $w(c) \leq M$, then, for any $0 \leq k \leq i$:*

1. *all vertices of B_i with weight in w_k^0 receive the same color S_k of c; and*
2. *there exists a unique color class S_k in c of weight in $\{w_k^j \mid 0 \leq j \leq m\}$.*

Proof. The proof is by induction on the index i. In case $i = 0$, we prove both statements of the lemma at once by observing that any two vertices of (T, w) of weight in $\{w_0^{j'} \mid 0 \leq j' \leq m\}$ must belong to the same color class S_0, otherwise we would conclude that $w(c) \geq 2$, that would be a contradiction to the fact that $w(c) \leq M < 2$.

Now, let $0 \leq k \leq i$, observe that the set of nodes of B_i with weight in w_k^0 is an independent set that dominates the nodes of B_i with smaller weights (i.e., in $\{w_{k'}^0 \mid k < k' \leq i\}$).

By induction hypothesis, for any $0 \leq k < i$, the set of nodes of B_i with weight in w_k^0 receive the same color S_k of c and this color class is the unique with weight in $\{w_k^j \mid 0 \leq j \leq m\}$. Then, for any $0 \leq k < i$, the root of B_i cannot be colored S_k, since it has a neighbor with weight w_k^0. Let S_i be the color of the root of B_i in c. We proved that the color S_i cannot contain nodes with weight greater than w_i^{m-1} and that c cannot have another color $S_i' \neq S_i$ with weight in $\{w_i^j \mid 0 \leq j \leq m\}$, because, otherwise the weight of c would be at least $\frac{1}{2^i} + \sum_{k=0}^{i} \frac{1}{2^k} = 2 > M$ in both cases. ◀

▶ **Corollary 6.** *Let (T, w) be a weighted tree having B_{4n+2} as subtree. Let c be any coloring of (T, w) s.t. $w(c) \leq M$. Then, $c = (S_0, \ldots, S_k)$ with $k \geq 4n+2$ and, for any $0 \leq i \leq 4n+2$, S_i is the unique color with weight in $\{w_i^j \mid 0 \leq j \leq m\}$.*

The trees we consider below will always satisfy the requirements of Corollary 6. Therefore, we are able to identify a color by its weight. In other words, in what follows, for any coloring $c = (S_0, \ldots, S_k)$ of weight at most M and for any $i \leq 4n+2$, S_i will be the unique color such that $w(S_i) \in \{w_i^j \mid 0 \leq j \leq m\}$.

Recall that we defined, for any $0 \leq k \leq 3$, $W_k = w_{4n+k}^0 = 1/2^{4n+k}$. By a slight abuse of notation, for any $0 \leq k \leq 3$, we denote $W_k = S_{4n+k}$ as the unique color with weight W_k.

4.2 Auxiliary trees and Variable-trees

This section is mainly devoted to the construction of subtrees that will represent the boolean variables. First, the family of *auxiliary trees* A_i^j, $0 \leq i < 4n, 0 \leq j \leq m$, will be used to introduce some choice when coloring $T(\Phi_{int})$.

▶ **Definition 7.** For any $0 \leq i < 4n, 0 \leq j \leq m$, let A_i^j be the weighted rooted tree defined as follows (see Figure 3). Note that A_i^j consists of 2^{4n} nodes.

1. Let u be its root with weight $w(u) = W_0$, and connect it to a node v (its *s*ubroot) with weight $w(v) = w_i^j$;
2. v is made adjacent to the root of a copy of B_ℓ, for any $0 \leq \ell < i-1$;
3. u is made adjacent to the root of a copy of B_ℓ, for any $0 \leq \ell < 4n$, $\ell \neq i-1$.

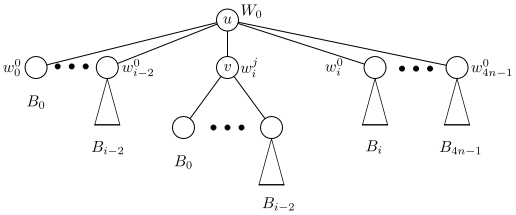


Figure 3 Auxiliary tree A_i^j.

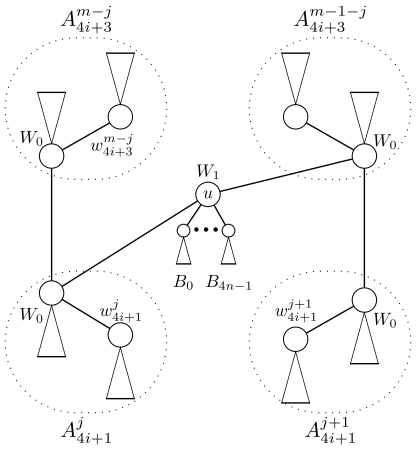


Figure 4 The variable tree $T(y_i^j)$.

▶ **Lemma 8.** *Let $0 \leq i < 4n$ and $0 \leq j \leq m$. Let (T, w) be any weighted tree having B_{4n+2} and A_i^j as subtrees. Let u and v be the root and the sub-root of A_i^j, respectively. For any coloring c of (T, w) with weight $w(c) \leq M$, then it holds:*

- *either v is colored S_{i-1} and u must be colored with the color W_0;*
- *or v is colored S_i (therefore, $w(S_i) \geq w_i^j$) and u can be colored with S_{i-1}.*

Proof. Recall that, by Corollary 6, we can identify the colors of c and their weights. By Lemma 5, the root of each subtree B_k, $0 \leq k < 4n$, must be colored with S_k and then the sub-root v can be colored only with color S_{i-1} or S_i. Note that, if v is colored with color S_p for some $p > i$, then $w(S_p) \geq w_i^j$, contradicting Corollary 6. In the first case, u is adjacent to a node with color S_k, for any $k < 4n$. Therefore, u must be colored with color $S_{4n} = W_0$. Otherwise, u may be colored with color S_{i-1}. ◀

Intuitively, the previous lemma states that, either we "pay" $j\epsilon$ in the weight of color S_i, or u must be colored with the color W_0. We now define the *variable-trees* $T(y_i^j)$ using the auxiliary trees.

▶ **Definition 9.** For any $0 \leq i < n, 0 \leq j < m$, let $T(y_i^j)$ be the weighted rooted tree, *representing the variable* y_i^j, defined as follows (see Figure 4):

- let u be its root with weight $w(u) = W_1$ and connected to the root of a copy of B_ℓ, for any $0 \leq \ell < 4n$;
- take one copy of $A_{4i+1}^j, A_{4i+1}^{j+1}, A_{4i+3}^{m-j}$ and A_{4i+3}^{m-1-j} and:
  - connect $r(A_{4i+1}^j)$ to $r(A_{4i+3}^{m-j})$, and $r(A_{4i+1}^{j+1})$ to $r(A_{4i+3}^{m-1-j})$;
  - connect u with $r(A_{4i+1}^j)$ and $r(A_{4i+3}^{m-j-1})$.

Note that $T(y_i^j)$ consists of $\mathcal{O}(2^{4n})$ nodes (i.e. polynomial in nm).

▶ **Lemma 10.** *Let (T, w) be any weighted tree having B_{4n+2} as subtree and containing $T(y_i^j)$ as subtree, for all $0 \leq i < n$ and $0 \leq j < m$. Let c be a coloring of T with weight $w(c) \leq M$.*

Then, there are $(j_0, \ldots, j_{n-1}) \in \{0, \ldots, m\}^n$ such that each root u of each subtree $T(y_i^j)$, for any $0 \leq i < n$ and $0 \leq j < m$, satisfies:

- *if $j \neq j_i$, then the color of u in c must be W_1;*
- *otherwise, neither of the two neighbors of u can be colored W_1 and neither of these two nodes need to be colored W_0.*

Proof. Since T contains B_{4n+2}, by Corollary 6, a coloring $c = (S_0, \ldots, S_k)$ of weight $w(c) \leq M$ is such that $k \geq 4n + 2$, and, for any $0 \leq i \leq 4n + 2$, S_i is the unique color such that $w(S_i) \in \{w_k^j \mid 0 \leq j \leq m\}$. In particular, $w(c) \geq \sum_{i=0}^{4n+2} 1/2^i = M - n(m-1)\epsilon$.

For any $0 \leq i < n$, let $j_i \leq m$ be such that $w(S_{4i+1}) = w_{4i+1}^{j_i}$.

First, let us assume that $j_i < m$. In particular, this means that every sub-root of a subtree A_{4i+1}^r, for each $j_i < r \leq m$, is colored S_{4i} (recall that its color is either S_{4i} or S_{4i+1}, by Lemma 8). Consequently, any root of a subtree A_{4i+1}^r, for each $j_i < r \leq m$, must be colored W_0. Therefore, by the construction of the variable-trees, any root of a subtree A_{4i+3}^{m-r}, for each $j_i < r \leq m$, cannot be colored W_0 because it is adjacent to a root of a subtree A_{4i+1}^r. Thus, by Lemma 8, it must be colored S_{4i+2} and the color of each sub-root of A_{4i+3}^{m-r} must be S_{4i+3}. Consequently, $w(S_{4i+3}) \geq w_{4i+3}^{m-(j_i+1)}$. Hence, for any $0 \leq i < n$, if $j_i < m$, we conclude that $w(S_{4i+3}) + w(S_{4i+1}) \geq w_{4i+1}^{j_i} + w_{4i+3}^{m-(j_i+1)} = (m-1)\epsilon + 1/2^{4i+1} + 1/2^{4i+3}$.

On the other hand, if $j_i = m$, it follows directly that $w(S_{4i+3}) + w(S_{4i+1}) \geq m\epsilon + 1/2^{4i+1} + 1/2^{4i+3}$.

Since $w(c) \leq M$, it implies that, for any $0 \leq i < n$, $j_i < m$ and $w(S_{4i+3}) = w_{4i+3}^{m-j_i-1}$ and, for any $0 \leq 2k < 4n$, $w(S_{2k}) = w_{2k}^0$. Consequently, by a similar argument, the roots of all subtrees A_{4i+3}^{m-j}, for each $0 \leq j \leq j_i$, must be colored W_0 and, then, the roots of all subtrees A_{4i+1}^r, for each $0 \leq j \leq j_i$, must be colored S_{4i}.

Let $0 \leq i < n$ and $0 \leq j < m$. Consider a subtree $T(y_i^j)$ of T. If $j \neq j_i$, then (exactly) one of the roots of A_{4i+1}^j and A_{4i+3}^{m-1-j} must be colored W_0. In that case, the color of the root u of $T(y_i^j)$ must be W_1. Indeed, u is adjacent to the root of B_k, $0 \leq k < 4n$, and therefore it cannot be colored S_k. Moreover, if u is colored W_2, then we have a contradiction as $w(c) > M$, because $w(u) = W_1$. On the other hand, if $j = j_i$, none of the roots of A_{4i+1}^j and A_{4i+3}^{m-1-j} need to be colored W_0. Finally, none of the roots of A_{4i+1}^j and A_{4i+3}^{m-1-j} can be colored W_1 because their weight is W_0 (it would imply $w(c) > M$). ◀

4.3 Clause-trees and definition of $T(\Phi_{int})$

We define the subtrees representing the clauses and combine them to get $T(\Phi_{int})$.

▶ **Definition 11.** Let $Q_\ell = \vee_{1 \leq k \leq p_\ell} u_k$ be any clause of Φ_{int} (recall that, for any $1 \leq k \leq p_\ell$, $u_k \in \{y_i^j \mid 0 \leq i < n, 0 \leq j < m\}$ and that $\ell \leq L$). For any $1 \leq k \leq p_\ell$, let $T(Q_\ell^k)$ be the rooted weighted tree defined recursively as follows:

1. $T(Q_\ell^1) = T(u_1)$;
2. for any $k > 1$, start with one copy of $T(Q_\ell^{k-1})$ with root a and one copy of $T(u_k)$ with root b. Let c, d be two nodes with weight W_1 and e, f be two nodes with weight W_2. For each node $v \in \{c, d, e, f\}$, and for any $0 \leq i < 4n$, add one copy of B_i and make its root adjacent to v. Add one copy of B_{4n+1} and make its root adjacent to e. Finally, we add the edges $\{\{a, f\}, \{b, c\}, \{c, f\}, \{d, e\}, \{e, f\}\}$ and d is chosen as the root.

Let us note $T(Q_\ell) = T(Q_\ell^{p_\ell})$ the *clause-tree* corresponding to Q_ℓ and that consists of $\mathcal{O}(p_\ell 2^{4n})$ nodes (i.e. polynomial in nm). $T(Q_\ell^k)$ is depicted in Figure 5.

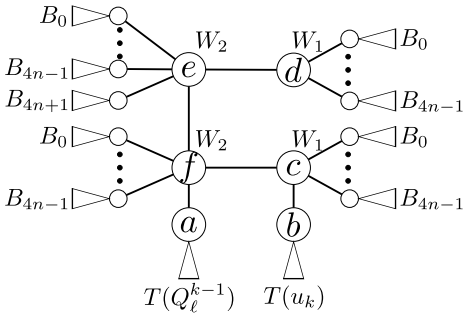


Figure 5 The clause tree $T(Q_\ell^k)$.

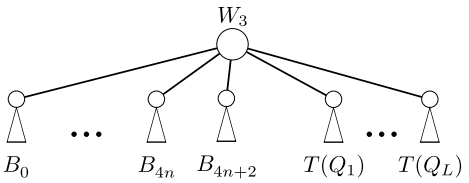


Figure 6 The final tree $T(\Phi_{int})$.

▶ **Lemma 12.** *Let (T, w) be any weighted tree having B_{4n+2} as subtree and containing $T(Q_\ell^k)$ as a subtree ($\ell \leq L$, $k \leq p_\ell$). Let c be any coloring of T with weight $w(c) \leq M$. If a and b are colored W_1, then the color of the root d of $T(Q_\ell^k)$ must be W_1;*

Proof. We prove it by induction on the number of variables k of Q_ℓ^k. Observe that in case $k = 1$, then $T(Q_\ell^k)$ is a variable-tree and the lemma trivially holds as the vertex b does not exist, thus the first statement is trivially satisfied, and, by Lemma 10, the color of its root must be either W_0 or W_1.

Now, consider that a and b are roots of a variable-tree and of a clause-tree on $k-1$ variables $T(Q_\ell^{k-1})$, respectively. By Lemma 10 and by the inductive hypothesis, the colors of a and b are either W_0 or W_1.

In case $c(a) = c(b) = W_1$, by the hypothesis $w(c) \leq M$, by Lemma 5 and Corollary 6, we conclude that c is colored W_0, f is colored W_2, e is colored W_0 and d is forced to be colored W_1. This proves the first statement of the lemma. Finally, by the construction of $T(Q_\ell^k)$, by Lemma 5 and Corollary 6, the root d may be colored either W_0 or W_1, since $w(c) \leq M$. ◀

▶ **Definition 13.** Let $T(\Phi_{int})$ be the weighted rooted tree obtained as follows (see Figure 6). Let r be the root with weight W_3. For any $1 \leq \ell \leq L$, the root of one copy of $T(Q_\ell)$ is made adjacent to r. For any $0 \leq i \leq 4n+2$, $i \neq 4n+1$, r is made adjacent to the root of one copy of B_i.

▶ **Lemma 14.** *$T(\Phi_{int})$ has size polynomial in $m = 2^n$.*

Proof. Observe that each clause-tree $T(Q_\ell)$ has size $\mathcal{O}(p_\ell 2^{4n})$ (see Definition 11), where p_ℓ is polynomial in m (since p_ℓ is at most the number nm of variables). Moreover, the number L of clauses is polynomial in m by the definition of Φ_{int}. ◀

▶ **Lemma 15.** *If Φ_{int} is integrally satisfiable, then $\chi_w(T(\Phi_{int})) \leq M$.*

Proof. Let $(y_i^j)_{i<n, j<m}$ be a valid integral assignment for Φ_{int}. For any $0 \leq i < n$, let j_i be the (unique) index such that $y_i^{j_i}$ is true. We construct a coloring c of $(T(\Phi_{int}), w)$ such that

$w(c) \leq M$. By Lemma 5, in any coloring c of $T(\Phi_{int})$ such that $w(c) \leq M$, the colors of all nodes of the binomial subtrees of $T(\Phi_{int})$ are forced. Consequently, we only need to decide the colors of the following nodes: the roots and sub-roots of any copy of A_i^j, the roots of the trees $T(y_i^j)$, and the nodes added to connect the variables-trees into clause-trees (the nodes a, b, c, d, e, f in Figure 5), for any $0 \leq i < n$ and $0 \leq j < m$.

We first set the weight of color S_i for any $0 \leq i < 4n$. In particular, for any $0 \leq i < n$, the color S_{4i+1} must have weight $w_{4i+1}^{j_i}$. As we observed in the proof of Lemma 10, this choice fixes the colors of all roots and sub-roots of all the A_i^j trees, i.e. all the nodes in the variable trees, except to the roots of the variable-trees $T(y_i^{j_i})$, by Lemma 10.

More precisely, for any $0 \leq i < n$ and $0 \leq j < m$, let us consider a subtree $T(y_i^j)$. Let $j' \in \{j, j+1\}$. The sub-root of $A_{4i+1}^{j'}$ receives color S_{4i+1} if $j' \leq j_i$ and receives color S_{4i} otherwise. The root of $A_{4i+1}^{j'}$ receives color S_{4i} if $j' \leq j_i$ and receives color W_0 otherwise. The sub-root of $A_{4i+3}^{m-j'}$ receives color S_{4i+3} if $j' > j_i$ and receives color S_{4i+2} otherwise. The root of $A_{4i+3}^{m-j'}$ receives color S_{4i+2} if $j' > j_i$ and receives color W_0 otherwise. Finally, if $j \neq j_i$, the root of $T(y_i^j)$ is colored W_1. On the other hand, if $j = j_i$, none of the neighbors of the root of $T(y_i^j)$ is colored W_0, therefore, we can color it either W_0 or W_1.

Now, let $Q_\ell = \vee_{1 \leq k \leq p_\ell} u_k$ be any clause of Φ_{int}. We show that we can extend the previous coloring such that the root of the clause-tree $T(Q_\ell)$ is colored W_0 and the weight of the coloring is $< M$. This is by induction on p_ℓ. Indeed, if $p_\ell = 1$, then Q_ℓ consists of a unique variable and this variable must be assigned to true (since the formula is true). Hence, $Q_\ell = y_i^{j_i}$ for some $0 \leq i < n$. That is $T(Q_\ell)$ is a subtree $T(y_i^{j_i})$. Hence, we can color the root of it with W_0.

Now, assume that the result is correct for any clause of length $p \geq 1$ and let $p_\ell = p + 1$. Thus, $Q_\ell = u_{p+1} \vee (\vee_{1 \leq k \leq p} u_k)$. Recall that $T(Q_\ell)$ is built from a variable subtree $T(u_{p+1})$ and a clause-subtree $T(Q_\ell^p)$. There are two cases to consider: either our assignment satisfies $\vee_{1 \leq k \leq p} u_k$ or not. In the first case, the root of the clause-tree $T(Q_\ell^p)$ (node b in Figure 5) is colored W_0 by induction. Moreover, by above paragraphs, the root of $T(u_{p+1})$ (node a in Figure 5) can be colored W_1. It is then easy to extend this coloring such that the root of $T(Q_\ell)$ is colored W_0: in Figure 5, node c is colored W_1, node e is colored W_2 and nodes f and d are colored W_0. If our assignment does not satisfy $\vee_{1 \leq k \leq p} u_k$, then it must satisfy u_{p+1}. That is, $u_{p+1} = y_i^{j_i}$ for some $0 \leq i < n$. By a similar induction, we prove that the root of $T(Q_\ell^p)$ can be colored W_1. Moreover, by above paragraphs, the root of $T(u_{p+1}) = T(y_i^{j_i})$ can be colored W_0. This coloring can be extended such that the root of $T(Q_\ell)$ is colored W_0: in Figure 5, node f is colored W_1, node e is colored W_2 and nodes c and d are colored W_0.

Thus, we color the roots of all the clause-trees with color W_0 and the root of $T(\Phi_{int})$ with the color W_1.

Hence, the weight of this coloring c is $w(c) = \sum_{i=0}^{4n+2} \frac{1}{2^i} + n(m-1)\epsilon = M$. ◀

▶ **Lemma 16.** *If Φ_{int} is not integrally satisfiable, then $\chi_w(T(\Phi_{int})) > M$.*

Proof. Φ_{int} is not integrally satisfiable. Let c be a coloring of $T(\Phi_{int})$ with weight at most M. By Lemma 10, there are integers $(j_0, \ldots, j_{n-1})$ such that the color of the root of any subtree $T(y_i^j)$ is forced to be W_1, if $j \neq j_i$. Let $Y = (y_i^j)_{i<n, j<m}$ be the corresponding integral assignment. In other words, for any variable y_i^j $(0 \leq i < n, 0 \leq j < m)$, $y_i^j = 0$ if $j \neq j_i$. Since Φ_{int} is not integrally satisfiable, there is a clause Q that is not satisfied by this assignment. Let us consider the clause-subtree $T(Q)$. It has been built from variable-trees corresponding to the variables constituting the clause Q. Because all these variables are assigned to false, the roots of these variable-trees are all colored with W_1, by Lemma 10.

By induction on the length of Q and by Lemma 12, the color of the root of $T(Q_\ell)$ must be W_1. Thus, the root of $T(\Phi_{int})$ can just be colored W_3. Consequently, the coloring c has weight $w(c) \geq \sum_{i=0}^{4n+3} \frac{1}{2^i} + n(m-1)\epsilon > M$. ◀

Proposition 3 follows directly from Lemmas 14, 15 and 16.

References

1 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symp. on Theory of Computing (STOC)*, pages 151–158, 1971.

2 Dominique de Werra, Marc Demange, Bruno Escoffier, Jérôme Monnot, and Vangelis Th. Paschos. Weighted coloring on planar, bipartite and split graphs: Complexity and approximation. *Discrete Applied Mathematics*, 157(4):819–832, 2009.

3 Marc Demange, Dominique de Werra, Jérôme Monnot, and Vangelis Th. Paschos. Weighted node coloring: When stable sets are expensive. In *28th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 2573 of *LNCS*, pages 114–125. Springer, 2002.

4 Leah Epstein and Asaf Levin. On the max coloring problem. *Theor. Comput. Sci.*, 462:23–38, 2012.

5 Bruno Escoffier, Jérôme Monnot, and Vangelis Th. Paschos. Weighted coloring: further complexity and approximability results. *Inf. Process. Lett.*, 97(3):98–103, 2006.

6 D. J. Guan and Xuding Zhu. A coloring problem for weighted graphs. *Inf. Process. Lett.*, 61(2):77–81, 1997.

7 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

8 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

9 Telikepalli Kavitha and Julián Mestre. Max-coloring paths: tight bounds and extensions. *J. Comb. Optim.*, 24(1):1–14, 2012.

10 C. Linhares Sales and B. Reed. Weighted coloring on graphs with bounded tree width. In *Annals of 19th International Symposium on Mathematical Programming*, 2006.

11 Sriram V. Pemmaraju, Sriram Penumatcha, and Rajiv Raman. Approximating interval coloring and max-coloring in chordal graphs. In *Proc. 3rd Int. Workshop on Experimental and Efficient Algorithms (WEA)*, volume 3059 of *LNCS*, pages 399–416. Springer, 2004.

12 Sriram V. Pemmaraju, Sriram Penumatcha, and Rajiv Raman. Approximating interval coloring and max-coloring in chordal graphs. *ACM J. of Exp. Algorithmics*, 10, 2005.

13 Sriram V. Pemmaraju and Rajiv Raman. Approximation algorithms for the max-coloring problem. In *Proceedings 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *LNCS*, pages 1064–1075. Springer, 2005.

14 Sriram V. Pemmaraju, Rajiv Raman, and Kasturi R. Varadarajan. Buffer minimization using max-coloring. In *15th ACM-SIAM Symp. on Discrete Alg. (SODA)*, pages 562–571, 2004.

15 Sriram V. Pemmaraju, Rajiv Raman, and Kasturi R. Varadarajan. Max-coloring and online coloring with bandwidths on interval graphs. *ACM Trans. on Algorithms*, 7(3):35, 2011.

16 Ryan Williams. A new algorithm for optimal constraint satisfaction and its implications. In *31st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3142 of *Lecture Notes in Computer Science*, pages 1227–1237. Springer, 2004.

Generalized Reordering Buffer Management

Yossi Azar*[1], Matthias Englert†[2], Iftah Gamzu[3], and Eytan Kidron[1]

1 **Blavatnik School of Computer Science, Tel-Aviv University, Israel**
{azar,eytankid}@tau.ac.il
2 **Department of Computer Science and DIMAP, University of Warwick, UK**
englert@dcs.warwick.ac.uk
3 **Yahoo! Research**
iftah.gamzu@yahoo.com

Abstract

An instance of the generalized reordering buffer management problem consists of a service station that has k servers, each configured with a color, and a buffer of size b. The station needs to serve an online stream of colored items. Whenever an item arrives, it is stored in the buffer. At any point in time, a currently pending item can be served by switching a server to its color. The objective is to serve all items in a way that minimizes the number of servers color switches. This problem generalizes two well-studied online problems: the paging problem, which is the special case when $b = 1$, and the reordering buffer problem, which is the special case when $k = 1$.

In this paper, we develop a randomized online algorithm that obtains a competitive ratio of $O(\sqrt{b} \ln k)$. Note that this result beats the easy deterministic lower bound of k whenever $b < k^{2-\varepsilon}$. We complement our randomized approach by presenting a deterministic algorithm that attains a competitive ratio of $O(\min\{k^2 \ln b, kb\})$. We further demonstrate that if our deterministic algorithm can employ $k/(1-\delta)$ servers where $\delta \in (0,1)$, then it achieves a competitive ratio of $O(\min\{\ln b/\delta^2, b/\delta\})$ against an optimal offline adversary that employs k servers.

1998 ACM Subject Classification F.2.2 Nonnumerical algorithms and problems

Keywords and phrases Online algorithms, paging, reordering buffer

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.87

1 Introduction

We consider the *generalized reordering buffer management* problem. In this problem, there is a service station, which is equipped with k servers and a buffer of size b. Each of the servers is initially configured with some color. An online stream of colored items has to be served by the service station. Whenever an item arrives, it is stored in the buffer. At any point in time, a currently pending item can be served by removing it from the buffer and switching a server to its color. In particular, if one of the servers is already configured with the color of a pending item, this item can be served without switching any server. The goal is to serve all items while minimizing the overall number of color switches of the servers.

This problem is a natural generalization of two well-studied online problems: the *paging* problem, introduced by Sleator and Tarjan [19], is the special case when $b = 1$, and the *reordering buffer* problem, introduced by Räcke et al. [18], is the special case when $k = 1$.

* Supported in part by the Israel Science Foundation (grant No. 1404/10) and by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).
† Supported in part by EPSRC award EP/D063191/1 and EPSRC grant EP/F043333/1.



© Yossi Azar, Matthias Englert, Iftah Gamzu, and Eytan Kidron;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 87–98



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



Apart from this, the problem is also an interesting abstraction of a number of problem scenarios occurring in computer science and manufacturing. We give two examples.

- Consider a network device that can maintain a maximum of k TCP/IP connections open at the same time. This device receives an online stream of unit sized packets that need to be forwarded to specific destinations. Arriving packets can be forwarded if they are addressed to one of the k currently open connections; otherwise, they have to be stored in a finite-sized random access packet buffer. If the buffer is full, the device needs to close one of the k open connections, and open a new one to the destination of one or more packets stored in the buffer. These packets can then be forwarded using the new connection and free up space in the buffer. The goal is to transmit all packets while minimizing the number of connection open/close operations.
- In the painting shop of a car manufacturing plant, car bodies traverse the final layer of painting, where each car body is painted with its own predetermined top coat. The painting shop is equipped with k painting machines and a finite-sized parking lot. Each machine, once configured with a color, can paint multiple cars with that color. However, switching a color in a machine causes non-negligible set-up and cleaning cost. The goal of the painting shop is to paint all incoming cars with a minimum number of color switches. The parking lot can be used to change the order in which the cars are painted, but it must never overflow.

Chan et al. [9] seem to have been the first to mention the generalized reordering buffer problem. They established that the offline variant of the problem, in which the entire stream of items is known in advance, is NP-hard. To the best of our knowledge, no other work has been done on generalized reordering buffer. In particular, no results are known for the online setting of the problem which we study.

Our results. We develop a randomized online algorithm that attains a competitive ratio of $O(\sqrt{b}\ln k)$. Note that for any $b < k^{2-\varepsilon}$ with $\varepsilon > 0$, our randomized upper bound beats the easy deterministic lower bound of k that applies for any b. Our algorithm has its roots in the randomized marking algorithm of Fiat et al. [12] for the paging problem, combined with several clean-up procedures. We emphasize that one natural approach for designing an algorithm for our generalized setting is to combine an algorithm for the reordering buffer problem with an algorithm for the paging problem. Specifically, the reordering buffer algorithm decides which color to serve next, and the paging algorithm decides which server should switch to that color. Unfortunately, we do not know if there are combinations of this nature that attain good performance guarantees. For example, we could combine the $O(\ln k)$-competitive randomized marking algorithm for paging with a deterministic algorithm for the reordering buffer problem, some of which are $O(\ln b)$-competitive or better. While one can easily prove that this combination is an $O(b \ln k)$-competitive algorithm, this guarantee is still weak; for example, when $b = k$. In fact, we were unable to show any bounds that are sublinear in k for such combinations.

We further present a deterministic online algorithm that attains a competitive ratio of $O(\min\{k^2 \ln b, kb\})$. This algorithm can complement our randomized algorithm in cases where b is large. Note that any lower bound for the paging problem also applies to the generalized reordering buffer management problem. To see this, consider any sequence of requests for pages (colors). This sequence can be modified by replacing each request to a page by b successive requests to that page. This does not change the cost for the paging problem, but it neutralizes the buffer of size b in the generalized reordering buffer

problem, i.e., at any time, all pending requests are always for the same page. Therefore, no deterministic online algorithm can be better than k-competitive for our problem, and our randomized online algorithm outperforms the best possible deterministic online algorithm for any $b = o((k/\log k)^2)$. We also show that if our deterministic algorithm can employ $k/(1-\delta)$ servers where $\delta \in (0,1)$, then it achieves a competitive ratio of $O(\min\{\ln b/\delta^2, b/\delta\})$ against an optimal offline adversary that employs k servers. Notice that this implies that if our algorithm can employ a constant fraction of servers more than the optimal algorithm then it achieves logarithmic competitiveness.

Note that due to space constraints, some proofs are omitted from this extended abstract and may be found in the full version of the paper.

Related work. The paging problem was introduced in the seminal paper of Sleator and Tarjan [19]. They proved that the LRU and FIFO strategies are k-competitive, and established that no deterministic algorithm can achieve a competitive ratio smaller than k. Karlin et al. [14] showed that the same bound is achieved by the flush-when-full strategy. Fiat et al. [12] presented a randomized $2H_k$-competitive marking algorithm, outperforming the lower bound for deterministic algorithms. Note that H_k is the kth harmonic number. They also established a lower bound of H_k on the competitive ratio that any randomized algorithm can achieve. Later on, Achlioptas et al. [1] and McGeoch and Sleator [17] provided algorithms matching this lower bound.

The reordering buffer problem was introduced by Räcke et al. [18], who devised an $O(\ln^2 b)$ -competitive algorithm. Englert and Westermann [11] presented an algorithm with an improved competitive ratio of $O(\ln b)$, which can be applied to a generalized non-uniform cost setting. Avigdor-Elgrabli and Rabani [4] developed an LP-based algorithm whose competitive ratio is $O(\ln b/\ln\ln b)$. Adamaszek et al. [2] presented an algorithm whose competitive ratio is $O(\sqrt{\ln b})$, and established lower bounds of $\Omega(\sqrt{\ln b/\ln\ln b})$ and $\Omega(\ln\ln b)$ for deterministic and randomized algorithms, respectively. Recently, Avigdor-Elgrabli and Rabani [6] developed a randomized online algorithm matching this lower bound.

Asahiro et al. [3] and Chan et al. [9] considered the offline variant of the reordering buffer problem, and established that it is NP-hard. Very recently, Avigdor-Elgrabli and Rabani [5] designed a constant factor approximation algorithm for this offline setting. The reordering buffer problem has also been studied on other metric spaces [8, 15, 13]. For example, Englert et al. [10] considered the more general variant in which items are associated with points in a metric space, and obtained a competitive ratio of $O(\ln^2 b \ln n)$, where n is the number of distinct points in the metric. Some research has been done on a maximization variant of the problem [16, 7].

2 A Randomized Algorithm

In this section, we develop a randomized algorithm whose competitive ratio is $O(\sqrt{b}\ln k)$. The algorithm sensibly combines the randomized marking algorithm due to Fiat et al. [12] with several buffer clean-up procedures.

2.1 The algorithm

We begin by briefly describing the random marking algorithm for the paging problem. Recall that in the underlying paging setting, there are k servers and a trivial buffer (i.e., $b = 1$). The algorithm is made up of *phases*. At the beginning of each phase, all colors are *unmarked*. When an item of color χ arrives, χ becomes marked and a server is moved to χ if there

is no server there already. In order to decide which server moves, the algorithm chooses a server uniformly at random from the servers which are located on unmarked colors. If there is no such server, then the current phase ends and a new phase begins. At the end of a phase, markings are removed from all colors. A crucial observation regarding this algorithm is that although it has random components, the order in which the colors are marked is deterministic and so is the partition into phases. We make use of this property also in the analysis of our algorithm.

Our algorithm combines the random marking algorithm with several buffer clean-up procedures. Similarly to the marking algorithm, our algorithm is also made up of phases. Each color has a *phase counter*, which is set to 0 at the beginning of each phase. When an item of color χ arrives, it is placed in the buffer, and the phase counter of χ is incremented by 1. When the phase counter of a color reaches $\sqrt{b}$, the color becomes marked and is served in a similar way to the marking algorithm. Note that the partition into phases is defined as in the marking algorithm.

We further define a status to each color: *marked*, *half-marked*, or *unmarked*. At the end of a phase, all half-marked colors become unmarked, and all marked colors become half-marked. Accordingly, arriving items are said to be either marked, half-marked or unmarked depending on the status of their color at their arrival time. For example, an item is said to be half-marked if its color was marked in the previous phase. We partition the buffer into two sub-buffers, each of size $b/2$. The *unmarked sub-buffer* stores unmarked items and the *half-marked sub-buffer* stores half-marked items. Note that marked items never stay in the buffer since a marked color always consists of a server that can immediately serve the arriving item. In fact, this may also be true for half-marked items, but it is never true for unmarked items. Namely, half-marked colors may or may not consist a server, but unmarked colors never consist of a server. If a half-marked item arrives, and a server is present on its corresponding color, then it is served immediately and does not need to be placed in the half-marked sub-buffer. Note that the phase counter of that color is still incremented. In order to avoid buffer overflows, we introduce the following three clean-up procedures:

Half-marked clean-up. A half-marked clean-up event takes place when the half-marked sub-buffer is full, and also at the end of each phase. Upon a *half-marked clean-up event*, all items in the half-marked sub-buffer are served. This is done by moving an arbitrary server through all the colors of items in the half-marked sub-buffer. The server then returns to its original position. Note that the number of half-marked clean-up moves in a half-marked clean-up event is one plus the number of different colors in the half-marked sub-buffer at the time of the event.

Targeted clean-up. A targeted single-color clean-up event takes place when a *buffer counter* of some color reaches $2\sqrt{b}$. The buffer counter maintains the number of items a color has in the unmarked sub-buffer. Note that this counter should not be confused with the phase counter used for marking. In a targeted single-color unmarked clean-up event, or simply *targeted clean-up event*, the items of a single color from the unmarked sub-buffer are served. An arbitrary server moves to that color and back to its original position. Hence, a targeted clean-up event consists of two targeted clean-up moves.

Unmarked clean-up. An unmarked clean-up event takes place if there are $\sqrt{b}/4$ different colors in the unmarked sub-buffer, and after $\sqrt{b}$ targeted clean-up events. Similarly to a half-marked clean-up event, upon an *unmarked clean-up event*, all items in the unmarked

sub-buffer are served by an arbitrary server, which later returns to its original position. The number of unmarked clean-up moves in an unmarked clean-up event is one plus the number of different colors in the unmarked sub-buffer at the time of the event.

2.2 The analysis

We begin by pointing out that although the algorithm is randomized, it still has several deterministic aspects: the points in time in which each color becomes marked, half-marked and unmarked are deterministic, and so is the partition into phases. In particular, the partition into phases is deterministic since a phase ends just before some color gets marked when each of precisely k different colors already got $\sqrt{b}$ items in that phase. The content of the unmarked sub-buffer at every point in time is also deterministic, and consequently, so is the point in time of every unmarked clean-up event and targeted clean-up event. Finally, we point out that although the content of the half-marked sub-buffer is not deterministic, and neither is the point in time of half-marked clean-up events, at the end of each phase the half-marked sub-buffer is emptied. Thus, the content of the entire buffer is deterministic at the beginning of every phase. The following lemma establishes the feasibility of the algorithm.

▶ **Lemma 1.** *The algorithm never has a buffer overflow.*

Proof. Recall that the buffer is partitioned into two sub-buffers, each of size $b/2$. The half-marked sub-buffer never overflows since whenever it becomes full, a half-marked clean-up event is initiated. The unmarked items in the unmarked sub-buffer are served by a combination of unmarked clean-up events and targeted clean-up events. The unmarked clean-up events make sure that there are never more than $\sqrt{b}/4$ different colors in the unmarked sub-buffer, and the targeted clean-up events ensure that no such color has more than $2\sqrt{b}$ items in the sub-buffer. Together, there cannot be more than $b/2$ unmarked items in that sub-buffer. ◀

Let ON and OPT denote our algorithm and the optimal offline algorithm, respectively. We also denote the respective overall number of server moves in ON and OPT by ON and OPT. Note that ON is the expected number of moves since ON is randomized. In the remainder of this section, we prove that ON is $O(\sqrt{b}\ln k)$-competitive. We first partition ON according to four types of moves that the servers of ON do: $ON = ON^M + ON^H + ON^T + ON^U$, where ON^M is the expected number of *marking* moves, ON^H is the expected number of *half-marked clean-up* moves, ON^T is the number of *targeted clean-up* moves and ON^U is the number of *unmarked clean-up* moves. We also define ON_i to be the expected number of ON's server moves in phase i, and OPT_i is the number of OPT's server moves in phases $i-1$ and i. Note this latter asymmetry, and notice that $OPT \leq \sum_i OPT_i \leq 2 \cdot OPT$. Similarly, for a set S of consecutive phases, we define $ON_S = \sum_{i \in S} ON_i$. Note that we also use the same notation as before when considering a partition of the expected number of ON's server moves in a phase or a set of phases to the four types of moves. For example, ON_i^U is the expected number of unmarked clean-up moves that ON servers makes in phase i. We now turn to bound the ratios between each of ON's movement types and OPT. The competitive ratio ON/OPT is the sum of these ratios.

Marking moves and half-marked clean-up moves. We bound the expected number of marking moves, ON^M, and the expected number of half-marked clean-up moves, ON^H, using similar techniques. Let $H_k = \sum_{j=1}^{k} 1/j$ be the kth harmonic number, and let m_i be

the number of colors that are marked in phase i but not marked in phase $i-1$. We start with a bound on the expected number of marking moves.

▶ **Lemma 2.** *For every phase i, $ON_i^M \leq m_i H_k$.*

The proof uses the same arguments as the proof for randomized marking algorithm by Fiat et al. [12] and is deferred to the full version of the paper. We now can use Lemma 2 to also derive our desired upper bound on half-marked clean-up moves.

▶ **Lemma 3.** *For every phase i, $ON_i^H = O(\min\{m_i\sqrt{b}\ln k, m_i^2 \ln k/\sqrt{b} + m_i\})$.*

Proof. Recall that half-marked items may only accumulate at colors which had a server at the beginning of the phase, and that server left the color during that phase. Lemma 2 implies that there can be at most $m_i H_k$ such colors in expectation. Since at most $\sqrt{b}$ half-marked items can arrive from each such color, no more than $m_i H_k \sqrt{b}$ items enter the half-marked sub-buffer during phase i in expectation.

An immediate consequence of the above observation is that $ON_i^H = O(m_i\sqrt{b}\ln k)$. This follows since each half-marked clean-up move, except the last move in each such event, cleans at least one item. The last moves of all half-marked clean-up events add at most a multiplicative factor of 2 to the number of half-marked clean-up moves.

For the purpose of proving that $ON_i^H = O((m_i^2 \ln k)/\sqrt{b} + m_i)$, notice that the half-marked sub-buffer is full with $b/2$ items every time that half-marked clean-up event is initiated (except maybe the last half-marked clean-up event in every phase). So the number of events is upper bounded in expectation by $m_i H_k \sqrt{b}/(b/2) + 1 = 2m_i H_k/\sqrt{b} + 1$. Note that the additional 1 is due to the half-marked clean-up event done at the end of each phase. Now, at any given time, there are no more than m_i void half-marked colors, namely, colors without a server. This implies that there are no more than m_i colors in the half-marked sub-buffer, and hence, a server makes at most $m_i + 1$ moves in each such event. As a result, $ON_i^H = (2m_i H_k/\sqrt{b} + 1)(m_i + 1) = O(m_i^2 \ln k/\sqrt{b} + m_i)$. ◀

We next set a bound of the ratio between $ON^M + ON^H$ and OPT. For this purpose, we partition the phases into groups of *marking phase sequences*. Each marking phase sequence contains consecutive phases, and each phase belongs to exactly one marking phase sequence. We let S denote the set of phases in a marking phase sequence, and use $\text{last}(S)$ to denote the last phase in S. Our partition maintains the property that in every marking phase sequence S (except maybe the last one), $m_i \leq 3\sqrt{b}$ for every $i \in S \setminus \{\text{last}(S)\}$, and $m_{\text{last}(S)} > 3\sqrt{b}$. Namely, the partition is set according to phases i such that $m_i > 3\sqrt{b}$. Let $m_S = \sum_{i \in S} m_i$.

▶ **Lemma 4.** *For every marking phase sequence S, $ON_S^M + ON_S^H = O((m_S + m_{last(S)}\sqrt{b})\ln k)$.*

Proof. Notice that $ON_S^M = \sum_{i \in S} ON_i^M \leq \sum_{i \in S} m_i H_k = O(m_S \ln k)$, where the inequality holds by Lemma 2. In addition, observe that

$$\begin{aligned} ON_S^H &= \sum_{i \in S \setminus \{\text{last}(S)\}} ON_i^H + ON_{\text{last}(S)}^H \\ &= \sum_{i \in S \setminus \{\text{last}(S)\}} O((m_i^2 \ln k/\sqrt{b}) + m_i) + O(m_{\text{last}(S)}\sqrt{b}\ln k) \\ &= \sum_{i \in S \setminus \{\text{last}(S)\}} O\left(m_i \ln k\right) + O(m_{\text{last}(S)}\sqrt{b}\ln k) = O((m_S + m_{\text{last}(S)}\sqrt{b})\ln k)\ , \end{aligned}$$

where the second equality follows from Lemma 3, and the third equality holds since $m_i \leq 3\sqrt{b}$ for every $i \in S \setminus \{\text{last}(S)\}$. ◀

We now turn to set a lower bound on OPT_S, where OPT_S is the overall number of server moves of OPT in S and the last phase before S. Recall that OPT_i is the number of server moves of OPT in phases i and $i-1$, and therefore, $OPT_S \leq \sum_{i \in S} OPT_i \leq 2 \cdot OPT_S$. We first set a bound applicable for all marking phase sequences S having a large m_S.

▶ **Lemma 5.** *$OPT_S = \Omega(m_S/\sqrt{b})$, for every marking phase sequence S having $m_S \geq 3\sqrt{b}$.*

Proof. Observe that in each pair of phases $i-1$ and i exactly $k+m_i$ colors are marked, and the servers of OPT are present in no more than $k+OPT_i$ colors. Hence, there are at least $\sqrt{b} \cdot (m_i - OPT_i)$ items that entered the buffer of OPT during phases $i-1$ and i. The overall number of items entering OPT's buffer during S and the last phase before S is therefore at least $\sqrt{b}/2 \cdot \sum_{i \in S}(m_i - OPT_i)$, where the half factor is due to the fact that every item may be counted twice. Now, notice that in each of the OPT_S server moves, OPT can clear no more than b items from its buffer. Moreover, there can be at most b items that may stay in the buffer and not cleared at the end of S. Hence,

$$OPT_S \geq \frac{\sqrt{b}/2 \cdot \sum_{i \in S}(m_i - OPT_i) - b}{b} \geq \frac{m_S}{2\sqrt{b}} - \frac{OPT_S}{\sqrt{b}} - 1 \ ,$$

where the last inequality holds since $\sum_{i \in S} OPT_i \leq 2 \cdot OPT_S$. This implies that $OPT_S = \Omega(m_S/\sqrt{b})$ since $m_S \geq 3\sqrt{b}$. ◀

The next lemma establishes a more specialized bound for all marking phase sequences S having a large $m_{\text{last}(S)}$.

▶ **Lemma 6.** *$OPT_S = \Omega(m_S/\sqrt{b} + m_{last(S)})$, for every marking phase sequence S having $m_{last(S)} > 3\sqrt{b}$.*

Proof. The fact that $OPT_S = \Omega(m_S/\sqrt{b})$ follows from Lemma 5 by noticing that $m_S \geq m_{\text{last}(S)} > 3\sqrt{b}$. We now complete the proof by establishing that $OPT_S = \Omega(m_{\text{last}(S)})$. We next prove a somewhat stronger argument stating that $OPT_i = \Omega(m_i)$, for every phase i such that $m_i > 3\sqrt{b}$. As a consequence, we attain that $OPT_S \geq OPT_{\text{last}(S)} = \Omega(m_{\text{last}(S)})$ since $m_{\text{last}(S)} > 3\sqrt{b}$. For the purpose of proving the above argument, notice that the number of items which arrived during phases $i-1$ and i, and entered OPT's buffer is at least $\sqrt{b} \cdot (m_i - OPT_i)$. Since OPT's buffer cannot overflow, we attain that $b \geq \sqrt{b} \cdot (m_i - OPT_i)$, and therefore, $OPT_i \geq m_i - \sqrt{b} > 2m_i/3$, where the last inequality holds since $m_i > 3\sqrt{b}$. ◀

The main result of this subsection is the following lemma.

▶ **Lemma 7.** *$ON^M + ON^H = O(\sqrt{b} \ln k) \cdot OPT + O(b \ln k)$.*

Proof. Notice that $OPT \geq \sum_S OPT_S/2$. Hence, it is sufficient that we establish the above mentioned bound for each marking sequence. Lemma 4 and Lemma 6 prove that $ON^M_S + ON^H_S = O(\sqrt{b} \ln k) \cdot OPT_S$, for every marking phase sequence S except maybe the last marking phase sequence. Consider the last marking phase sequence S'. If $m_{\text{last}(S')} > 3\sqrt{b}$ then the same bound also holds for S'. Otherwise, if $m_{S'} > 3b$ then from Lemma 5 we know that $OPT_{S'} = \Omega(m_{S'}/\sqrt{b})$, while from Lemma 4 we attain that $ON^M_{S'} + ON^H_{S'} = O(m_{S'} \ln k)$. Namely, the same bound ratio holds also in this case. Finally, when $m_{\text{last}(S')} \leq 3\sqrt{b}$ and $m_{S'} < 3b$, we get that $ON^M_{S'} + ON^H_{S'} = O(b \ln k)$, which is exactly the additive term in the above ratio. ◀

Targeted clean-up moves. We now turn our attention to bound the number of targeted clean-up moves ON^T. Recall that each targeted clean-up event consists of two server moves, and each such event happens when there are $2\sqrt{b}$ items of a single color in the unmarked sub-buffer.

▶ **Lemma 8.** *There is no time interval during which no server of OPT moves but more than $2\sqrt{b}$ targeted clean-up events occur.*

Proof. Let us assume by way of contradiction that there exists a time interval I during which no server of OPT moves and there are more than $2\sqrt{b}$ targeted clean-up events. Recall that after every $\sqrt{b}$ targeted clean-up events, the unmarked sub-buffer is cleared by an unmarked clean-up event. This implies that the last $\sqrt{b}$ targeted clean-up events in I clear items that arrived during I. We next focus only on those $\sqrt{b}$ targeted clean-up events. We number them by $1, \ldots, \sqrt{b}$.

Let χ_j be the color cleared in the jth targeted clean-up event. We say that a targeted clean-up event j is an *OPT-present* clean-up event if OPT has a server on χ_j during I; otherwise, this event is *OPT-absent*. Let ℓ be the number of OPT-present events and $\sqrt{b} - \ell$ be the number of OPT-absent events. In each of the $\sqrt{b} - \ell$ OPT-absent events, OPT accumulates $2\sqrt{b}$ items arriving during I. We count only the first $\sqrt{b}$ items in each such event due to a reason that will be explained later. Thus, summing up over all OPT-absent events, it follows that OPT accumulates at least $\sqrt{b}(\sqrt{b} - \ell)$ items. The crucial observation needed to complete the proof is that every OPT-present clean-up event implies that there is an OPT server missing from a marked color at some phase. Specifically, let us concentrate on an OPT-present clean-up event j. The $2\sqrt{b}$ unmarked items cleared at that event must have been accumulated in the buffer of ON during at least two marking phases; otherwise, the underlying color would have been marked. Let i_j be the first phase during which the unmarked items cleared by the jth targeted clean-up event started accumulating. Let d_i be the number of OPT-present targeted clean-up events whose items started accumulating at phase i, that is, $d_i = |\{j : i = i_j\}|$. Notice that during the marking phase i, the servers of OPT are not present on at least d_i colors that become marked. Since $\sqrt{b}$ items arrive to each of those colors, OPT accumulates in its buffer at least $d_i\sqrt{b}$ items during that phase. As a result, OPT accumulated at least $\ell\sqrt{b}$ additional items as $\sum_i d_i = \ell$.

Notice that the number of items accumulated in OPT's buffer during I is at least $\sqrt{b}(\sqrt{b} - \ell) + \ell\sqrt{b} = b$, a contradiction to our assumption that OPT does not move during I. Recall that we assumed that OPT accumulates $\sqrt{b}$ (and not $2\sqrt{b}$) items in each OPT-absent event. This is required to ensure that the sets of accumulated items due to OPT-absent and OPT-present events are disjoint. In general, an item cleared in a targeted clean-up move may be counted as one of $\sqrt{b}$ items that induced a marking move. However, none of the first $\sqrt{b}$ items accumulated may be counted towards a marking move since otherwise, the underlying color becomes marked before the buffer counter reaches $2\sqrt{b}$, and thus, a targeted clean-up event cannot occur. ◀

Lemma 8 implies that $ON^T \leq 4\sqrt{b} \cdot (OPT + 1)$ since every targeted clean-up event consists of two targeted clean-up moves. Since we may assume that OPT moves at least once, this immediately gives us the following lemma.

▶ **Lemma 9.** $ON^T = O(\sqrt{b}) \cdot OPT$.

Unmarked clean-up moves. We finally bound the number of unmarked clean-up moves ON^U. Recall that an unmarked clean-up event takes place when either (1) the unmarked

sub-buffer has $\sqrt{b}/4$ different colors, or (2) after $\sqrt{b}$ targeted clean-up events. For the sake of the analysis, it is sufficient that we focus only on unmarked clean-up moves due to type (1). Clean-up moves due to type (2) can be charged against the targeted clean-up moves with an additional constant multiplicative factor. Specifically, one can observe that the number of unmarked clean-up moves of type (2) is no more than $ON^T/8$. During $\sqrt{b}$ targeted clean-up events there are $2\sqrt{b}$ targeted clean-up moves, while the unmarked clean-up event that results from this sequence of targeted clean-ups has at most $\sqrt{b}/4$ unmarked clean-up moves; otherwise, an unmarked clean-up event of type (1) would take place before that. As a result, in the remainder of this subsection, when we refer to unmarked clean-up events or moves, we specifically mean unmarked clean-up events or moves of type (1).

Let u_i be the number of unmarked clean-up events in phase i. We partition the marking phases into groups of *clean-up phase sequences*. A clean-up phase sequence is a sequence of consecutive marking phases such that each phase belongs to exactly one clean-up phase sequence. A clean-up phase sequence S ends when $\sum_{i \in S} u_i > 60\sqrt{b}$. Let $u_S = \sum_{i \in S} u_i$, and observe that a straightforward upper bound on the number of unmarked clean-up moves in any sequence S is $ON^U_S = O(u_S\sqrt{b})$ since every unmarked clean-up event has $\sqrt{b}/4 + 1$ moves. Since we do not have an upper bound on u_S, it is convenient to consider two types of clean-up phase sequences: a clean-up phase sequence S that has a phase $i \in S$ such that $u_i > 6\sqrt{b}$, and a sequence S that does not have such a phase.

▶ **Lemma 10.** *$OPT_i = \Omega(u_i)$ for a phase i such that $u_i > 6\sqrt{b}$.*

Proof. Notice that all the unmarked items that were cleared during phase i, with the exception of the items cleared in the first unmarked clean-up event of the phase, arrived during phase i. As a result, the number of unmarked items which arrive during phase i is at least $(u_i - 1) \cdot \sqrt{b}/4$. Note that no color is associated with more than $\sqrt{b}$ of these items. In phase $i-1$, there are k marked colors. None of the previously mentioned $(u_i - 1) \cdot \sqrt{b}/4$ unmarked items can be from those colors as any item of those colors arriving in phase i would not be considered an unmarked item. Let us restrict our attention to the first $\sqrt{b}$ items of each of those marked colors. Summing up, we know that $(u_i - 1) \cdot \sqrt{b}/4 + k\sqrt{b}$ items arrived during phases $i-1$ and i, and no single color has more than $\sqrt{b}$ of these items.

During phases $i-1$ and i, the servers of OPT could not have been located in more than $k + OPT_i$ different colors. Therefore, they could have served no more than $\sqrt{b} \cdot (k + OPT_i)$ of the previously mentioned items. The remaining items must have entered the buffer of OPT. Since OPT's buffer cannot accommodate more than b items, then

$$\frac{(u_i - 1)\sqrt{b}}{4} + k\sqrt{b} - \sqrt{b} \cdot (k + OPT_i) = \sqrt{b} \cdot \left(\frac{u_i - 1}{4} - OPT_i \right) \leq b \ .$$

This implies that $OPT_i \geq (u_i - 1)/4 - \sqrt{b} \geq u_i/24$, where the last inequality holds since $u_i > 6\sqrt{b}$. ◀

We now introduce the notion of an *extended phase*. An extended phase is defined only for phases i with $u_i > 0$. The extended phase includes phase i and phases $i-1, i-2$, and so on, until a phase i' with $u_{i'} > 0$. As a result, any extended phase contains at least two phases, and only the first and last of them has unmarked clean-up events. For any clean-up phase sequence S, let OPT_S be the number of moves of OPT servers during the extended phases of all relevant $i \in S$.

▶ **Lemma 11.** *$OPT_S = \Omega(\sqrt{b})$ for any clean-up phase sequence S such that $u_i \leq 6\sqrt{b}$ in all phases $i \in S$.*

Proof. Let us assume by way of contradiction that $OPT_S \leq \sqrt{b}/60$. Let χ_S be the set of colors that OPT visited during the extended phases of S. Since OPT makes at most $\sqrt{b}/60$ moves, $|\chi_S| \leq k + \sqrt{b}/60$. Let x_i be the number of items arriving during the extended phase i whose colors are not in χ_S. We next argue that $x_i \geq u_i\sqrt{b}/30$ for every extended phase $i \in S$. Then, we get that the number of items arriving during the extended phases of S whose colors are not in χ_S is at least $\sum_{i\in S} x_i/2 \geq \sum_{i\in S} u_i\sqrt{b}/60 > b$, where the half factor is due to the fact that every item belongs to at most two extended phases, and the last inequality results since $u_S > 60\sqrt{b}$. This implies that OPT must have accumulated more than b items in its buffer, a contradiction.

We turn to prove the above-mentioned argument. Let us focus on the extended phase i, and recall that phase $i-1$ is within this extended phase. At phase $i-1$, k colors receive at least $\sqrt{b}$ items and become marked. Let R_i be the set of these colors that are not in χ_S, and let $r_i = |R_i|$. Note that at least $r_i\sqrt{b}$ items arrive out of χ_S. If $r_i \geq u_i/30$ then $x_i \geq u_i\sqrt{b}/30$, and we are done. Hence, in the remainder of this proof, we may assume that $r_i < u_i/30$ and $|\chi_S \cup R_i| < k + \sqrt{b}/60 + u_i/30$. Notice that no more than $\sqrt{b}/60 + u_i/30$ of the colors in $\chi_S \cup R_i$ may correspond to unmarked items in phase i. This follows since $\chi_S \cup R_i$ includes the k colors marked in phase $i-1$. Each unmarked clean-up event cleans items from $\sqrt{b}/4$ different unmarked colors, and at least $\sqrt{b}/4 - \sqrt{b}/60 - u_i/30$ items of colors outside χ_S. However, $\sqrt{b}/4 - \sqrt{b}/60 - u_i/30 = 7\sqrt{b}/30 - u_i/30 \geq \sqrt{b}/30$, where the last inequality follows since $u_i \leq 6\sqrt{b}$ in any phase $i \in S$. This implies that at least $u_i \cdot \sqrt{b}/30$ unmarked items were cleaned in phase i. Notice that an unmarked item cleaned in phase i must have arrived at the extended phase i, and thus, $x_i \geq u_i\sqrt{b}/30$. ◀

We can now complete the main contribution of this subsection.

▶ **Lemma 12.** $ON^U = O(\sqrt{b}) \cdot OPT + O(b)$.

Proof. Notice that $OPT \geq \sum_S OPT_S/2$. Hence, it is sufficient that we establish the above mentioned bound for each clean-up phase sequence. We prove that $ON^U_S = O(\sqrt{b}) \cdot OPT_S$, for every clean-up phase sequence S except the last one. We then complete the proof by demonstrating that the last clean-up phase sequence contributes an additive value of $O(b)$.

Consider a clean-up phase sequence S that is not the last one. Observe that if there is a phase $i \in S$ such that $u_i > 6\sqrt{b}$ then $u_S = \Theta(\max_{i\in S} u_i)$. This observation uses the fact that such a clean-up phase sequence ends when $\sum_{i\in S} u_i > 60\sqrt{b}$. Using Lemma 10, one can infer that $OPT_S = \Omega(\max_{i\in S} u_i)$, and the claimed bound follows by recalling that $ON^U_S = O(u_S\sqrt{b})$. In case that $u_i \leq 6\sqrt{b}$ for all phases i of S, then $u_S = O(\sqrt{b})$, and therefore, $ON^U_S = O(b) = O(\sqrt{b}) \cdot OPT_S$, where the last equality follows from Lemma 11. Now, let us focus on the last clean-up phase sequence S'. Clearly, $u_{S'} < 60\sqrt{b}$, and hence, $ON^U_{S'} = O(b)$. ◀

Putting everything together. Combining the bounds from Lemma 7, Lemma 9, and Lemma 12, gives the main theorem of this section. Note that in adherence to competitive analysis and online algorithms research, we allow additive terms that are independent of the input stream and its properties.

▶ **Theorem 13.** *There is a randomized algorithm whose competitive ratio is* $O(\sqrt{b}\ln k)$.

3 A Deterministic Algorithm

We develop two deterministic algorithms: the first has a competitive ratio of $O(\ln b/\delta^2)$ in a δ-augmentation setting and a competitive ratio of $O(k^2 \ln b)$ when there is no server

augmentation, and the other has a competitive ratio of $O(b/\delta)$ in a δ-augmentation setting and a competitive ratio of $O(kb)$ with no augmentation. Then, one can execute the algorithm that achieves a better competitive ratio depending on the underlying parameters k, b, and δ. This results in a $O(\min\{k^2 \ln b, kb\})$-competitive algorithm, and a $O(\min\{\ln b/\delta^2, b/\delta\})$-competitive algorithm for the δ-augmentation settings. Note that in a δ-augmentation setting, an online algorithm may employ $k/(1-\delta)$ servers where $\delta \in (0,1)$, while an optimal offline adversary can employ at most k servers. Also note that all the proofs of this section can be found in the full version of the paper.

Algorithm 1. Our first algorithm sensibly combines the algorithm for the reordering buffer problem on arbitrary metric spaces [10], and the FIFO algorithm for the paging problem [19]. Specifically, we utilize the algorithm for reordering buffer to decide which color to serve next, and then use the FIFO algorithm to decide which of the servers moves to serve this color. Formally, the algorithm consists of alternating selection and service steps. We maintain a cost $c_\chi \in [0,1]$ for every color χ, which is initially 0. During the *selection* step the buffer is full, and the cost of all inactive colors, namely, colors that currently do not consist of a server, is incremented. The cost of each color is incremented at a rate proportional to the number of items it has in the buffer. Once the cost of some color reaches 1, this color is selected, and the selection step ends. If more than one color reaches a cost of 1 then one of them is selected arbitrarily. Once the selection step ends, the service step begins. In the *service* step, a server is moved to the selected color. The choice of which server should be moved is done in a FIFO manner, that is, we move the server that has not moved the longest. Then, the cost of the selected color drops from 1 to 0, and all its items in the buffer are cleared and served. This makes room in the buffer for more items. Arriving items from active colors are served immediately, while items from inactive colors are accumulated in the buffer. Once the buffer is full again, a new selection step starts. Note that colors retain their cost from the previous selection step.

▶ **Theorem 14.** *Algorithm 1 achieves a competitive ratio of $O(\ln b/\delta^2)$ in a δ-augmentation setting and a competitive ratio of $O(k^2 \ln b)$ when there is no server augmentation.*

Algorithm 2. Our second algorithm is simply a FIFO algorithm. This algorithm completely ignores the buffer, and serves the items according to their arrival order. Specifically, when an item arrives, it is immediately served by either a server that is already located on the corresponding color, or by moving a server that has not moved the longest to that color.

▶ **Theorem 15.** *There is a deterministic algorithm whose competitive ratio is $O(b/\delta)$ in a δ-augmentation setting and $O(kb)$ when there is no augmentation.*

4 Conclusions

We made a first step in analyzing the generalized reordering buffer management problem in an online setting, and provided non-trivial upper bounds on the competitive ratio. An obvious direction for future research is the design of new algorithms with further improved bounds. By now, both the paging problem and the reordering buffer problem are very well understood. It seems natural to utilize the known techniques and state of the art algorithms for these problems in order to obtain better results for generalized reordering buffer. Nevertheless, it turns out to be a challenging task to establish bounds on such combinations. We do not know whether natural combinations of such algorithms can attain good performance guarantees,

although we have identified several combinations that fail. Any progress in this direction would be of great interest.

References

1 Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000.

2 Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. Almost tight bounds for reordering buffer management. In *STOC*, pages 607–616, 2011.

3 Yuichi Asahiro, Kenichi Kawahara, and Eiji Miyano. Np-hardness of the sorting buffer problem on the uniform metric. In *FCS*, pages 137–143, 2008.

4 Noa Avigdor-Elgrabli and Yuval Rabani. An improved competitive algorithm for reordering buffer management. In *SODA*, pages 13–21, 2010.

5 Noa Avigdor-Elgrabli and Yuval Rabani. A constant factor approximation algorithm for reordering buffer management. In *SODA*, pages 973–984, 2013.

6 Noa Avigdor-Elgrabli and Yuval Rabani. An optimal randomized online algorithm for reordering buffer management. *CoRR*, abs/1303.3386, 2013.

7 Reuven Bar-Yehuda and Jonathan Laserson. Exploiting locality: Approximating sorting buffers. *Journal of Discrete Algorithms*, 5(4):729–738, 2007.

8 Siddharth Barman, Shuchi Chawla, and Seeun Umboh. A bicriteria approximation for the reordering buffer problem. In *ESA*, pages 157–168, 2012.

9 Ho-Leung Chan, Nicole Megow, Rob van Stee, and René Sitters. A note on sorting buffers offline. *Theoretical Computer Science*, 423:11 – 18, 2012.

10 Matthias Englert, Harald Räcke, and Matthias Westermann. Reordering buffers for general metric spaces. *Theory of Computing*, 6(1):27–46, 2010.

11 Matthias Englert and Matthias Westermann. Reordering buffer management for non-uniform cost models. In *ICALP*, pages 627–638, 2005.

12 Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *CoRR*, cs.DS/0205038, 2002.

13 Iftah Gamzu and Danny Segev. Improved online algorithms for the sorting buffer problem on line metrics. *ACM Transactions on Algorithms*, 6(1):15:1–15:14, 2009.

14 Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

15 Rohit Khandekar and Vinayaka Pandit. Online sorting buffers on line. In *STACS*, pages 584–595, 2006.

16 Jens S. Kohrt and Kirk Pruhs. A constant approximation algorithm for sorting buffers. In *LATIN*, pages 193–202, 2004.

17 Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

18 Harald Räcke, Christian Sohler, and Matthias Westermann. Online scheduling for sorting buffers. In *ESA*, pages 820–832, 2002.

19 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

Shapley meets Shapley

Haris Aziz[1] and Bart de Keijzer[2]

1 **NICTA and University of New South Wales, Sydney, Australia**
`haris.aziz@nicta.com.au`
2 **Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands**
`keijzer@cwi.nl`

Abstract

This paper concerns the analysis of the *Shapley value* in *matching games*. Matching games constitute a fundamental class of cooperative games which help understand and model auctions and assignments. In a matching game, the value of a coalition of vertices is the weight of the maximum size matching in the subgraph induced by the coalition. The Shapley value is one of the most important solution concepts in cooperative game theory. After establishing some general insights, we show that the Shapley value of matching games can be computed in polynomial time for some special cases: graphs with maximum degree two, and graphs that have a small modular decomposition into cliques or cocliques (complete k-partite graphs are a notable special case of this). The latter result extends to various other well-known classes of graph-based cooperative games. We continue by showing that computing the Shapley value of unweighted matching games is #P-complete in general. Finally, a fully polynomial-time randomized approximation scheme (FPRAS) is presented. This FPRAS can be considered the best positive result conceivable, in view of the #P-completeness result.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Matching games, Shapley, counting complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.99

1 Introduction

In economics and computer science, one of the most fundamental problems is the allocation of profits or costs based on contributions of the nodes in a network. The problem has assumed even more importance as networks have become ubiquitous. In this paper, we address this problem by simultaneously studying two concepts that can be traced to Lloyd S. Shapley – the *Shapley value* and *matching games*.

Lloyd S. Shapley is one of the most influential game theorists in history. Among his numerous contributions, two of them are the following: (i) formulating the *assignment game* as a rich and versatile class of cooperative games [25], and (ii) proposing the *Shapley value* as a highly desirable solution concept for cooperative games [24]. Both contributions have had far-reaching impact and were part of Shapley's Nobel Prize winning achievements. The assignment game is a cooperative game based on bipartite graphs, and models the interaction between buyers and sellers. It is the *transferable utility* version of the well-known stable marriage setting and is a fundamental model that is used for modelling exchange markets and auctions [23]. Assignment games were later generalized to *matching games*, for *non*-bipartite graphs (see e.g., [11, 17]). The main idea of a matching game is that each node represents an agent and the value of a coalition of nodes is the weight of the maximum weight matching in the subgraph induced by the coalition of nodes. Whereas the matching game is one of the


© Haris Aziz and Bart de Keijzer;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 99–111

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


most natural and important cooperatives game, the Shapley value has been termed "the most important normative payoff division scheme" in cooperative game theory [28]. It is based on the idea that the payoff of an agent should be proportional to his marginal contributions to the payoff for the set of all players. For an excellent overview of the concept, we refer the reader to [21, Chapter 5,]. The Shapley value is the only solution concept that satisfies simultaneously the following properties: efficiency, symmetry, additivity, and dummy player property.

In this paper we address a gap in the computational cooperative game theory literature, and we initiate research on the computational aspects of the Shapley value in matching games. This gap is surprising on two fronts: (i) computational aspects of Shapley values have been extensively studied for a number of cooperative games (see e.g., [12, 15, 14]) and (ii) matching games are a well-established class of cooperative games, and the structure and computational complexity of computing important solution concepts such as the core, least core, and nucleolus have been examined in-depth for matching games (see e.g., [1, 26, 17, 10, 6, 5]).

Our results. We study the algorithmic aspects and computational complexity of the Shapley value for matching games for the first time. We establish first some general insights and some particular special cases for which the exact Shapley value can be computed in polynomial time for: graphs with a constant size decomposition into clique and coclique modules (these include e.g., complete k-partite graphs, for k constant), and for graphs with maximum degree two. The non-trivial algorithm required for graphs of maximum degree two illustrates that exact computation of the Shapley value quickly becomes rather complex, even for very simple graph classes. We then move on to the central results of this paper, which concerns the general problem: we prove that the computational complexity of computing the Shapley value of matching games is #P-complete even if the graph is unweighted. The proof relies on Berge's Lemma and the fact that a certain matrix related to the Pascal triangle has a non-zero determinant. We subsequently present an *FPRAS* (i.e., a *fully polynomial time randomized approximation scheme*) for computing the Shapley value of (weighted) matching games. In view of our #P-completeness result, the FPRAS is the best possible result we can hope for.

Related work. The complexity of computing the Shapley value of important classes of cooperative games has been the topic of detailed studies. The papers [12] and [15] present polynomial-time algorithms to compute the Shapley value of *graph games* and *marginal contribution nets* respectively. On the other hand, computing the Shapley value is known to be intractable for a number of cooperative games (see e.g., [14, 2]).

Among the classes of cooperative games, matching games are one of the most well-studied. The core of matching games is characterized in [11], where it is also shown that various computational problems regarding the core and the least core of matching games can be solved in polynomial time. For matching games, there has been considerable algorithmic research on the *nucleolus*: an alternative single valued solution concept (see e.g., [26, 17]).

As networks analysis becomes an increasingly important area, centrality indices of graphs have received immense interest (see e.g., [7]). The idea is to get a ranking of vertices according to their ability to connect with other vertices. Recently, a Shapley-values based game theoretic approach has been used to gauge the centrality or connectivity of vertices by representing different valuation functions with a graph [20, see e.g.,]. The motivation is that the Shapley value of a vertex captures various synergies which standard centrality measures do not. In this vein, Shapley values of the matching game constitutes an interesting

method of gauging centrality/connectivity of the vertices. In particular it quantifies in a principled manner the ability of a vertex to match with other vertices to increase the value of the coalition.

2 Preliminaries

We work throughout this text with undirected weighted graphs $G = (N, E, w)$, where N is the vertex set, E is the edge set, and $w : E \to \mathbb{R}_{\geq 0}$ is a weight function. For $S \subseteq N$, we denote by $G(S)$ the subgraph of G induced by S, i.e., the graph $(S, \{e \in E : e \in S \times S\})$. We assume for the remainder of this text that the reader is familiar with basic notions related to graphs and matchings.

A *cooperative game* consists of a set N of $n = |N|$ players and a characteristic function $v : 2^N \to \mathbb{R}$ associating a value $v(S)$ to every subset $S \subseteq N$. A subset of N is referred to as a *coalition* in this context. Deciding how to distribute the value $v(N)$ among the players in a fair and stable manner is an objective of central importance in the research area of cooperative games.

A *matching game* is a cooperative game (N, v) induced by an undirected weighted graph $G = (N, E, w)$ (with vertex set N, edge set E, and weight function $w : E \to \mathbb{R}_{\geq 0}$) such that for any $S \subseteq N$, $v(S)$ is the weight of a maximum weight matching of the subgraph $G(S)$. For a given graph G, we will denote by $MG(G)$ the matching game corresponding to graph G.

An *unweighted* matching game is a matching game for which all weights are 1 in the associated graph. In unweighted matching games, it holds that $v(S \cup \{i\}) - v(S) \in \{0, 1\}$ for all $S \subset N$, $i \in N \backslash S$. If, for an unweighted matching game (N, v), a player $i \in N$, and a coalition $S \subseteq N \backslash \{i\}$, it holds that $v(S \cup \{i\}) = v(S) + 1$, then we say that player i is *pivotal* (for coalition S, in game (N, v)). Similarly, if $\sigma : N \to N$ is a permutation on N, and i is pivotal for set of players $p(i, \sigma) = \{j : \sigma^{-1}(j) < \sigma^{-1}(i)\}$ (i.e., the players occurring before i in σ), then we say that σ is pivotal for i.

For the general case of weighted matching games, when S is a coalition not containing player i, we refer to the value $v(S \cup \{i\}) - v(S)$ as *the marginal contribution of i to S*. When σ is a permutation of N, we refer to the value $v(p(i, \sigma) \cup \{i\}) - v(p(i, \sigma))$ as *the marginal contribution of i to σ*.

The *Shapley value* of a player $i \in N$ in a cooperative game (N, v) is denoted by $\varphi_i(N, v)$, and is defined as follows.

$$\varphi_i(N, v) = \kappa_i(N, v)/n!, \qquad \kappa_i(N, v) = \sum_{S \subseteq N \backslash \{i\}} (|S|!)(n - |S| - 1)!(v(S \cup \{i\}) - v(S)). \quad (1)$$

κ_i is called the *raw Shapley value*. It is well-known and straightforward to obtain that the raw Shapley value can be written as $\kappa_i(N, v) = \sum_{\sigma \in S_N} (v(p(i, \sigma) \cup \{i\}) - v(p(i, \sigma),))$, where S_N is the set of permutations on the player set N. For an unweighted matching game, the raw Shapley value of a player is thus equal to the number of pivotal permutations. We refer to the vectors $\varphi = (\varphi_1(N, v), \ldots \varphi_n(N, v))$ and $\kappa = (\kappa_1(N, v), \ldots, \kappa_n(N, v))$ respectively as the Shapley value and the raw Shapley value of the game (N, v).

The players $i, j \in N$ are called *symmetric* in (N, v) if $v(S \cup \{i\}) = v(S \cup \{j\})$ for any coalition $S \subseteq N \setminus \{i, j\}$. A player $i \in N$ is a *dummy* if $v(S \cup \{i\}) - v(S) = 0$ for all $S \subseteq N$. The Shapley value satisfies the following properties: (i) *Efficiency*: $\sum_{i \in N} \varphi_i(N, v) = v(N)$; (ii) *Symmetry*: if $i, j \in N$ are symmetric, then $\varphi_i(N, v) = \varphi_j(N, v)$; (iii) *Dummy*: if i is a dummy, then $\varphi_i(N, v) = 0$; (iv) *Additivity*: $\varphi_i(N, v^1 + v^2) = \varphi_i(N, v_1) + \varphi_i(N, v_2)$ for all

$i \in N$;[1] and (v) *Anonymity*: relabeling the agents does not affect their Shapley value. We are interested in the following computational problem.

Shapley
Instance: A weighted graph $G = (N, E, w)$ and a specified player $i \in N$.
Question: Compute $\varphi_i(MG(G))$.

2.1 General insights

In this subsection, we gain some general insights about the Shapley value of matching games. First, if the graph is not connected, then the problem of computing the Shapley value of the graph reduces to computing the Shapley value of the respective connected components.

▶ **Lemma 1** (Shapley value in connected components). *Let $G = (N, E, w)$ be a weighted graph with k connected components, and let the respective vertex sets of these connected components be $N_1, \ldots, N_k$. Let v be the characteristic function of the matching game $MG(G)$ on that graph, and let $c : N \to [k]$ be the function that maps a vertex i to the number j such that $j \in N_k$.*[2] *Then, for every vertex i it holds that $\varphi_i(v) = \varphi_i(v_{c(i)})$, where v_j denotes the characteristic function of the matching game on the subgraph induced by N_j.*

It is rather straightforward to see that a vertex has a Shapley value zero if and only if it is not connected to any other vertex.

▶ Observation 1. A player in a matching game has a non-zero Shapley value if and only if there is an edge in the graph that contains the player. It can thus be decided in linear time whether a player in a matching game has a Shapley value of zero.

Next, we present another lemma concerning the Shapley value of unweighted matching games.

▶ **Lemma 2.** *Consider an unweighted matching game (N, v). If for each $s \in [n-1]$, the number of coalitions of size s for which player i is pivotal in (N, v) can be computed in time $f(n)$ for some function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$, then the Shapley value of i can be computed in time $nf(n)$.*

3 Exact algorithms for restricted graph classes

Some classes of matching games for which computing the Shapley value is trivial are symmetric graphs (e.g. cliques and cycles), and graphs with a constant number of vertices. We proceed to prove this for two additional special cases: weighted graphs that admit constant size (co)clique modular decompositions, and unweighted graphs with degree at most two.

3.1 Graphs with a constant number of clique or coclique modules

An important concept in the context of undirected graphs is that of a *module*. A subset of vertices $S \subseteq N$ is a module if all members of S have the same set of neighbors in $N \setminus S$. We can extend this notion to weighted graphs by requiring that all members of S are connected

[1] The sum of two characteristic functions v_1 and v_2 on the same player set is defined in the standard way: as $v_1(S) + v_2(S)$ for all $S \subseteq N$.
[2] For $a \in \mathbb{N}$, we write $[a]$ to denote $\{b \in \mathbb{N} : 1 \leq b \leq a\}$.

to the same set of neighbors, by edges of the same weight. A *modular decomposition* is a partition of the vertex set into modules.

A *clique module* (resp. *coclique module*) of a weighted graph is a module of which the vertices are pairwise connected by edges of the same weight (resp. pairwise disconnected). Note that every graph has a trivial modular decomposition into cliques (and cocliques): the partition of N into singletons.

We prove that if a weighted graph G has a size k modular decomposition consisting of only cliques or only cocliques, then the Shapley value of $MG(G)$ can be found in polynomial time. In fact, we will show that this holds for the more general class of *subgraph-based* games: We call a cooperative game (N, v) *subgraph-based* if there exists a weighted graph $G = (N, E, w)$ such that for $S, T \subset N$, it holds that $v(S) = v(T)$ if $G(S)$ and $G(T)$ are isomorphic.

▶ **Theorem 3.** *Consider a subgraph-based cooperative game (N, v). Then, the Shapley value of (N, v) can be computed in polynomial time if the following three conditions hold: i.) the weighted graph $G = (N, E, w)$ associated to (N, v) is given or can be computed in time polynomial in the size of the representation of (N, v); ii.) there exists a modular decomposition $\gamma(G)$ into k cocliques or k cliques and G is unweighted in the latter case; and iii.) $v(S)$ can be computed in polynomial time for all $S \subseteq N$.*

Proof. Note first that one can find for G in polynomial time a minimum cardinality modular decomposition into cocliques: simply check for each pair of vertices whether they are disconnected and connected to identical sets of vertices through edges with identical weights. If so, then they can be put in the same module. Similarly, a minimum cardinality modular decomposition into cliques can be found in polynomial time in case the graph is unweighted, by finding a minimum cardinality modular decomposition into cocliques in the complement of G (i.e., the graph that contains only those edges not in E).

A set of players S is said to be of the same *player type* if all player pairs in S are symmetric. We first show that all players in the same module of $\gamma(G)$ are of the same player type. Let i, j be two players in the same module M in $\gamma(G)$. Then, for every coalition $C \subseteq N \backslash \{i, j\}$, the subgraphs $G(C \cup \{i\})$ and $G(C \cup \{j\})$ are isomorphic (because $G(M)$ is a clique or coclique), so $v(C \cup \{i\}) = v(C \cup \{j\})$. Therefore, we know that the vertices can be divided into a constant number k of player types.

[27] showed that any cooperative game in which the value of a given coalition can be computed in polynomial time, and there is known size k partition of the players into sets of the same player type, then the Shapley value can be computed in polynomial time via dynamic programming. The number of player types in our game is constant number k of clique and coclique modules. Therefore the result of [27] can be applied, and this proves our claim. ◀

For matching games, the function v can be evaluated using any polynomial time maximum weight matching algorithm. Therefore, the above result implies that computing the Shapley value can be done in polynomial time for classes of graphs where we can find efficiently a size k modular decomposition into cliques or cocliques. This includes the class of complete k-partite graphs and any strong product[3] of an arbitrary size clique (or coclique) with a graph on k vertices.

[3] The *strong product* of two graphs $G_1 = (N, E_1)$ and $G_2 = (M, E_2)$ is defined as the graph $(N \times M, E')$, where $E' = \{\{(i_N, i_M), (j_N, j_M)\} \subseteq N \times M : i_M = j_M \wedge \{i_N, j_N\} \in E_1 \vee \{i_M, j_M\} \in E_2\}$.

▶ **Corollary 4.** *For matching games based on complete k-partite graphs, where k is a constant, the Shapley value can be computed in polynomial time.*

Theorem 3 also applies to cooperative games such as s-t vertex connectivity games and min-cost spanning tree games [10, 11], as these are subgraph-based games.

3.2 Graphs of degree at most two

We first examine *linear graphs* (or: "paths"), i.e., unweighted connected graphs in which two vertices have out-degree one and the remaining vertices have out-degree two.

▶ **Lemma 5.** *The Shapley value of a player in a matching game on an unweighted linear graph can be computed in $O(n^4)$ time.*

Proof. Assume without loss of generality that the vertex set is N and the edge set is $\{\{j, j+1\} : j \in N\backslash\{n\}]\}$, and that $i \in N$ is the player of whom we want to compute the Shapley value. Fix any $s \in [n-1]$, and let η_i^s be the number of coalitions of size s for which vertex i is pivotal. We compute η_i^s by subdividing in separate cases and taking the sum of them:

- The number $\eta_i^{s,\text{left}} = |\{S \cup \{i+1\} : S \subseteq N\backslash\{i, i-1, i+1\}, i \text{ is pivotal for } S\}|$. Intuitively: the number of coalitions S where i is pivotal such that adding i to S extends the left of a line segment.
- The number $\eta_i^{s,\text{right}} = |\{S \cup \{i-1\} : S \subseteq N\backslash\{i, i-1, i+1\}, i \text{ is pivotal for } S\}|$.
- The number $\eta_i^{s,\text{connect}} = |\{S \cup \{i-1, i+1\} : S \subseteq N\backslash\{i, i-1, i+1\}, i \text{ is pivotal for } S\}|$. Intuitively: the number of coalitions S where i is pivotal, such that i connects two line segments.
- $\eta_i^{s,\text{isolated}} = |\{S : S \subseteq N\backslash\{i, i-1, i+1\}, i \text{ is pivotal for } S\}|$.

It is immediate that $\eta_i^{s,\text{isolated}} = 0$, since adding i to a coalition S not containing $i+1$ nor $i-1$ results in a coalition forming a subgraph in which i is an isolated vertex. For the remaining three values, $\eta_i^{s,\text{left}}$, $\eta_i^{s,\text{right}}$, and $\eta_i^{s,\text{connect}}$, we show below how to compute them efficiently.

- For $\eta_i^{s,\text{left}}$, observe that adding a vertex to the left of a (non-empty) line segment L increases the cardinality of a maximum matching if and only if L has an even number of edges (and thus an odd number of vertices). Therefore, define $\eta_i^{s,\text{left}}(k)$ to be the number of coalitions S of size s for which i is pivotal such that S contains the line segment $\{i+1, \ldots, i+k+1\}$, and does not contain $\{i-1, i+k+2\}$. The number $\eta_i^{s,\text{left}}(k)$ is easy to determine:

$$\eta_i^{s,\text{left}}(k) = \begin{cases} 0 & \text{if } k \text{ is odd,} \\ \binom{|N\backslash\{i-1,\ldots,i+k+2\}|}{s-|\{i-1,\ldots,i+k+1\}\cap N|} & \text{otherwise.} \end{cases}$$

  We can then express $\eta_i^{s,\text{left}}$ as $\sum_{k=1}^{\max\{n-i-1,s-1\}} \eta_i^{s,\text{left}}(k)$. There is only a linear number of terms in this sum, and all of them can be computed in linear time.
- $\eta_i^{s,\text{right}}$ is computed in an analogous fashion.
- For $\eta_i^{s,\text{connect}}$, observe that adding a vertex i to a coalition such that i connects two line segments L_1 and L_2, increases the cardinality of a maximum matching if and only if L_1 and L_2 do not both have an odd number of edges (or equivalently: not both have an even number of vertices). Therefore, define $\eta_i^{s,\text{connect}}(k_1, k_2)$ to be the number of coalitions S

of size s for which i is pivotal such that S contains the line segments $\{i-k_1-1,\ldots,i-1\}$ and $\{i+1,\ldots,i+k_2+1\}$, and does not contain $\{i-k_1-2, i+k_2+2\}$. The number $\eta_i^{s,\text{connect}}(k_1,k_2)$ is easy to determine:

$$\eta_i^{s,\text{connect}}(k_1,k_2) = \begin{cases} 0 & \text{if } k_1 \text{ and } k_2 \text{ are both odd,} \\ \binom{|N\setminus(\{i-k-2,\ldots,i+k+2\}|}{s-|\{i-k-1,\ldots,i+k+1\}\cap N|} & \text{otherwise.} \end{cases}$$

We can then express $\eta_i^{s,\text{connect}}$ as $\sum_{k_1=1}^{\max\{i-2,s-1\}} \sum_{k_2=1}^{\max\{n-i-1,s-k_1-2\}} \eta_i^{s,\text{left}}(k_1,k_2)$. The number of terms in this sum is quadratic, and all of these terms can be computed in linear time. We can thus compute $\eta_i^{s,\text{connect}}$ in $O(n^3)$ time.

The claim now follows from Lemma 2. ◀

▶ **Theorem 6.** *For graphs with maximum degree 2, the Shapley value can be computed in polynomial time.*

Proof. A graph with degree at most two is a disjoint union of cycles and linear graphs. From Lemma 1, we can compute the Shapley value of the connected components separately. From Lemma 5, we know that the Shapley value of linear graphs can be computed in polynomial time. Due to anonymity, the Shapley value of a cycle is uniform. ◀

The above proof for linear graphs demonstrates that computation of the Shapley value of a matching game already becomes involved for even the simplest of graph structures. We would be interested in seeing an extension of this result that enables us to exactly compute the Shapley value in *any* non-trivial class of graphs that contains a vertex of degree at least three.

4 Computational complexity of the general problem

In this section, we examine the computational complexity of the general problem of computing the Shapley value for matching games. As we mentioned in Section 2, Shapley is equivalent to the problem of counting the number of pivotal permutations for a player in an unweighted matching game, and is therefore a counting problem. It is moreover easy to see that this counting problem is a member of the complexity class #P.[4]

For certain cooperative games such as weighted voting games [14], intractability of computing the Shapley value can be established by proving that even checking whether a player gets non-zero Shapley value is NP-complete. Proposition 1 tells us that this is not the case for matching games. Before we proceed, we establish some notation. Let $G=(N,E)$ be a graph. Let $\alpha_k(G)$ be the number of vertex sets $S \subseteq N$ such that $|S|=k$ and the subgraph $G(S)$ of G induced by S admits a perfect matching. Then $\overline{\alpha_k}(G) = \binom{n}{k} - \alpha_k(G)$ is the number of subsets $S \subseteq N$ of size k such that $G(S)$ does not admit a perfect matching. In order to characterize the complexity of Shapley, we first define the following problem.
#MatchableSubgraphs$_k$
Instance: Undirected and unweighted graph $G=(N,E)$ and an even integer k.
Question: Compute $\alpha_k(G)$.

▶ **Lemma 7.** #MatchableSubgraphs$_k$ *is* #P*-complete.*

[4] Informally: #P is the class of computational problems that correspond to counting the number of accepting paths on a non-deterministic Turing machine. We refer the reader to any introductory text on complexity theory.

Proof. In [9] it is proved that the following problem is #P-complete: Given an undirected and unweighted bipartite graph $G = (S \cup I, E)$, compute the number of subsets of $B \subseteq S$, such that $G(B \cup I)$ admits a perfect matching.[5] The problem is equivalent to #MatchableSubgraphs$_{2|I|}$. ◀

▶ **Theorem 8.** *Computing the Shapley value of a matching game on an unweighted graph is* #P*-complete.*

Proof. We present a polynomial-time Turing reduction from #MatchableSubgraphs$_k$ to Shapley. We show that if there exists a polynomial-time algorithm for Shapley, then we can solve #MatchableSubgraphs$_k$ for a given graph G in polynomial time, by solving Shapley on a set of graphs that we construct from G. For each of these graphs, we show that a linear equation holds that relates the Shapley value of a vertex of G to the values α_k and $\overline{\alpha_k}$. The coefficient matrix of this system of equations will then turn out to be invertible, hence it can be solved in polynomial time via Gaussian elimination in order to compute the values α_k and $\overline{\alpha_k}$.

We remind the reader that the symbol κ is used to denotes the raw Shapley value, as defined in Section 2.

Let $G = (N, E)$ be the given graph, and let G_0 be the graph in which a new vertex y_0 is added to G that is connected to all vertices in N. For $i > 0$, let G_i be G_0 with i additional vertices $y_1, y_2, \ldots, y_i$ and i additional edges $\{\{y_j, y_{j-1}\} : j \in [i]\}$.

The first part of the proof consists of showing that the following set of equations hold:

$$\kappa_{y_i}(MG(G_i)) = \begin{cases} C(i) + \sum_{k=0}^{n}(k+i)!(n-k)!\overline{\alpha_k}(G) & \text{if i is even,} \quad (2) \\ C(i) + \sum_{k=0}^{n}(k+i)!(n-k)!\alpha_k(G) & \text{if i is odd,} \quad (3) \end{cases}$$

where

$$C(i) = \sum_{k=1}^{\lfloor i/2 \rfloor} \sum_{j=0}^{n+i-2k} (j+2k-1)!(n+i-j-2k+1)!\binom{n+i-2k}{j}.$$

Define a *type 1 pivotal coalition for y_i in $MG(G_i)$* as a pivotal coalition for i in $MG(G_i)$ that *does not* contain all players $y_0, \ldots, y_{i-1}$. Define a *type 2 pivotal coalition for y_i in $MG(G_i)$* as a pivotal coalition for y_i in $MG(G_i)$ that *does* contain all players $y_0, \ldots, y_{i-1}$. Denote by $H_i^{\text{type 1}}(s)$ (resp. $H_i^{\text{type 2}}(s)$) the set of type 1 (resp. type 2) pivotal coalitions for i in $MG(G_i)$ that are of size s. From (1), it follows that

$$\kappa_{y_i}(MG(G_i)) = \sum_{s=1}^{n+i} s!(n+i-s)!|H_i^{\text{type 1}}(s)| + \sum_{s=1}^{n+i} s!(n+i-s)!|H_i^{\text{type 2}}(s)|. \quad (4)$$

First we characterize the coalitions in $H_i^{\text{type 2}}(s)$.

▶ **Lemma 9.** *If i is even, a coalition S of $MG(G_i)$ is in $H_i^{type\ 2}(s)$ if and only if $G(S \cap N)$ is not perfectly matchable (and $\{y_0, \ldots, y_{i-1}\} \subseteq S, |S| = s$). If i is odd, a coalition S of $MG(G_i)$ is in $H_i^{type\ 2}(s)$ if and only if $G(S \cap N)$ is perfectly matchable (and $\{y_0, \ldots, y_{i-1}\} \subseteq S, |S| = s$).*

[5] The proof of Colbourn resolved "an exceptionally difficult problem" [9]. Interestingly, the corresponding decision problem of checking whether there exists a subgraph of size k that does not admit a perfect matching, appears to be open.

Proof. *Case of even i.* ($\Rightarrow$) Let M be a maximum matching for $G_i(S)$. S is pivotal for y_i, so M is not a perfect matching. We can assume though, that all vertices $\{y_0, \ldots, y_{i-1}\}$ are matched to each other in the matched graph $(G_i(S), M)$, because $G_i(\{y_0, \ldots, y_{i-1}\})$ is a linear graph with an even number of vertices, and is thus perfectly matchable. It follows that the exposed nodes of $(G_i(S), M)$ are all in N, and therefore the matching M restricted to N is a maximum matching for $G(S \backslash \{y_0, \ldots, y_{i-1}\}) = G(S \cap N)$ that is non-perfect.

($\Leftarrow$) Let M be a maximum (non-perfect) matching for $G(S \cap N)$ and let y be an exposed vertex of $(G(S \cap N), M)$. Then $M' = M \cup \{\{y_j, y_{j+1}\} : j \text{ even} \wedge j < i\}$ is a maximum matching for $G_i(S)$, by Berge's Lemma, as it is clear that there is no augmenting path in $(G_i(S), M')$. Moreover, observe that in $(G_i(S), M')$ there is an even-length alternating path from y to y_{i-1}. Therefore, there is in (G_i, M') an augmenting path from y to y_i, and it follows again by Berge's lemma that S is pivotal.

Case of odd i. ($\Rightarrow$) Let M' be a maximum matching for $G_i(S)$. S is pivotal, so in $(G_i(S), M')$ there is an even-length alternating path P from an exposed node y to y_{i-1}. Obtain the matching M by augmenting M' along P. M is then a maximum matching for $G_i(S)$ in which y_{i-1} is exposed. $G_i(\{y_0, \ldots, y_{i-1}\})$ is a linear graph and M is maximum, so it follows that y_{i-1} is the only exposed node in $(G_i(S), M)$ among $\{y_0, \ldots y_{i-1}\}$. Therefore $S \cap N$ must be matched to each other in $(G(S), M)$ (for otherwise, in $(G_i(S), M)$ there would be an augmenting path from y_{i-1} to an exposed node of $S \cap N$, contradicting the fact that M is a maximum matching for $G_i(S)$). It follows that $G(S \cap N)$ is perfectly matchable.

($\Leftarrow$) Let M be a maximum perfect matching for $G(S \cap N)$. Let M' be a maximum matching for $G_i(\{y_0, \ldots, y_{i-1}\})$ in which y_{i-1} is the only exposed node. Then $M \cup M'$ is a matching for $G_i(S)$ in which y_{i-1} is the only exposed node. $M \cup M'$ is clearly a maximum matching, and in $(G_i, M \cup M')$ the edge $\{y_{i-1}, y_i\}$ is exposed. So S is pivotal. ◀

From the above lemma, it follows that the coalitions in $H_i^{\text{type 2}}(s)$ are precisely the coalitions of the form $T \cup \{y_0, \ldots, y_{i-1}\}$, where $T \subset N$ is such that for even i, $G(T)$ is not perfectly matchable, and for odd i, $G(T)$ is perfectly matchable. Therefore $|H_i^{\text{type 2}}(s)| = \overline{\alpha_{s-i}}(G)$ for even i and $|H_i^{\text{type 2}}(s)| = \alpha_{s-i}(G)$ for odd i, and this implies:

$$\sum_{s=1}^{n+i} s!(n+i-s)!|H_i^{\text{type 2}}(s)| = \begin{cases} \sum_{k=0}^{n}(k+i)!(n-k)!\overline{\alpha_k}(G) & \text{if } i \text{ is even,} \\ \sum_{k=0}^{n}(k+i)!(n-k)!\alpha_k(G) & \text{if } i \text{ is odd.} \end{cases}$$

In words: the second summation of (4) equals the summation of (2) when i is even, and the summation of (3) when i is odd. Therefore, it suffices to prove that the first summation of (4) equals $C(i)$.

For this sake, define $H_i^{\text{type 1}}(s,k)$ for $k \in [\lfloor i/2 \rfloor]$ as $\{S \in H_i^{\text{type 1}}(s) : y_{i-2k} \notin S \wedge \{y_{i-1}, \ldots, y_{i-2k+1}\} \subseteq S\}$. Observe that $\{H_i^{\text{type 1}}(s,1), \ldots, H_i^{\text{type 1}}(s, i/2)\}$ is a partition of $H_i^{\text{type 1}}(s)$. For a given k and s, note that the set $H_i^{\text{type 1}}(s,k)$ consists of all coalitions of the form $T \cup \{y_{i-1}, \ldots, y_{i-2k+1}\}$, where $T \subseteq N \cup \{y_0, \ldots, y_{i-2k-1}\}$, $|T| = s - 2k + 1$. Hence, $|H_i^{\text{type 1}}(s,k)| = \binom{n+i-2k}{s-2k+1}$ (defining $\binom{a}{b} = 0$ whenever $b < 0$ or $b > a$). Therefore:

$$\begin{aligned} \sum_{s=1}^{n+i} s!(n+i-s)!|H_i^{\text{type 1}}(s)| &= \sum_{k=1}^{\lfloor i/2 \rfloor} \sum_{s=2k-1}^{n+i-1} s!(n+i-s)!\binom{n+i-2k}{s-2k+1} \\ &= \sum_{k=1}^{\lfloor i/2 \rfloor} \sum_{j=0}^{n+i-2k} (j+2k-1)!(n+i-j-2k+1)!\binom{n+i-2k}{j}. \end{aligned}$$

This shows that (2) and (3) hold.

The second part of the proof consists of showing that all $\alpha_k(G), k \in N$ can be computed from $\kappa_{y_i}(MG(G_i))$ in polynomial time, using (2) and (3), for $i \in N \cup \{0\}$. This is sufficient to complete the proof, because the graphs $G_0, \ldots, G_n$ can clearly be constructed from G in polynomial time, hence a polynomial time algorithm that computes α_k from $\kappa_{y_i}(MG(G_i)), i \in N$ is a polynomial Turing reduction.

Let $\beta_i(G) = \alpha_i(G)$ for even i and let $\beta_i(G) = \overline{\alpha_i}(G)$ for odd i. We can represent (2) and (3) for $i \in N \cup \{0\}$ as the following system of equations:

$$\begin{pmatrix} 0!n! & 1!(n-1)! & \cdots & n!0! \\ 1!n! & & \cdots & (n+1)!0! \\ \vdots & \vdots & \ddots & \vdots \\ n!n! & & \cdots & (2n)!0! \end{pmatrix} \times \begin{pmatrix} \beta_0(G) \\ \beta_1(G) \\ \vdots \\ \beta_n(G) \end{pmatrix} = \begin{pmatrix} \kappa_{y_0}(MG(G_0)) - C(0) \\ \kappa_{y_1}(MG(G_1)) - C(1) \\ \vdots \\ \kappa_{y_n}(MG(G_n)) - C(n) \end{pmatrix} \tag{5}$$

Denote by A the $(n+1) \times (n+1)$ matrix in the above equation. Recall that a scalar multiplication of a column by a constant c multiplies the determinant by c. Therefore, A is nonsingular if and only if nonsingularity also holds for the $(n+1) \times (n+1)$ matrix B, defined by $B_{ij} = (i+j)!$. B is a matrix that is related to Pascal's triangle, and it is known that its determinant is equal to $\prod_{i=0}^{n} i!^2 \neq 0$ [3, 2]. It follows that A is nonsingular, so our system of equations (5) is linearly independent and has a unique solution. Note that all entries in the system can be computed in polynomial time (assuming that the Shapley value of a matching game is polynomial time computable): The constants $C(i)$ consist of polynomially many terms, and all factorials and binomial coefficients that occur in (5) are taken over numbers of magnitude polynomial in n.

Therefore, we can use Gaussian elimination to solve (5) in $O(n^3)$ time. It follows that for all $i \in N$, $\beta_i(G)$ can be computed in polynomial time, and hence $\alpha_i(G)$ can be computed in polynomial time. Therefore, if there exists an algorithm that solves Shapley in polynomial time, then it can also be used to solve #MatchableSubgraphs$_k$ in polynomial time. ◀

5 An approximation algorithm

In this section, we show that although computing exactly the Shapley value of matching games is a hard problem, approximating it is much easier.

Let Σ be a finite alphabet in which we agree to describe our problem instances and solutions. A *fully polynomial time randomized approximation scheme (FPRAS)* for a function $f : \Sigma^* \to \mathbb{Q}$ is an algorithm that takes input $x \in \Sigma^*$ and a parameter $\epsilon \in \mathbb{Q}_{>0}$, and returns with probability at least $\frac{3}{4}$ a number in between $f(x)/(1+\epsilon)$ and $(1+\epsilon)f(x)$. Moreover, an FPRAS is required to run in time polynomial in the size of x and $1/\epsilon$. The probability of $\frac{3}{4}$ is chosen arbitrarily: by a standard amplification technique, it can be replaced by an arbitrary number $\delta \in (1/2, 1)$. The resulting algorithm would then run in time polynomial in $n, 1/\epsilon$, and $\log(1/\delta)$.

We will now formulate an algorithm that approximates the raw Shapley value of a player in a weighted matching game, and show that it is an FPRAS. Note that we cannot utilize approximation results in [18] and [4] since matching games are neither convex nor simple. Our FPRAS is based on Monte Carlo sampling, and works as follows: Let $(G = (N, E, w), i, \epsilon)$ be the input, where G is the weighted graph representing matching game $MG(G)$, $i \in N$ is a player in $MG(G)$, and ϵ is the precision parameter. For notational convenience, we write κ_i as a shorthand for $\kappa_i(MG(G))$. The algorithm first determines whether $\kappa_i = 0$ (Observation 1). If so, then it outputs 0 and terminates. If not, then it samples $\lceil 4n^2(n-1)^2/\epsilon^2 \rceil$ permutations

of the player set uniformly at random. Denote this multiset of sampled permutations by P. The algorithm then outputs the average marginal contribution of player i over the permutations in P and terminates. Note that this average marginal contribution is efficiently computable: it is given by $1/\lceil 4n^2(n-1)^2/\epsilon^2 \rceil$ times the sum of the marginal contributions of player i to each of the sampled permutations. Determining these marginal contributions can be done in polynomial time, using any maximum weight matching algorithm. Denote our sampling algorithm by MatchingGame-Sampler.

MatchingGame-Sampler resembles the algorithms in [19, 18]: the differences are that the algorithm takes a different number of samples, and that it determines whether the Shapley value of player i is 0 prior to running the sampling procedure. Moreover, its proof of correctness requires different insights.[6]

▶ **Theorem 10.** MatchingGame-Sampler *is an FPRAS for the raw Shapley value in a weighted matching game.*

Proof. Denote by $\bar{\kappa}_i$ the output of the algorithm. If $\kappa_i = 0$, then MatchingGame-Sampler is guaranteed to output the right solution, so assume that $\kappa_i > 0$. Let $w_i^{\max}$ be the maximum weight among the edges attached to i, and let $e_i^{\max} \in E$ be an edge that is attached to i such that $w(e_i^{\max}) = w_i^{\max}$. Let X be a random variable that takes the value of $n!$ times the marginal contribution of player i in a uniformly randomly sampled permutation of the players. Note that $\mathbf{E}[X] = \kappa_i$. Note that the marginal contribution of a player in any permutation is at most $w_i^{\max}$, so X is at most $w_i^{\max} n!$.

Let j be the neighbor of i connected by $e_i^{\max}$. Observe that any permutation in which j is positioned first, and i is positioned second, is a permutation for i in which the marginal contribution of i is $w_i^{\max}$. There are $(n-2)!$ such permutations, so the raw Shapley value κ_i of i is at least $w_i^{\max}(n-2)!$. For the variance of X we obtain $\mathbf{Var}[X] = \mathbf{E}[X^2] - \mathbf{E}[X]^2 \leq \mathbf{E}[X^2] \leq (w_i^{\max})^2 n!^2 \leq n^2(n-1)^2\kappa_i^2$.

Observe that $\bar{\kappa}_i$ is a random variable that is equal to $\frac{\sum_{j=1}^{\lceil 4n^2(n-1)^2/\epsilon^2 \rceil} X_j}{\lceil 4n^2(n-1)^2/\epsilon^2 \rceil}$, where X_j are independent random variables with the same distribution as X. From this we obtain that $\mathbf{E}[\bar{\kappa}_i] = \mathbf{E}[X] = \kappa_i$. The desired approximation guarantee then follows from Chebyshev's inequality,[7] and completes the proof:

$$\begin{aligned}\mathbf{Pr}[|\bar{\kappa}_i - \kappa_i| \geq \epsilon\kappa_i] &\leq \frac{\mathbf{Var}[\bar{\kappa}_i]}{\epsilon^2\kappa_i^2} = \frac{\mathbf{Var}\left[\frac{1}{\lceil 4n^2(n-1)^2/\epsilon^2 \rceil}\sum_{j=1}^{\lceil 4n^2(n-1)^2/\epsilon^2 \rceil} X_j\right]}{\epsilon^2\kappa_i^2} \\ &= \frac{\left(\frac{\mathbf{Var}[X]}{\lceil 4n^2(n-1)^2/\epsilon^2 \rceil}\right)}{\epsilon^2\kappa_i^2} \leq \frac{n^2(n-1)^2\kappa_i^2}{(4n^2(n-1)^2/\epsilon^2)\cdot\epsilon^2\kappa_i^2} \leq \frac{1}{4}.\end{aligned}$$

◀

▶ **Corollary 11.** *The algorithm that runs* MatchingGame-Sampler *and returns its output scaled down by* $1/n!$*, is an FPRAS for the Shapley value of a weighted matching game.*

Observe that MatchingGame-Sampler is an FPRAS in the strong sense that its running time does not depend on the weights of the edges. Due to the #P-completeness result stated in Theorem 7, this FPRAS is the best one can hope for, and provides us with a complete answer to the precise complexity of this problem (based on our best judgment).

[6] To be precise, this applies only to [18]. For the sampling algorithm in [19], no proof or approximation-quality analysis of any kind is given.

[7] Here, one could also choose to apply Hoeffding's inequality instead of Chebyshev's inequality, but this will not result in an asymptotically better bound.

6 Conclusions

In this paper, we examined the structure, algorithms, and computational complexity for the problem of computing the Shapley value in a matching game. There are many special cases of the problem that have not been treated in this paper, but nonetheless are potentially worthwhile to analyze: trees, bipartite graphs, connected regular graphs, and series-parallel graphs. Among these, bipartite graphs are especially interesting, since they model two-sided markets. There are some interesting computational problems related to Shapley value computation, such as the problem of comparing the Shapley values of two vertices. One may also pursue the same questions for *fractional matching games* in which the value of a coalition is the maximum size of a fractional matching [8]. Moreover, our study motivates the investigation of unexplored connections with some objects in matching theory. The matching polytope is one of the most-well studied objects in polyhedral combinatorics [22]. It will be interesting to identify any relation between the matching polytope of a graph and the Shapley values of the corresponding matching game. Secondly, network flows are fundamentally connected to matchings for the case of bipartite graphs. An interesting research direction is to explore the connection of network flow games [16] with matching games and whether computing Shapley values of one game is reducible to computing Shapley values of the other game, under certain conditions.

Acknowledgements. The authors thank Ross Kang for various helpful discussions.

References

1 A. Alkan and D. Gale. The core of the matching game. *Games and Economic Behavior*, 2(3):203–212, 1990.

2 H. Aziz, O. Lachish, M. Paterson, and R. Savani. Power indices in spanning connectivity games. In *Proc. of 5th International Conference on Algorithmic Aspects in Information and Management (AAIM)*, volume 5564 of *LNCS*, pages 55–67. Springer, 2009.

3 R. Bacher. Determinants of matrices related to the Pascal triangle. *Journal de théorie des nombres de Bordeaux*, 14:19–41, 2002.

4 Y. Bachrach, E. Markakis, E. Resnick, A. D. Procaccia, J. S. Rosenschein, and A. Saberi. Approximating power indices: theoretical and empirical analysis. *Autonomous Agents and Multi-Agent Systems*, 20:105–122, 2010.

5 P. Biró, M. Bornhoff, P. A. Golovach, W. Kern, and D. Paulusma. Solutions for the stable roommates problem with payments. *Theoretical Computer Science*, 2013.

6 P. Biró, W. Kern, and D. Paulusma. Computing solutions for matching games. *International Journal of Game Theory*, 41(1):75–90, 2011.

7 U. Brandes and T. Erlebach, editors. *Network Analysis*, volume 3418 of *LNCS*. Springer, 2005.

8 N. Chen, P. Lu, and H. Zhang. Computing the nucleolus of matching, cover and clique games. In *Proc. of 26th AAAI Conference*, 2012.

9 C. J. Colbourn, J. S. Provan, and D. Vertigan. The complexity of computing the Tutte polynomial on transversal matroids. *Combinatorica*, 15(1):1–10, 1995.

10 X. Deng and Z. Fang. Algorithmic cooperative game theory. In A. Chinchuluun, P. M. Pardalos, A. Migdalas, and L. Pitsoulis, editors, *Pareto Optimality, Game Theory And Equilibria*, volume 17 of *Springer Optimization and Its Applications*. Springer-Verlag, 2008.

11 X. Deng, T. Ibaraki, and H. Nagamochi. Algorithmic aspects of the core of combinatorial optimization games. *Mathematics of Operations Research*, 24(3):751–766, 1999.

12 X. Deng and C. H. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 12(2):257–266, 1994.

13 J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

14 E. Elkind, L. A. Goldberg, P. W. Goldberg, and M. Wooldridge. On the computational complexity of weighted voting games. *Annals of Mathematics and Artificial Intelligence*, 56(2):109–131, 2009.

15 S. Ieong and Y. Shoham. Marginal contribution nets: A compact representation scheme for coalitional games. In *Proc. of 6th ACM-EC Conference*, pages 193–202. ACM Press, 2005.

16 E. Kalai and E. Zemel. Generalized network problems yielding totally balanced games. *Operations Research*, 30:998–1008, 1982.

17 W. Kern and D. Paulusma. Matching games: The least core and the nucleolus. *Mathematics of Operations Research*, 28(2):294–308, 2003.

18 D. Liben-Nowell, A. Sharp, T. Wexle, and K. Woods. Computing shapley value in supermodular coalitional games. In *Proc. of 18thCOCOON*, 2011.

19 I. Mann and L. S. Shapley. Values of large games, iv: Evaluating the electoral college by montecarlo techniques. Technical Report RM-2651, RAND Corporation, 1960.

20 T. P. Michalak, K. V. Aadithya, P. L. Szczepanski, B. Ravindran, and N. R. Jennings. Efficient computation of the shapley value for game-theoretic network centrality. *Journal of Artificial Intelligence Research*, 46:607–650, 2013.

21 H. Moulin. *Fair Division and Collective Welfare*. The MIT Press, 2003.

22 M. D. Plummer. Matching theory - a sampler: from Dénes König to the present. *Discrete Mathematics*, 100:177–219, 1992.

23 A. Roth and M. A. O. Sotomayor. *Two-Sided Matching: A Study in Game Theoretic Modelling and Analysis*. Cambridge University Press, 1990.

24 L. S. Shapley. A value for n-person games. *Annals of Math Studies*, 28:307–317, 1953.

25 L. S. Shapley and M. Shubik. The Assignment Game I: The Core. *International Journal of Game Theory*, 1:111–130, 1972.

26 T. Solymosi and T. E. S. Raghavan. An algorithm for finding the nucleolus of assignment games. *International Journal of Game Theory*, 23:119–143, 1994.

27 S. Ueda, M. Kitaki, A. Iwasaki, and M. Yokoo. Concise characteristic function representations in coalitional games based on agent types. In T. Walsh, editor, *Proc. of 22nd IJCAI*, pages 393–399. AAAI Press, 2011.

28 E. Winter. The Shapley value. In *Handbook of Game Theory with Economic Applications*, chapter 53, pages 2025–2054. Elsevier, 2002.

Complexity classes on spatially periodic Cellular Automata

Nicolas Bacquey

GREYC – Université de Caen Basse-Normandie / ENSICAEN / CNRS, Caen, France, `nicolas.bacquey@unicaen.fr`

Abstract

This article deals with cellular automata (CA) working over periodic configurations, as opposed to standard CA, where the initial configuration is bounded by persistent symbols. We study the capabilities of language recognition and computation of functions over such automata, as well as the complexity classes they define over languages and functions. We show that these new complexity classes coincide with the standard ones starting from polynomial time. As a by-product, we present a CA that solves a somehow relaxed version of the density classification problem.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes

Keywords and phrases Language recognition, Cyclic languages, Computable functions, Algorithms on Cellular Automata, Linear space, Polynomial time, Density classification problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.112

1 Introduction

Spatially periodic computation on cellular automata is a natural and well-defined notion (see e.g. [5]), though it seems it has not been studied extensively. This paper deals with algorithms and computational complexity of cellular automata (CA) acting on spatially periodic configurations, or, equivalently, on ring-cellular automata, i.e., CA whose underlying structure is a ring. The input of a ring-CA is a circular word, that is a finite word defined up to shift (around the ring). Clearly, a language recognized by a ring-CA is a cyclic language (as defined in [1], [2]), i.e., a language which is closed under shift, power and root. To our knowledge, our results are the first ones to deal with computational complexity on such automata.

A natural problem is the following, denoted MINIMAL-PERIOD: Given a spatially periodic configuration, compute its lexicographically minimal period, or, equivalently, given an input word w around a ring, compute its *canonical* root u, i.e., the lexicographically minimal word u such that $w = u^p$ up to shift, for some (maximal) integer p. We exhibit an algorithm on a ring-CA that computes the MINIMAL-PERIOD problem in polynomial time. This basic result allows us to compare computational complexity of ring-CA with complexity of standard CA whose input word is bounded by persistent symbols. Informally, we prove that the complexity classes on ring-CA and standard CA coincide down to polynomial time complexity. More precisely, we prove the following equivalences, for any cyclic language L:

- L is recognizable on a ring-CA iff L is LINSPACE;
- L is recognizable in polynomial time on a ring-CA iff L is recognized by a standard CA in linear space and polynomial time.


© Nicolas Bacquey;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 112–124

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


Moreover, we naturally extend those complexity results to functions, and use these results to show that the density classification problem (a well-studied problem defined in [6]) can be solved if we can use more states than the input alphabet $\{0, 1\}$.

2 Definitions and context

2.1 The computational model

We will use along this article the standard definition of cellular automata (CA) as a tuple $\mathcal{A} = (d, Q, N, \delta)$ (see [5]). In these lines, we will work with $d = 1$, *i.e.* cellular automata of dimension 1, so the underlying network will be $\mathbb{Z}$. Q denotes the set of *states*, and $N = \{-1, 0, 1\}$ is the *standard neighbourhood*. The local transition function of the automaton is denoted by $\delta : Q^{\{-1,0,1\}} \to Q$. As we work with cellular automata from the point of view of *language recognition*, we will identify a particular $\Sigma \subseteq Q$ as the *input alphabet*. We also introduce the *global transition function* $F_\delta : Q^{\mathbb{Z}} \to Q^{\mathbb{Z}}$ defined by the global synchronous application of δ over configurations of $\mathbb{Z}$ by $\forall C \in Q^{\mathbb{Z}}\ (F_\delta(C))(i) = \delta(C(i-1), C(i), C(i+1)))$.

We suppose that the reader is familiar with the notions of signals and computation layers on cellular automata. If it is not the case, we strongly encourage the reading of [5] or [7] for such general matters on cellular automata.

▶ **Definition 1.** We define a *Ring-Cellular Automaton (ring-CA)* as a cellular automaton whose initial configuration (and therefore any subsequent configuration) is periodic. Note that this model is equivalent to an automaton that would work on a finite, ring-like cell network. Let $u \in \Sigma^*$, we denote $C_u \in Q^{\mathbb{Z}}$ the bi-infinite repetition of u ($C_u = \dots uuu \dots$).

2.2 Recognition on ring-CA

On a standard CA where the input word is bounded by persistent states, it is easy to identify a particular cell of the configuration (e.g. the first one). We say that a word is *accepted* (or *rejected*) when this particular cell enters a persistent acceptance (or rejection) state (see [11] for all subjects related to language recognition on CA). However, if the configuration we are working on is periodic, the identification of a particular cell is intrinsically impossible. Therefore, we must define the acceptance or rejection of a word as a global phenomenon. Here are the two definitions of acceptance that we will use on ring-CAs:

▶ **Definition 2.** We say that a language $L \subset \Sigma^*$ is *ring-recognizable* if there exists a ring-CA $\mathcal{C}$ such that for all $u \in \Sigma^*$, the automaton $\mathcal{C}$ given the periodic configuration C_u as an input evolves in such a way that for some time t:

- *weak recognition:* All cells enter the same particular subset of states, either $S_a \subset Q$ (for accept) or $S_r \subset Q$ (for reject) and never leave it afterwards.
- *strong recognition:* All cells enter the same particular state (S_a and S_r are singletons), with the transition function defined so that this configuration is a fixed point of the global transition function F_δ.

Note that our definition of strong recognition is the best recognition we can hope for on a ring-CA, due to its intrinsically spatially periodic nature. Also note that due to that very nature, for any language that is weakly or strongly recognized, there exists a time exponentially bounded in the length of the period after which all the cells are in their definitive subset of states (S_a or S_r).

While it is obvious that any strongly recognizable language is weakly recognizable (actually by the same automaton), we will show that those two definitions are actually equivalent in section 4.

▶ **Definition 3.** We introduce the *shift function* denoted as $\sigma : \Sigma^* \to \Sigma^*$ and defined for all $u = u_1u_2...u_n \in \Sigma^*$ by $\sigma(u) = u_2...u_nu_1$.

Since one cannot discriminate between configurations produced by a word, and those produced by shifts, powers or roots of this word, every language recognized by a ring-CA must be closed under shift, power and root operations. More formally, we can only recognize *cyclic* languages defined as follows:

▶ **Definition 4.** A language $L \subset \Sigma^*$ is said to be *cyclic* (see [2]) if $\forall w \in \Sigma^*, \forall k \geq 1$:
- $w \in L$ iff $\sigma^k(w) \in L$,
- $w \in L$ iff $w^k \in L$.

We say that a cyclic language L is *strongly (resp. weakly) ring-recognizable* if there exists a ring-CA that strongly (resp. weakly) recognizes it.

2.3 Computability of functions on ring-CA

We can also design our ring-CAs to compute functions, as an extension of language recognition, which is a particular function with $\{0, 1\}$ as its output set. Intuitively, we want to give the input of the function as a periodic configuration of the CA, wait some time, and read the result of the function when the CA has reached a fixed point. We now give the following formal definition of a computable function:

▶ **Definition 5.** Let $f : \Sigma^* \to \Gamma^*$ be a function over words of Σ^*.
f is said to be *ring-computable* if there exists a *ring-CA* $\mathcal{A}$ such that $\forall u \in \Sigma^*$:
- there exists an integer t such that $F_\delta^t(C_u) = C_{f(u)}$,
- $F_\delta(C_{f(u)}) = C_{f(u)}$ (i.e. $C_{f(u)}$ is a fixed point of F_δ).

We define the time complexity of the computation of such a function f on a ring-CA $\mathcal{C}$ over a word u as the smallest t such that $C_{f(u)} = F_\delta^t(C_u)$.

We define an analogous to cyclic languages in the case of functions. As we read the output of the function on the automaton where the word was input, we need an additional condition on the length of the output of those functions ($\|w\|$ is the length of the word w):

▶ **Definition 6.** A function $f : \Sigma^* \to \Gamma^*$ is said to be *cyclic* if $\forall u \in \Sigma^*, \forall k \geq 1$:
- $f(\sigma^k(u)) = \sigma^k(f(u))$,
- $f(u^k) = f(u)^k$,
- $\|f(u)\| = \|u\|$.

2.4 Complexity classes and results

▶ **Definition 7.** Due to the circular nature of the model, we define the *size* n of an input as the length of its minimal period.

We note that, as a consequence of the fact that our work space can be seen as a finite ring, we can't use more than a linear space for our computations on ring-CAs, with respect to the input size. Therefore, every language or function we study must be in $LINSPACE$, which is a robust complexity class. Then we give the following definitions:

▶ **Definition 8.** We denote the complexity class of languages that are strongly recognizable in polynomial time on a ring-CA as $TIME_{ring}(POLY(n))$. We also denote the complexity class of languages that are recognizable in linear space and polynomial time on a bounded input CA (or equivalently, classical models such as Turing machines, RAM machines...) as $SPACETIME(n, POLY(n))$. We will abusively use this notation to talk about complexity classes of functions.

We will prove our results along these lines:

▶ **Theorem 9 (Recognition of languages).** *Let L be a cyclic language, then:*
- *L is strongly ring-recognizable $\iff$ $L \in LINSPACE$,*
- *$L \in TIME_{ring}(POLY(n)) \iff L \in SPACETIME(n, POLY(n))$.*

▶ **Theorem 10 (Computability of functions).** *Let f be a cyclic function, then:*
- *f is ring-computable $\iff$ $f \in LINSPACE$,*
- *$f \in TIME_{ring}(POLY(n)) \iff f \in SPACETIME(n, POLY(n))$.*

3 From the cyclic model to the standard model

Throughout the rest of the paper, $L \subset \Sigma^*$ will denote a cyclic language. In this section, we will prove that L is ring-recognizable implies $L \in LINSPACE$.

This part of the proof is quite straightforward: Indeed, if we consider a ring-automaton $\mathcal{C}$ that weakly recognizes L, it is easy to design a standard cellular automaton $\mathcal{A}$ that recognizes it: it suffices to add an additional layer to $\mathcal{C}$, in which $\mathcal{A}$ will write the mirrored input word in a preprocessing phase. The automaton will now simulate two computations of $\mathcal{C}$ in parallel, connecting the beginnings and ends of the two simulated configurations (see Fig 1). After this linear time preprocessing, $\mathcal{A}$ will simulate $\mathcal{C}$ step-by-step.

Though, one must be careful when defining the halting conditions of $\mathcal{A}$. Indeed, there can be a state where all simulated cells of $\mathcal{C}$ are in the "accept" or "reject" subset of states, but the computation has not ended yet. Therefore, $\mathcal{A}$ also has to construct the exponential time bound before which we are sure that the computation of $\mathcal{C}$ is over, then decide if the word is accepted or rejected by checking the state of an arbitrary cell of its array (by construction, all cells are in the same "accept" or "reject" subset of states at that moment).

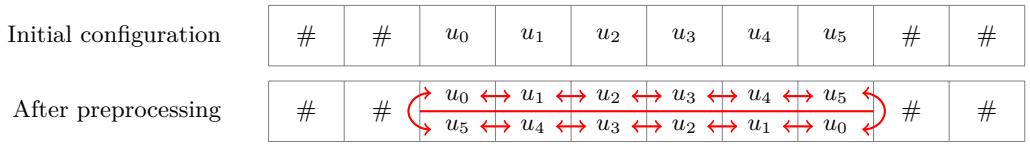


Figure 1 Simulating a ring-CA on a standard CA.

For the proof that $L \in TIME_{ring}(POLY(n))$ implies $L \in SPACETIME(n, POLY(n))$, it suffices to construct the polynomial time bound before which the computation of $\mathcal{C}$ is over instead of the exponential one, and the construction still holds.

4 From the standard model to the cyclic model

In this section, we will prove that $L \in LINSPACE$ implies L is ring-recognizable.

Let $\mathcal{A}$ be a standard CA recognizing $L \subset \Sigma^*$. We will design a ring-CA $\mathcal{C}$ that will mimic the computation of $\mathcal{A}$. If we denote as Q the work alphabet of $\mathcal{A}$, our automaton $\mathcal{C}$ will use

a new set of states $Q' = Q \times \Sigma \times \omega$, where ω will handle all the specific constructions of $\mathcal{C}$. We also introduce a bijection $val : \Sigma \cup \{\vdash\} \to [0, |\Sigma|]$ such that $val(\vdash) = 0$. We want each cell of $\mathcal{C}$ to retain its input letter, which means that the Σ part of Q' will never change. All our computation will be done through two computational layers, namely $Q \times \omega$.

4.1 Global vision

We will now present a mechanism that is able to find a minimal period of every periodic configuration it is started on. Note that this minimal period is defined up to a shift. We will combine this mechanism to a simulation of $\mathcal{A}$ over the word contained in this minimal period.

The periodic configuration of $\mathcal{C}$ will be divided into *intervals*, in which we will simulate the computation of $\mathcal{A}$ over the word contained in the interval. We design those intervals so that two different adjacent intervals will merge over time.

For a better understanding of the algorithm, we can imagine that each interval can be in three states, namely "*merge-to-the-right*", "*merge-to-the-left*" and "*waiting*". Those states can evolve over time, according to the following rule:

▶ Rule 11. If an interval is in the *merge-to-the-right* state and its right neighbour is in the *merge-to-the-left* state, then they must merge together, and it is the only way for intervals to merge.

When an interval is created, we use the Q layer to simulate the computation that $\mathcal{A}$ would have done over the word contained in the interval. When the computation is over, and if the interval has not merged yet, a special success/failure state of Q is propagated over all the cells of the interval.

At the beginning, each cell will define its own interval, and those intervals will start to merge as the computation goes. We state that intervals will merge until every two adjacent intervals are equal (see fig 2 for an intuition of the fusion process).

Once this configuration where every interval contains the same word is reached, the simulation of $\mathcal{A}$ in this interval and the propagation of the success/failure state will properly match our first recognition condition (i.e. all cells enter the same particular subset of states and never leave it).

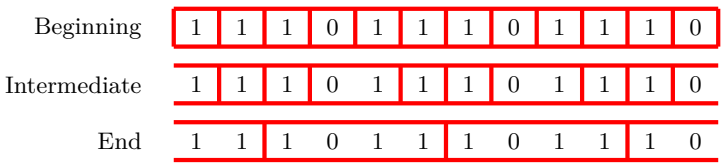


Figure 2 Outline of the merging process on a cyclic configuration.

4.2 Basic tools

Intervals: Let us identify a specific sub-layer of our work alphabet ω, which is of the form $\omega = \omega' \times \{\#, \varnothing\}$; We define the *intervals* of a configuration as the maximum sets of adjacent cells beginning with a $\#$ and containing exactly one $\#$. The size of an interval I is the number of cells it contains, denoted by $size(I)$ (see Fig 3).

We define the *content* of an interval as the word formed by the concatenation of the canonical projection of the states of its cells over Σ, preceded by the special symbol "$\vdash$". We say that two intervals are *different* if their contents are different words.



Figure 3 Intervals of size 2, 5 and 3.

We will construct a method ensuring the following property:

▶ **Lemma 12.** *There exists an integer C such that, if two adjacent intervals have different contents at time t, then at least one of them will merge before time $t + C \times n^3$, where n is the size of the larger of both intervals.*

Signals and pointers: Every interval will have a signal (as defined in, e.g. [8]) going back and forth in it, starting from its left border, and will also keep a pointer over the letters of its content (see Fig 4). Each time the signal will enter a cell containing the pointer from its right, it will move the pointer one letter to the right (for this purpose, we suppose that the first "$\vdash$" is stored in the first cell of the interval). When the pointer is at the rightmost letter of the content, its next move brings it back to "$\vdash$" (in n time steps).

At the first time step, a # is written on each cell, so that every cell will form an interval of size 1, a signal starts in each interval, and all the pointers are set to the first letter of the two-letter content of each interval, namely "$\vdash$". For an interval of size n, let a_i denote the symbol currently pointed, with $i \in [0, n]$ and $a_0 =$ "$\vdash$".

Let $k = |\Sigma| + 1$. The signal will wait $t_i = kn^2 + 2n \times val(a_i)$ time steps on each border, then move at speed ± 1 to the other border. These times (t_i) will encode the content of the interval, thus allowing an interval to compare itself with its neighbours, and merging if necessary. Note that by standard techniques (see [8]), the function $n \mapsto kn^2 + 2n \times val(a_i)$ is easily time-constructible using properly defined signals carrying $val(a_i)$ in their state, thus allowing use to wait such times on the borders.

We say that an interval is in the *merge-to-the-left* (*resp. merge-to-the-right*) state of our global vision if its signal is on the left border (*resp.* right border), and in the *waiting* state otherwise.

4.3 Merging process

Let us consider what happens when two adjacent intervals I_1 and I_2 have their signals meet on their common border. It ensues from our previous definitions that I_1 is in the *merge-to-the-right* state, and I_2 is in the *merge-to-the-left* state. Therefore according to rule 11, a merging of I_1 and I_2 must occur. We do so by erasing the # symbol defining their common border, and destroying the signals. Then new signals are sent to the beginning and the end of the new interval. Those two signals will reset the pointer to the beginning of the new word, and reset the simulation of the computation of $\mathcal{A}$. Finally, the whole process starts again (see fig 5).

Proof of lemma 12. Let I_1 and I_2 be two adjacent intervals with different contents a and b, and S_1 and S_2 be their respective signals. Let I_1 be the interval on the left and I_2 be the one on the right. Let $size(I_1) = m$ and $size(I_2) = n$. It suffices to consider the case where neither I_1 nor I_2 merge with other intervals during the time spans we consider. We will prove our lemma by considering two different cases, whether I_1 and I_2 have different sizes or not.

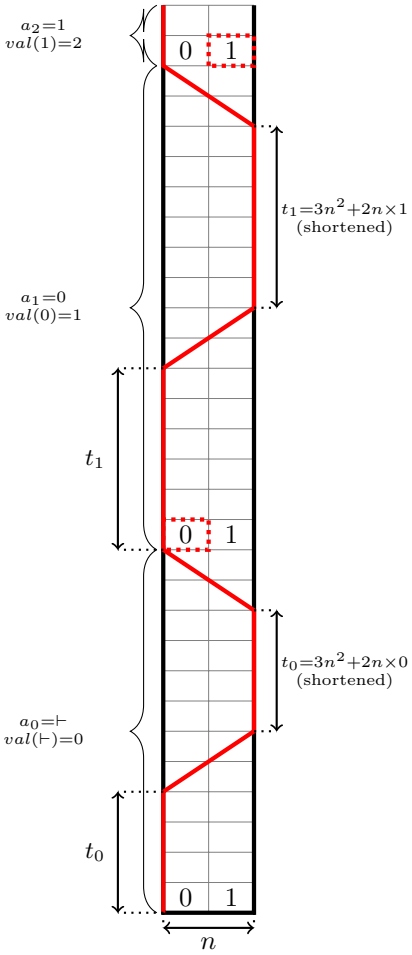


Figure 4 Basic move of a signal in an interval (with $|\Sigma| = 2$). The dashed square figures the pointer, which is set to "⊢" at the beginning and until further notice.

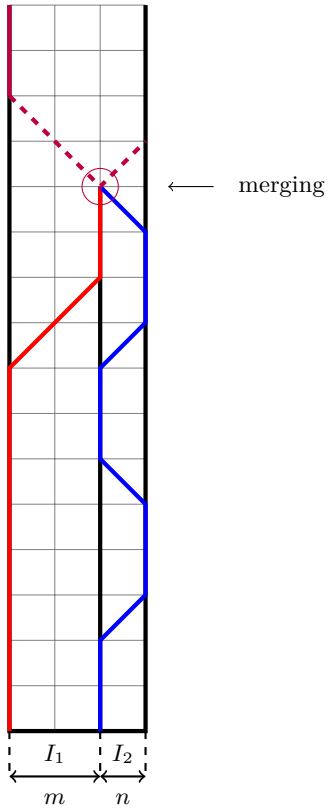


Figure 5 The merging process – different sizes

Merging intervals of different sizes

We suppose without loss of generality that $m > n$, and we will prove that the time interval when S_1 is on the common border cannot be contained in the time interval when S_2 is away from that border. The signal S_1 will wait at least $T_1 = km^2$ on each border, in particular on the border between I_1 and I_2. Now S_2 will be away from a border during at most $T_2 = kn^2 + 2(k-1)n + 2n = kn^2 + 2kn$ consecutive time steps (since the journey back and forth from a border lasts $2n$ time steps, and $val(a_i) \leq k-1$).

Since $m \geq n+1$, we have $T_1 \geq k \times (n+1)^2 = kn^2 + 2kn + k = T_2 + k > T_2$. $T_1 > T_2$ means that S_2 cannot be away from the common border long enough not to meet S_1, which means that I_1 and I_2 will merge eventually.

Complexity analysis: It is easily seen that I_1 and I_2 will merge together in time $O(max(n,m)^2)$, because the signal of the larger of both intervals cannot achieve a complete trip back and forth without encountering the signal from the smaller one, and such a trip takes a time $O(max(n,m)^2)$. *A fortiori*, there exists an integer c such that I_1 and I_2 will merge before time $t + c \times max(n,m)^3$, which proves Lemma 12 for the case $n \neq m$.

Merging intervals of equal sizes and different contents

We now consider the case where two adjacent intervals have the same size n, but different contents. We will show that they will merge when their pointer will be set on different symbols. Let us start by defining some useful notions.

Asynchronicity: During the lifetime of I_1, we consider the case where its signal S_1 does one or more round trips between its borders (the contrary would mean that I_1 has merged before, see Fig 6). Let t_1 be any time when S_1 reaches the left border. S_1 comes from the right border, where it has waited during a certain time interval T_0. Note that by construction, we must have $T_0 \geq 2n$, because $kn^2 \geq 2n$.

I_1 has not merged during the time interval T_0. That means that S_2 must have been away from the left border of I_2 during that time. There are two cases: either S_2 was on the right border, or it was travelling through I_2. Note that S_2 can't have been travelling more than $2n-2$ time steps, because that would mean it would have encountered the left border. As $T_0 > 2n-2$, this means that there must be a time t_1' during T_0 when S_2 was on the right border of I_2. Let us now define t_2 as the first time after t_1' when S_2 will reach the left border of I_2.

▶ **Definition 13.** As we have defined a specific time for each interval, we can now define the *asynchronicity* between them, as $\delta = t_1 - t_2$.

We note that this asynchronicity can be defined each time S_1 reaches the left border of I_1. Those times t_1 and t_2 are special in our construction, as they are the times when the intervals move their pointer one cell to the right (or move it back to the first symbol of their content). Thus, we can associate a pair of symbols $(a_i, b_j) \in \Sigma \cup \{\vdash\}$ to each pair of times (t_1, t_2) that define an asynchronicity (these symbols are the ones which are newly pointed by I_1 and I_2 respectively). Now let us consider the boundaries of the asynchronicity δ: If $a_{i-1} = b_{j-1}$, we must have $-n < \delta < n$. Indeed, the contrary would mean that S_1 and S_2 would have met before t_1, and a merging would have occurred (see Fig 7 for the absurd cases where $a_{i-1} = b_{j-1}$ and $\delta \leq -n$ (7a) or $\delta \geq n$ (7b)).

Let us prove that the sequences (a_j) and (b_j) cannot be the same if $a \neq b$. Indeed, the contrary would mean that (a_j) and (b_j) would coincide on every symbol, including the only "$\vdash$" a and b contains, and therefore any subsequent symbol until the next "$\vdash$". This would mean that $a = b$, hence a contradiction (let us recall that "$\vdash$" marks the beginning of the content).

We consider the first pair of times (t_1, t_2) when the associated symbols (a_i, b_j) are different (it exists, since we have $a \neq b$). If we note δ the associated asynchronicity, we have $|\delta| < n$, because $a_{i-1} = b_{j-1}$. We state that I_1 and I_2 will merge before their signals can go back and forth. We will identify two cases, whether $a_i < b_j$ or not (See Fig 8).

We will first see what happens if $a_i < b_j$ (See Fig 8a). S_1 will wait a time $t_0 = kn^2 + 2n \times val(a_i)$ on each border, and S_2 will wait $t_0 + \Delta_t$, with $\Delta_t = 2n \times (val(b_j) - val(a_i))$. Since $a_i < b_j$, we must have $val(b_j) > val(a_i)$, and therefore $\Delta_t \geq 2n$. Now S_1 will arrive on the right border of I_1 at time $t_1 + t_0 + n$, and S_2 will stay on this border from time $t_1 + \delta$ to time $t_1 + \delta + t_0 + \Delta_t$. Let us prove that $t_0 + n \in [\delta, \delta + t_0 + \Delta_t]$, which means that the two signals will meet on the common border.

Since $t_0 > 0$ and $|\delta| < n$, we have $t_0 + n > \delta$. $\Delta_t \geq 2n$, so $\delta + \Delta_t > n$, and therefore $t_0 + n < \delta + t_0 + \Delta_t$.

If $a_i > b_j$ (See Fig 8b), the merging occurs later, but for similar arguments. t_0 is now defined as the waiting time of S_2, and Δ_t as $2n \times (val(a_i) - val(b_j))$. We will have to prove

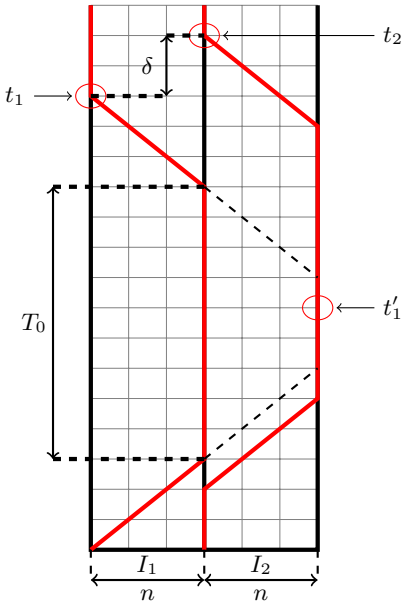
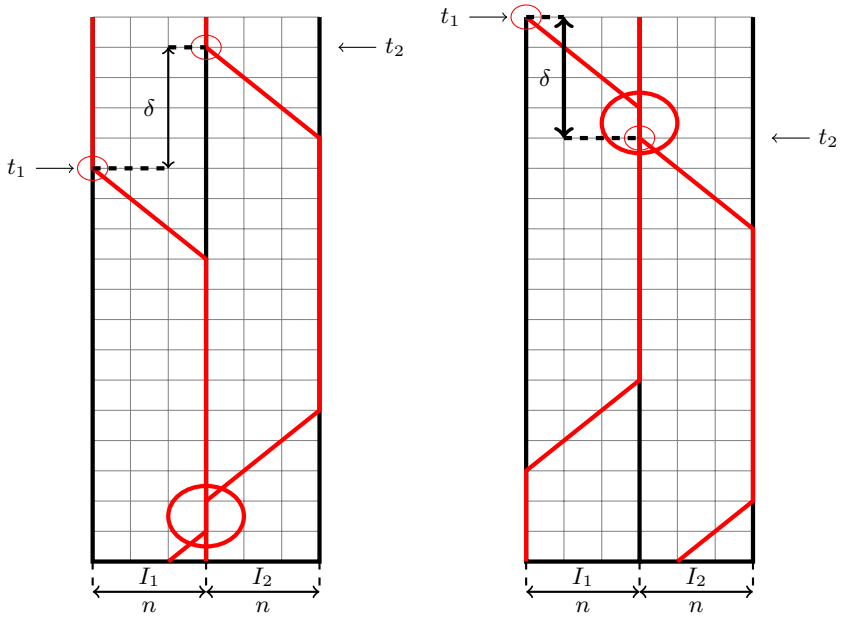


Figure 6 Definition of asynchronicity between I_1 and I_2 at time (t_1, t_2).

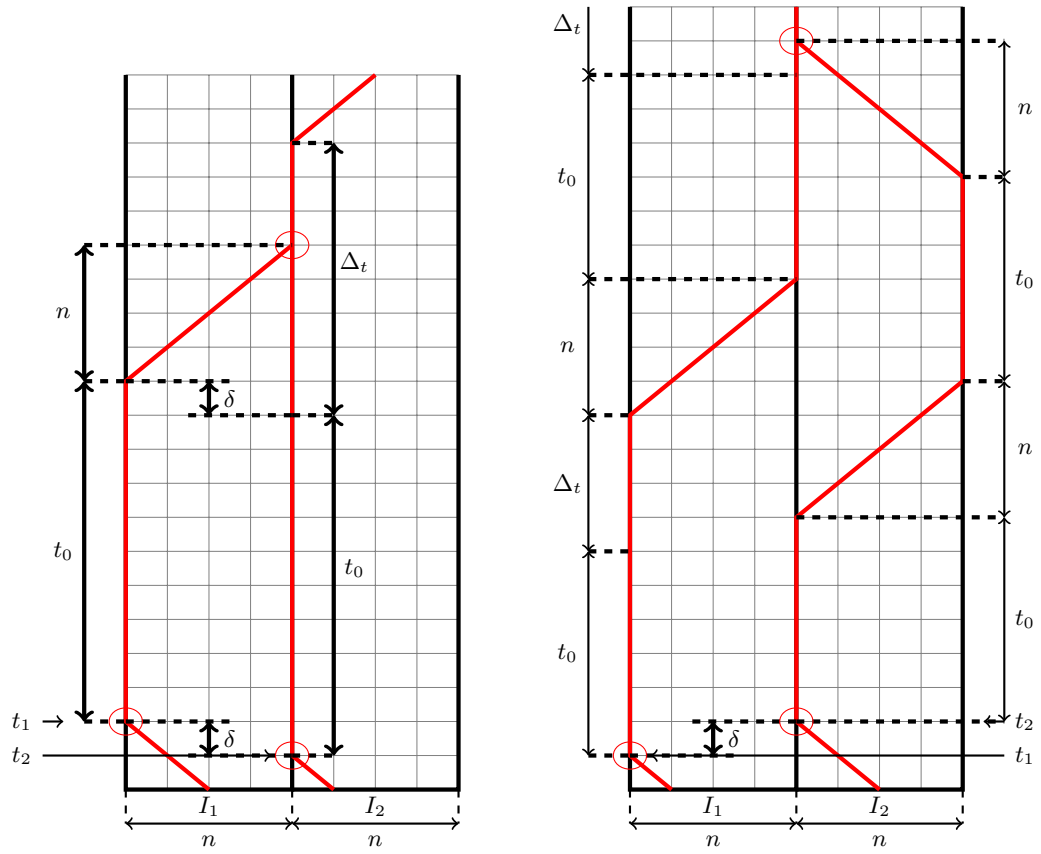


(a) if $\delta \leq -n$ **(b)** if $\delta \geq n$

Figure 7 Boundaries of asynchronicity.

that $2t_0 + 2n + \delta \in [t_0 + \Delta_t + n, 2t_0 + 2\Delta_t + n]$, using all former remarks plus the fact that $\Delta_t \leq 2n^2$. The complete proof is left to the reader (see Fig 8b for an intuition).

Complexity analysis: Once again, let (t_1, t_2) be the first pair of times when their associated symbols (a_i, b_j) are different. We claim that there exists a c such that I_1 and I_2 will merge



(a) $a_i < b_j$ **(b)** $a_i > b_j$

Figure 8 The merging process – intervals of equal sizes.

before time $t_1 + c \times n^2$ (or $t_2 + c \times n^2$ equivalently, since $|t_1 - t_2| < n$). Indeed, it is true because both signals cannot complete a trip back and forth from t_1 (resp. t_2). Now let us see how much time happens before such a pair of times (t_1, t_2) is encountered. The pointers over the contents of I_1 and I_2 change with delay $O(n^2)$ by construction. Therefore, the signals S_1 and S_2 have a period of $O(n^3)$, which means that they must encounter in time $O(n^3)$, thus proving the last remaining case of Lemma 12. ◀

Now let us recall that at the beginning, each period of our workspace is divided into n intervals of size 1. Lemma 12 states that while there exists two adjacent different intervals at least one of them must merge before time $O(n^3)$. Therefore, the number of intervals in a single period of the workspace must decrease every $O(n^3)$ time steps until all intervals are equal, which means there exists only one interval per period. At this point, which happens after $O(n^4)$ time steps, it suffices to wait for the end of the simulation of $\mathcal{A}$ in each interval for the ring-CA $\mathcal{C}$ to enter in a loop in which no merging can occur. The acceptance or rejection of the input can be then decided by projection of the states of $\mathcal{C}$ over Q. This behaviour matches our definition of weak recognition, therefore L is weakly ring-recognizable.

The proof remains the very same if $L \in SPACETIME(n, POLY(n))$ and we want to prove $L \in TIME_{ring}(POLY(n))$. Indeed, our construction only adds an $O(n^4)$ time before the simulation of $\mathcal{A}$ decides in polynomial time whether or not the input belongs to L.

4.4 Strengthening the construction to achieve strong recognition

We will now add a few enhancements to our construction, so that the ring-CA $\mathcal{C}$ will strongly recognize a language if the underlying standard automaton $\mathcal{A}$ recognizes it. The first thing we have to do is to add two states encoding the acceptance or rejection of a word to our work alphabet Q' (we need this to cope with our definition of *strong recognition*). Our new work alphabet is now: $Q' = (Q \times \Sigma \times \omega) \cup \{accept, reject\}$ The naive way to achieve strong recognition is to put the cells of an interval in the *accept* or *reject* state when a local simulation of $\mathcal{A}$ is over. This leads to a crucial issue, which will be explained in the following paragraphs.

False positives: The main issue that arises when we enforce the *accept* or *reject* states into the cells instead of letting the signals work is the risk of false acceptance (or rejection): one can imagine a case where an interval decides that its word is accepted, then overwrites itself with the *accept* state, whereas there are other intervals in the configuration, whose contents are different from itself and thus need to merge with it, leading to an error. Even if we somehow manage to reconstruct the interval afterwards, its initial content (a word from Σ^*) is lost. The following mechanism solves this problem.

Saving private content: Regarding the previous paragraphs, an interval should not overwrite its content with *accept* or *reject* until we are sure that its initial input can be found elsewhere, and then restored if necessary. We will design a mechanism that will ensure that an interval overwrites itself only if the content of its left neighbour is the same as its proper content. Our Lemma 12 ensures that if any pair of adjacent intervals I_1 and I_2 do not merge during a certain time $c \times n^3$, then they have the same content. We want the intervals to detect those cases that will eventually happen, and only to overwrite themselves when they are assured that their left neighbour has the same content as themselves. As a consequence of Lemma 12, there exists a constructible time t_{eq} such that if two adjacent intervals evolve together during a time t_{eq}, then their content is equal. We will exploit that property by adding a "timer" layer to the intervals that will wait for time t_{eq} and check if the left border of the interval is periodically visited by the signal of its left neighbour (this ensures that its left neighbour has not merged yet; One can check this by leaving a "token" when a signal reaches the right border of its interval). We redesign the overwriting process so that it can only happen when the interval has waited for at least t_{eq} alongside its left neighbour. This ensures that an interval only overwrites itself when it is sure that its content can be restored later.

Restoring lost content: As the right neighbour of an interval may have overwritten itself, we must ensure that each time the content of an interval changes (i.e. when it merges), its former content is sent to its right to replace the *accept* or *reject* states. We can copy the content of an interval by the method detailed on Fig 9. When an interval gets its old content back by this method, it immediately creates a new signal, begins its computation as a newly created interval, and checks if its right neighbour should also be restored.

5 Extensions

Towards function computing: We suppose that we are given a standard CA $\mathcal{A}$ that computes a cyclic function f. A few minor tweaks to the mechanism that gave us strong recognition are enough to obtain a ring-CA that computes the function f. Indeed, instead

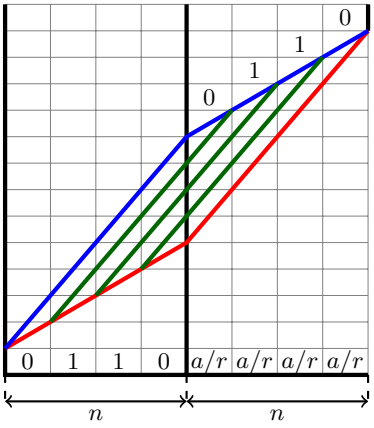


Figure 9 An intuition of the copying mechanism (a/r means accept/reject).

of adding the $\{accept, reject\}$ set of states to the automaton, we will add Γ, the output alphabet of the function f. When an interval was supposed to overwrite itself with the *accept* or *reject* state, it writes the output of f over its content instead (it has enough room to do so, as $\|f(u)\| = \|u\|$). *Ceteris paribus*, our new automaton exactly computes f, therefore f is ring-computable.

Density classification problem: We consider the density classification problem, as defined in [6] and studied in [3], [4]. The goal of this problem is to design a CA that works on periodic configurations on the alphabet $\{0, 1\}$ and converges towards the bi-infinite configuration composed only of 1 (denoted as $1^{\mathbb{Z}}$) if there are more 1s than 0s in the initial configuration and $0^{\mathbb{Z}}$ otherwise. This can be seen as the computation of a specific function, which happens to be cyclic. Therefore, our construction provides a solution to the open problem of the density classification in deterministic case, by using more states than the sole input alphabet.

6 Conclusion and open problems

We note that our method computes an optimal leader election on a spatially periodic configuration: the leaders are the cells where the # finally remain. We can shift those symbols in such a way that they delimit the lexicographically minimal word, thus solving in polynomial time the MINIMAL-PERIOD problem defined in the introduction.

Whereas spatially periodic CA have been studied in the framework of dynamic systems, e.g. for proving that a given automaton is injective or surjective (see [5], [9]), very little work has been done to our knowledge in the framework of language recognition or computability. Several cyclic languages can be suggested from this article: the majority languages, which consist of the languages where a specific symbol appears more than the other ones, or the cyclic closure of regular languages. However, it is difficult to describe how comprehensive and interesting cyclic languages can be. It is also natural to extend spatially periodic computation problems to 2-dimensional cellular automata. As usual, it raises several issues, including the very definition of a somehow "minimal pattern" of a bi-periodic infinite picture, and its computability on a cellular automaton. Thus, in the 2-dimensional case the definition of cyclic languages and cyclic functions fitting our framework is unclear, and is a fine food for thought for future works.

References

1 Marie-Pierre Béal, Olivier Carton, and Christophe Reutenauer. Cyclic languages and strongly cyclic languages. In *STACS*, volume 1046 of *LCNS*, pages 49–59, Berlin, 1996. Springer.

2 Olivier Carton. A hierarchy of cyclic languages. *ITA*, 31(4):355–369, 1997.

3 Nazim Fatès. Stochastic cellular automata solve the density classification problem with an arbitrary precision. In *STACS*, volume 9 of *LIPIcs*, pages 284–295, 2011.

4 Henryk Fúks. Solution of the density classification problem with two cellular automata rules. *Phys. Rev. E*, 55:R2081–R2084, 1997.

5 Jarkko Kari. Basic concepts of cellular automata. In Rozenberg et al. [10], pages 3–24.

6 Mark WS Land and Richard K Belew. No two-state CA for density classification exists. *Physical Review Letters*, 74(25):5148–5150, 1995.

7 Jacques Mazoyer. Computations on one-dimensional cellular automata. *Annals of Mathematics and Artificial Intelligence*, 16(1):285–309, 1996.

8 Jacques Mazoyer and Véronique Terrier. Signals in one-dimensional cellular automata. *TCS*, 217(1):53–80, 1999.

9 John Myhill. The converse of Moore's garden-of-eden theorem. In *Proceedings of the American Mathematical Society*, volume 14, pages 658–686, 1963.

10 Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors. *Handbook of Natural Computing*. Springer, 2012.

11 Véronique Terrier. Language recognition by cellular automata. In Rozenberg et al. [10], pages 124–158.

Asymmetry of the Kolmogorov complexity of online predicting odd and even bits

Bruno Bauwens

Université de Lorraine, LORIA, Vandœuvre-lès-Nancy, France
Brbauwens@gmail.com

Abstract

Symmetry of information states that $C(x) + C(y|x) = C(x, y) + O(\log C(x))$. In [3] an online variant of Kolmogorov complexity is introduced and we show that a similar relation does not hold. Let the even (online Kolmogorov) complexity of an n-bitstring $x_1x_2 \dots x_n$ be the length of a shortest program that computes x_2 on input x_1, computes x_4 on input $x_1x_2x_3$, etc; and similar for odd complexity. We show that for all n there exists an n-bit x such that both odd and even complexity are almost as large as the Kolmogorov complexity of the whole string. Moreover, flipping odd and even bits to obtain a sequence $x_2x_1x_4x_3 \dots$, decreases the sum of odd and even complexity to $C(x)$. Our result is related to the problem of inferrence of causality in timeseries.

1998 ACM Subject Classification E.4 Coding and Information Theory

Keywords and phrases (On-line) Kolmogorov complexity, (On-line) Algorithmic Probability, Philosophy of Causality, Information Transfer

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.125

1 Introduction

Imagine two people want to perform a two-person theater play. First suppose that the play consists of only two independent monologues each one performed by one player. Before performing, the players must memorize their part of the play, and the total studying effort for the two players together can be assumed to be equal to the effort for one person to study the whole script.

Now imagine a play consisting of a large dialogue where both players alternate lines. Each player only needs to study their half of the lines, and it is sufficient to remember each line only after hearing the last lines of the other player. Thus each player needs only to remember their incremental amount of information in his lines, and this suggests the total studying effort might be close to the effort for one person to study the whole script.

However, it often happens that after studying only his own lines, an actor can reproduce the whole piece. Sometimes actors just study the whole piece. This suggests that studying each half of the lines can be as hard as studying everything. In other words, the total effort of both players together might be close to twice the effort of studying the full manuscript.

Can we interpret this example in terms of Shannon information theory? In the first case, let a theater play be modeled by a probability density function $P(X, Y)$ where X and Y represent the two monologues. Symmetry of information states that $H(X) + H(Y|X) = H(X, Y)$, i.e. the information in the first part plus the new information in the second part equals the total information. This equality is exact and can be extended to the interactive case where a similar additivity property remains valid, and this contrasts to the story above.

An absolute measure of information in a string is given by its Kolmogorov complexity, which is the minimal length of a program on a universal Turing machine that prints the string.



© Bruno Bauwens;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 125–136



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



See section 2 for formal definitions. Symmetry of information for Kolmogorov complexity holds within logarithmic terms [19, 1]: $C(x) + C(y|x) = C(x, y) + O(\log C(x, y))$.

For the interactive case, we need the online variant of Kolmogorov complexity introduced in [3]. Let $C_{\text{ev}}(x)$ denote the length of a shortest program that computes x_2 on input x_1, computes x_4 on input $x_1x_2x_3$, etc.; and similar for $C_{\text{odd}}(x)$. In the above example all x_i with odd i correspond to lines for the first player and the others to the second.

In Theorem 1, we show that there exist infinitely many bitstrings x, such that both $C_{\text{ev}}(x)$ and $C_{\text{odd}}(x)$ are almost as big as $C(x)$, in agreement with our example. In Theorem 2, we show that there exists $c > 0$ such that $(C_{\text{ev}} + C_{\text{odd}} - C)(x) \geq c|x|$, i.e. the online asymmetry of information can be large compared to the length of x. Finally, we raise the question how large $(C_{\text{ev}} + C_{\text{odd}} - C)(x)$ can be in terms of $|x|$. A more direct upper bound is $|x|/2 + O(1)$, and one can raise the question whether this is tight. We show there exists a smaller one: there exists $c > 0$ such that $(C_{\text{ev}} + C_{\text{odd}} - C)(x) \leq (1/2 - c)|x|$ for all large x.

Our main result is stronger and is related to the problem of defining causality in time series. Imagine there exists a complex system (e.g. a brain) and we make some measurements in two parts of it. The measurements are represented by bitstrings x (from some part X of the brain) and y (from some part Y). We perform these measurements regularly and get a sequence of pairs

$$(x_1, y_1), (x_2, y_2), \ldots$$

We assume that both parts are communicating with each other, however, the time resolution is not enough to decide whether y_i is a reply to x_i or vice versa. However, we might compare the *dialogue complexity* $C_{\text{odd}} + C_{\text{ev}}$ of

$$x_1, y_1, x_2, y_2, \ldots$$

and

$$y_1, x_1, y_2, x_2, \ldots$$

and (following Occam's Razor principle) choose an ordering that makes the dialogue complexity minimal. We show that these complexities can differ substantially.

Questions of causality are often raised in neurology and economics. The notions of Granger causality and information transfer reflect the idea of "influence" and our result implies a theoretical notion of asymmetry of influence that does not need to assume a time delay to "transport" information between X and Y in contrast to existing definitions [6, 7, 15, 11].[1]

To understand why (current) practical algorithms need a time delay to make inferences about the direction of influence, consider two variables X, Y with a joint probability density function $P(X, Y)$. Using Shannon entropy, we can quantify the influence of X upon Y as $I(Y; X) = H(Y) - H(Y|X)$. Symmetry of information directly implies that this equals the influence of Y upon X: $H(X) - H(X|Y) = H(X) + H(Y) - H(X, Y)$. In the online setting, mutual information is replaced by information transfer, which is well studied in the engineering literature [4, 15, 10, 14, 18, 11, 13]. For time delays k and $l > k$ the information transfer from $\mathcal{X}$ to $\mathcal{Y}$ is given by

$$H(Y_n|Y_{n-l}, \ldots, Y_{n-1}) - H(Y_n|Y_{n-l}, \ldots, Y_{n-1}, X_{n-l}, \ldots X_{n-k}),$$

[1] In the case of three or more timeseries there exist algorithms that infer directed information flows between some variables in some special cases where enough conditional independence exist among the variables, see [12, p. 19–20, 50]. In our example no independence is assumed.

(if this term is dependent on n, the sum is taken). This quantification of causality coincides with Granger causality [6, 7] if all involved conditional distributions are Gaussian.

If we incorporate a time delay $k \geq 1$, the information transfers from $\mathcal{X}$ to $\mathcal{Y}$ and $\mathcal{Y}$ to $\mathcal{X}$ can be different. On the other hand, for $k = 0$ they are always equal, and this is a corollary of (the conditional version of) symmetry of information. In the offline case, a similar observation holds for algorithmic mutual information: $C(x) - C(x|y) = C(y) - C(y|x) + O(\log C(x, y))$.[2] In the online setting, algorithmic mutual information can be generalized to algorithmic information transfer. For an n-bit x and y the version without time delay is given by

$$IT(x \to y) = C(y) - C_{\text{ev}}(x_1 y_1 \dots x_n y_n)\,.$$

We show that for all $\epsilon > 0$ there are infinitely many pairs (x, y) with $|x| = |y|$ and $C(x, y) \geq \Omega(|x|)$ such that $IT(x \to y) \leq \epsilon C(x, y)$ while $IT(y \to x)$ exceeds $C(x, y) + O(1)$. Hence, in contrast to Shannon information theory, significant online dependence of x_i on y_i might not imply significant online dependence of y_i on x_i.

Warning: The example where influence (and causality) is asymmetric heavily uses that shortest models are not computable. Decompression algorithms used in practice are always total (or can be extended to total ones). On the other hand, if one wants to be practical, it is natural to not only consider total algorithms but algorithms that terminate within some reasonable time bound (say polynomial). On that level non-symmetry may reappear, even for one pair of messages, which was not possible in our setting. For example suppose x_1 represents a pair of large primes and y_1 represents their product, then it is much easier to produce first x_1 and then y_1 then vice versa.

Muchnik paradox is a result about online randomness [9] that is related to our observations. Consider the example from [3]: in a tournament (say chess), a coin toss decides which player starts the next game. Consider the sequence $b_1, w_1, b_2, w_2, \dots$ of coin tosses and winners of subsequent games. This sequence might not be random (the winner might depend on who starts), but we would be surprised if the coin tossing depends on previous winners.

More precisely, a sequence is Martin-Löf random if no lower semicomputable martingale succeeds on it. To define randomness for even bits, we consider martingales that only bet on even bits, i.e. a martingale F satisfies $F(x0) = F(x1)$ if $|x0|$ is odd. The even bits of ω are *online random* if no lower semicomputable martingale succeeds that only bets on even bits. (In our example, coin tosses b_i are unfair if a betting scheme makes us win on $b_1 w_1 b_2 w_2 \dots$ while keeping the capital constant for "bets" on w_i.) In a similar way randomness for odd bits is defined. Muchnik showed that there exists a non-random sequence for which both odd and even bits are online random. Hence, contributed information by the odd and even bits does not "add up". Muchnik's paradox does not hold for the online version of computable randomness (where martingales are restricted to computable ones), and is an artefact of the non-computability of the considered martingales.

The article is organised as follows: the next section presents definitions and results. The subsequent three sections are devoted to the proofs: first theorems are reformulated using online semimeasures, and then lower bounds are proven. In the full version of the paper, which is available on ArXiv, there are four appendices containing: a proof of the chain rule

[2] However, logarithmic deviations can appear, if one considers prefix complexity, for example if y is chosen to be a string consisting of $K(x)$ zeros. In this case, it is known that for each n there exist n-bit x such that $K(K(x)) - K(K(x)|x) \leq O(1)$ while $K(x) - K(x|K(x)) \geq \log n - O(\log \log n)$. Moreover, this small error was exploited in an earlier and more involved proof of Theorem 2 [2].

for online complexity, the generalization of Theorem 1 for online computation with more machines, a version of Theorem 2 with a larger linear constant, and a full proof of the upper bound (Theorem 3).

2 Definitions and results

Kolmogorov complexity of a string x on an optimal machine U is the minimal length of a program that computes x and halts. More precisely, associate with a Turing machine a function U that maps pairs of strings to strings. The conditional Kolmogorov complexity is given by

$$C_U(x|y) = \min\{|p| : U(p,y) = x\} .$$

This definition depends on U, but there exist a class of machines for which $C_U(x|y)$ is minimal within an additive constant for all x and y. We fix such an optimal U, and drop this index, see [8, 5] for details. If y is the empty string, we write $C(x)$ in stead of $C(x|y)$, and the complexity of a pair $C(x,y|z)$ is given by applying an injective computable pairing function to x and y.

The *even (online Kolmogorov) complexity* [3] of a string z is

$$C_{\rm ev}(z) = \min\{|p| : U(p, z_1 \dots z_{i-1}) = z_i \text{ for all } i = 2, 4, \dots, \leq |z|\} .$$

Again, there exists a class of optimal machines U for which $C_{\rm ev}$ is minimal within an additive constant and we assume that U is such a machine. Note that $C(x|y) - O(1) \leq C_{\rm ev}(y_1 x_1 \dots y_n x_n) \leq C(x) + O(1)$ for n-bit x and y. Let $C_{\rm ev}(w|v)$ be the conditional variant. The chain rule for the concatenation vw of strings v and w holds: $C_{\rm ev}(vw) = C_{\rm ev}(v) + C_{\rm ev}(w|v) + O(\log(|v|))$, see the full version of the paper. In a similar way $C_{\rm odd}(x)$ is defined. A direct lower and upper bound for $C_{\rm odd} + C_{\rm ev}$ are[3]

$$C(z) - O(\log|z|) \leq (C_{\rm odd} + C_{\rm ev})(z) \leq 2C(z) + O(1) .$$

The lower bound is almost tight, for example if all even bits of z are zero. Surprisingly, the upper bound can also be almost tight and $C_{\rm odd} + C_{\rm ev}$ can change significantly after a simple permutation of the bits.

▶ **Theorem 1.** *For every $\varepsilon > 0$ there exist $\delta > 0$ and a sequence ω such that for large n*

$$\begin{matrix} C_{\rm odd}(\omega_1 \dots \omega_n) \\ C_{\rm ev}(\omega_1 \dots \omega_n) \end{matrix} \geq (1-\varepsilon) C(\omega_1 \dots \omega_n) + \delta n .$$

Moreover, for all even n

$$C_{\rm odd}(\omega_2\omega_1 \dots \omega_n\omega_{n-1}) = C(\omega_1 \dots \omega_n) + O(\log n) \quad (1)$$

$$C_{\rm ev}(\omega_2\omega_1 \dots \omega_n\omega_{n-1}) \leq O(1) . \quad (2)$$

The first part implies

$$\limsup_{|x|\to\infty} \frac{C_{\rm odd}(x) + C_{\rm ev}(x)}{C(x)} \geq 2 ,$$

[3] The $O(\log|x|)$ term could be decreased to $O(1)$ if we compared online complexity with decision complexity [17] as in [3]. However, plain and decision complexity differ by at most $O(\log|x|)$, and because we focus on linear bounds, we do not use this rare variant of complexity.

and by the upper bound $C_{\text{odd}}, C_{\text{ev}} \leq C + O(1)$, this supremum equals 2. Recall the definition $IT(x \to y) = C(y) - C_{\text{ev}}(x_1 y_1 \dots x_n y_n)$ for x, y, n such that $n = |x| = |y|$. Let $x = \omega_1 \omega_3 \dots \omega_{2n-1}$ and $y = \omega_2 \omega_4 \dots \omega_{2n}$, Theorem 1 implies

$$\begin{aligned} IT(x \to y) &\leq \varepsilon C(x,y) + O(1) \\ IT(y \to x) &= C(x,y) + O(1)\,, \end{aligned}$$

(where $C(x,y) \geq \delta n - O(1)$).[4]

Theorem 1 can be generalized to dialogues between $k \geq 2$ machines, i.e. if k sources need to perform a dialogue, it can happen that each source must contain almost full information about the dialogue. Moreover, if the order is changed, the "contribution" of all except one source becomes computable. Let the complexity of bits i mod k be given by

$$C_{i \bmod k}(x) = \min\{|p| : U(p, x_1 \dots x_{j-1}) = x_j \text{ for all } j = i, i+k, \dots, \leq |x|\}\,.$$

For every k and $\varepsilon > 0$ there exist a $\delta > 0$ and a sequence ω such that for all $i \leq k$ and large n

$$C_{i \bmod k}(\omega_1 \dots \omega_n) \geq (1-\varepsilon) C(\omega_1 \dots \omega_n) + \delta n$$

Moreover, for $\tilde{\omega} = \omega_k \omega_1 \dots \omega_{k-1}\, \omega_{2k} \omega_{k+1} \dots \omega_{2k-1} \dots$ for all n, and $i = 2 \dots k$:

$$\begin{aligned} C_{1 \bmod k}(\tilde{\omega}_1 \dots \tilde{\omega}_n) &= C(\omega_1 \dots \omega_n) + O(\log n) \\ C_{i \bmod k}(\tilde{\omega}_1 \dots \tilde{\omega}_n) &\leq O(1)\,. \end{aligned}$$

In Theorem 1 the difference between C and $C_{\text{odd}} + C_{\text{ev}}$ is linear in the length of the prefix of ω. One might wonder how big this difference can be. A direct bound is $|x|/2 + O(1)$. Indeed, the odd complexity of x is at most $C(x)$ hence

$$(C_{\text{odd}} + C_{\text{ev}})(x) - C(x) = (C_{\text{odd}}(x) - C(x)) + C_{\text{ev}}(x) \leq O(1) + |x|/2 + O(1)\,.$$

The next theorem shows that the difference can indeed be $c|x|$ for a significant c.

▶ **Theorem 2.** *There exist a sequence ω such that for all n*

$$(C_{\text{odd}} + C_{\text{ev}})(\omega_1 \dots \omega_n) \geq n(\log \tfrac{4}{3})/2 + C(\omega_1 \dots \omega_n) - O(\log n)\,.$$

Moreover, Equations (1) *and* (2) *are satisfied.*

The factor $(\log \frac{4}{3})/2$ can be further improved to $(\log \frac{3}{2})/2 \approx 0.292$ at the cost of weakening (1) and (2) (see full version of this paper). On the other hand, the upper bound $1/2$ can not be reached:

▶ **Theorem 3.** *There exist $\beta < \frac{1}{2}$ such that for large x*

$$(C_{\text{ev}} + C_{\text{odd}} - C)(x) \leq \beta |x|\,.$$

In summary, $\frac{1}{2} \log \frac{3}{2} \leq \limsup \frac{(C_{\text{ev}} + C_{\text{odd}} - C)(x)}{|x|} < \frac{1}{2}$, but the precise value of the lim sup is unknown.

[4] For the first we use $C(y) \leq C(\omega_{1\dots 2n}) = C(x,y)$ up to $O(1)$ terms. For the second $C(x,y) \geq C(x) \geq C_{\text{ev}}(y_1 x_1 \dots y_n x_n) = C(x,y)$, thus $C(x) = C(x,y)$, while $C_{\text{ev}}(y_1 x_1 \dots y_n x_n) \leq O(1)$. Also, note that $C(\omega_{1\dots 2n})$ must exceed δn because it exceeds $C_{\text{odd}}(\omega_{1\dots 2n}) \geq \delta n$, all up to $O(1)$ terms.

3 Online semimeasures

We show that the problem of constructing strings where additivity of online complexity is violated is equivalent to constructing lower semicomputable semimeasures that can not be factorized into "odd" and "even" online lower semicomputable semimeasures. Before defining such semimeasures and reformulating Theorems 1–3, we recall the algorithmic coding theorem.

A (continuous) semimeasure P is a function from strings to $[0,1]$ such that $P(x0)+P(x1) \leq P(x)$ for all x. A real function f on strings is lower semicomputable if the set of all pairs (x,r) of strings and rational numbers such that $f(x) \leq r$ is enumerable. There exist a maximal lower semicomputable semimeasure $M(x)$, i.e. a lower semicomputable that exceeds any other such semimeasures within a constant factor: $M(x) = \sum_i 2^{-i} P_i(x)$ for an enumeration P_1, P_2, ... of all such semimeasures (see [5, 8, 16] for details). The coding theorem [8, Theorem 4.3.4] implies

$$\log 1/M(x) = C(x) + O(\log C(x)) .$$

An *even (online) semimeasure* [3] is a function from strings to $[0,1]$ such that for all x

i. $P(x0) + P(x1) \leq P(x)$ if $|x0|$ is even,
ii. $P(x0) = P(x1) = P(x)$ otherwise.

The coding theorem generalizes to the online setting.

▶ **Theorem 4** ([3]). *There exist maximal even (respectively odd) semimeasures. All such semimeasures M_{ev} (resp. M_{odd}) satisfy*

$$\log 1/M_{\mathrm{ev}}(x) = C_{\mathrm{ev}}(x) + O\left(\log C_{\mathrm{ev}}(x)\right) .$$

Let $\omega_{k\ldots l} = \omega_k \ldots \omega_l$. Theorems 1, 2 and 3 follow from

▶ Proposition 5. For all $\varepsilon > 0$ and lower semicomputable odd and even online semimeasures Q_{odd} and Q_{ev}, there exist δ, a sequence ω, a lower semicomputable semimeasure P, and a partial computable F such that for all n

$$(Q_{\mathrm{odd}}Q_{\mathrm{ev}})(\omega_{1\ldots n}) \leq (1-\delta)^n P(\omega_{1\ldots n})^{2-2\varepsilon}$$

and $F(\omega_{1\ldots 2n}, \omega_{2n+2}) = \omega_{2n+1}$.

▶ Proposition 6. For all lower semicomputable odd and even online semimeasures Q_{odd} and Q_{ev}, there exist a sequence ω, a lower semicomputable semimeasure P, and a partial computable F such that for all n

$$(Q_{\mathrm{odd}}Q_{\mathrm{ev}})(\omega_{1\ldots 2n}) \leq (3/4)^n P(\omega_{1\ldots 2n})$$

and $F(\omega_{1\ldots 2n}, \omega_{2n+2}) = \omega_{2n+1}$.

▶ Proposition 7. For all lower semicomputable semimeasures Q, there exist $\alpha > \sqrt{1/2}$ and a family of odd and even semimeasures $P_{\mathrm{odd},n}$ and $P_{\mathrm{ev},n}$ uniformly lower-semicomputable in n, such that for all x

$$P_{\mathrm{odd},|x|}(x) P_{\mathrm{ev},|x|}(x) \geq \alpha^{|x|} Q(x)/4 . \tag{3}$$

Proof that Proposition 7 implies Theorem 3. Choose $Q = M$ in Proposition 7 and let for a sufficiently small $c > 0$

$$P_{\mathrm{odd}}(x) = c\left(\frac{1}{1^2}P_{\mathrm{odd},1}(x) + \frac{1}{2^2}P_{\mathrm{odd},2}(x) + \ldots\right) .$$

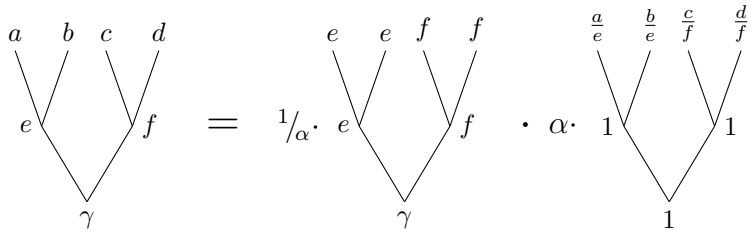


Figure 1 Decomposing semimeasures into odd and even ones.

Note that P_{odd} is a lower semicomputable odd semimeasure and by universality $P_{\text{odd}}(x) \leq O(M_{\text{odd}}(x))$. Hence $-\log M_{\text{odd}}(x) \leq -\log P_{\text{odd},|x|}(x) + O(\log|x|)$. Similar for $P_{\text{ev}}(x)$. By the online coding theorem we obtain up to terms $O(\log|x|)$,

$$(C_{\text{odd}} + C_{\text{ev}})(x) \leq -\log\left(P_{\text{odd},|x|}(x)P_{\text{ev},|x|}(x)\right) \leq -|x|\log\alpha - \log Q(x)\,.$$

Here, $-\log\alpha < 1/2$ and the last term is bounded by $-\log M(x) \leq C(x) + O(\log|x|)$. The $O(\log|x|)$ can be removed for large $|x|$ by choosing $-\log\alpha < \beta < 1/2$. ◀

Proof that Proposition 6 implies Theorem 2. Choosing $Q_{\text{odd}} = M_{\text{odd}}$ and $Q_{\text{ev}} = M_{\text{ev}}$, the first part is immediate by the coding theorem and (2) follows directly from the definition of even complexity. For any x we have

$$C_{\text{odd}}(x) - O(1) \leq C(x) \leq C_{\text{odd}}(x) + C_{\text{ev}}(x) + O(\log|x|)$$

We obtain (1) by applying $C_{\text{ev}}(x) \leq O(1)$. ◀

Proof that Proposition 5 implies Theorem 1. For Theorem 1 we also apply Proposition 5 with $Q_{\text{odd}} = M_{\text{odd}}$ and $Q_{\text{ev}} = M_{\text{ev}}$ to obtain for some $\delta' > 0$

$$(C_{\text{odd}} + C_{\text{ev}})(\omega_{1\ldots 2n}) \geq (2 - 2\varepsilon)C(\omega_{1\ldots 2n}) + \delta' n\,.$$

Notice that $C_{\text{odd}} \leq C + O(1)$, hence $C_{\text{ev}}(\omega_{1\ldots 2n}) \geq (1 - 2\varepsilon)C(\omega_{1\ldots 2n}) + \delta' n$; and similar for C_{odd}. Conditions (1) and (2) follow in a similar way as above. ◀

The generalization of Theorem 1 mentioned in section 2 is shown in the full version. We remark that P in these theorems can not be computable, this follows from the subsequent lemma.

▶ **Lemma 8.** *For every computable semimeasure P, there exist computable odd and even online semimeasures P_{odd} and P_{ev} such that $P_{\text{odd}}P_{\text{ev}} = P$.*

Proof. Let ε be the empty string and let $P_{\text{odd}}(\varepsilon) = P(\varepsilon)$ and $P_{\text{ev}}(\varepsilon) = 1$. Suppose that at some node x we have defined $P_{\text{odd}}(x)$ and $P_{\text{ev}}(x)$ such that $P_{\text{odd}}(x)P_{\text{ev}}(x) = P(x)$. Then P_{odd} and P_{ev} are defined on 2-bit extensions of x according to Figure 1 for $\gamma = P(x)$ and $\alpha = P_{\text{ev}}(x)$ [our assumption implies $P_{\text{odd}}(x) = \gamma/\alpha$]. Note that P_{odd} and P_{ev} are indeed computable odd and even semimeasures and that $P_{\text{odd}}P_{\text{ev}} = P$. ◀

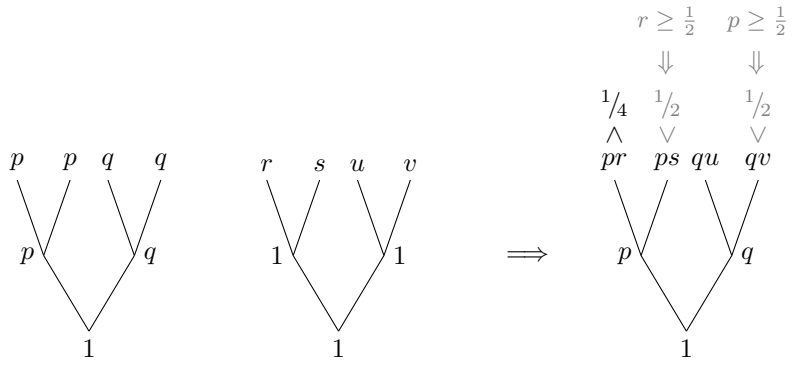


Figure 2 Game for Proposition 6 with $n = 1$.

4 Proofs of lower bounds

We start with Proposition 6, and repeat it for convenience.

▶ Proposition. For all lower semicomputable odd and even online semimeasures Q_{odd} and Q_{ev}, there exist a sequence ω, a lower semicomputable semimeasure P, and a partial computable F such that for all n

$$(Q_{\text{odd}}Q_{\text{ev}})(\omega_{1\ldots 2n}) \leq (3/4)^n P(\omega_{1\ldots 2n})$$

and $F(\omega_{1\ldots 2n}, \omega_{2n+2}) = \omega_{2n+1}$.

To develop some intuition, we first consider a game. The game is played between two players (Alice and Bob) who alternate turns. Alice maintains values for $P(x)$ on 2-bit x. At each round she might pass or increase some values as long as $\sum\{P(x) : |x| = 2\} = 3/4$. Bob maintains lower semicomputable odd and even semimeasures $Q_{\text{odd}}(x)$ and $Q_{\text{ev}}(x)$, see figure 2. Also Bob might pass or increase some values as long as the conditions of the definition of online semimeasure are satisfied, (hence $\max\{p+q, r+s, u+v\} \leq 1$ in figure 2). Alice wins if in the limit $P(x) \geq Q_{\text{odd}}(x)Q_{\text{ev}}(x)$ holds for some x (i.e. if $P(00) \geq pr$ or $P(01) \geq ps$ or $P(10) \geq qu$ or $P(11) \geq qv$).

In this game Alice has a winning strategy. She starts by putting $1/4$ at one leaf and zero at the others, say $P(00) = 1/4$. Then she waits until Bob increases either Q_{odd} or Q_{ev} above $1/2$ at this leaf (thus $Q_{\text{odd}}(0) = Q_{\text{odd}}(00) > 1/2$ or $Q_{\text{ev}}(00) > 1/2$). If none of this happens, Alice wins. Otherwise if $Q_{\text{odd}}(0) > 1/2$, she plays $P(11) = 1/2$ and if $Q_{\text{ev}}(00) > 1/2$, she plays $P(01) = 1/2$. In the first case Alice wins because $Q_{\text{odd}}(1) \leq 1 - Q_{\text{odd}}(0) < 1/2$ and hence $Q_{\text{odd}}(1)Q_{\text{ev}}(11) < 1/2$ and in the second case she wins because $Q_{\text{ev}}(01) \leq 1 - Q_{\text{ev}}(00) < 1/2$ and hence $Q_{\text{odd}}(0)Q_{\text{ev}}(01) < 1/2$. Note that in both cases $\sum\{P(x) : |x| = 2\} = 1/2 + 1/4$, (and otherwise it is $1/4$) and Alice's condition is always satisfied. (Also note that the second bit of x on which Alice wins is 1 if $Q_{\text{odd}}(0) > 1/2$ or $Q_{\text{ev}}(00) > 1/2$. So for lower-semicomputable Q_{odd} and Q_{ev}, we can use this bit to determine which inequality was first realized, and hence to compute the first bit of x. A similar observation will be used to construct F in the proof below.)

To show the proposition, we need to concatenate strategies for the game above to strategies for larger games. For this, it seems that the winning rule needs to be strengthened, and this makes either the winning rule or the winning strategy for the small game complicated. Therefore, in the more concise proof below, we gave a formulation without use of game technique.

Proof. We construct $\omega_{1\dots 2n}$ together with thresholds o_n, e_n inductively. Let $o_0 = e_0 = 1$. For x of length $2n$, consider the conditions $Q_{\text{odd}}(x0) > o_n/2$ and $Q_{\text{ev}}(x00) > e_n/2$. We fix some algorithm that enumerates Q_{odd} and Q_{ev} from below and after each update tests both conditions. Let O_x be the condition that $Q_{\text{odd}}(x0) > o_n/2$ is true at some update and $Q_{\text{ev}}(x00) > e_n/2$ did not appear at any update strictly before; and let E_x be the condition that $Q_{\text{ev}}(x00) > e_n/2$ is true after some update but $Q_{\text{odd}}(x0) > o_n/2$ is false at the current update (and hence at any update before). Note that O_x and E_x cannot happen both. Let

$$(\omega_{2n+1}\omega_{2n+2}, o_{n+1}, e_{n+1}) = \begin{array}{llll} (11, & o_n/2, & e_n) & \text{if } O_{\omega_{1\dots 2n}} \text{ happens,} \\ (01, & o_n, & e_n/2) & \text{if } E_{\omega_{1\dots 2n}} \text{ happens,} \\ (00, & o_n/2, & e_n/2) & \text{otherwise.} \end{array}$$

By induction it follows that $o_n \geq Q_{\text{odd}}(\omega_{1\dots 2n})$ and $e_n \geq Q_{\text{ev}}(\omega_{1\dots 2n})$. Indeed, this follows directly for $n = 0$. For $n \geq 1$, consider the case where $O_{\omega_{1\dots 2n}}$ happens. Thus $\omega_{1\dots 2n+2} = \omega_{1\dots 2n+1}1$ and

$$Q_{\text{odd}}(\omega_{1\dots 2n}1) \leq Q_{\text{odd}}(\omega_{1\dots 2n}) - Q_{\text{odd}}(\omega_{1\dots 2n}0) \leq o_n - o_n/2 = o_n/2\,.$$

On the other hand, $Q_{\text{ev}}(\omega_{1\dots 2n+2}) \leq Q_{\text{ev}}(\omega_{1\dots 2n}) \leq e_n = e_{n+1}$. The case where $E_{\omega_{1\dots 2n}}$ happens is similar, and the last one is direct.

It remains to define F and P such that $F(\omega_{1\dots 2n}, \omega_{2n+2}) = \omega_{2n+1}$ and

$$P(\omega_{1\dots 2n}) = (4/3)^n o_n e_n\,.$$

Note that $\omega_{2n+2} = 1$ iff $O_{\omega_{1\dots 2n}}$ or $E_{\omega_{1\dots 2n}}$ happens, and knowing that one of the events happens, we can decide which one and therefore also ω_{2n+1}. Hence, given $\omega_{1\dots 2n}$ and ω_{2n+2} we can compute ω_{2n+1} and this procedure defines the partial computable function F.

To define P, observe that ω can be approximated from below: start with $\omega = 00\dots$, each time $O_{\omega_{1\dots 2n}}$ (respectively $E_{\omega_{1\dots 2n}}$) happens, change $\omega_{2n}\omega_{2n+1}$ from 00 to 01 (respectively to 11), let all subsequent bits be zero, and repeat the process. Hence, for all n and $2n$-bit x at most one pair (o_n, e_n) is defined which we denote as (o_x, e_x). Let $P(x)$ be zero unless (o_x, e_x) is defined in which case

$$P(x) = (4/3)^{|x|/2} o_x e_x\,.$$

Note that P is lower semicomputable and the equation above is satisfied. Also, P is a semimeasure: $P(\varepsilon) = (4/3)^0 \cdot 1 \cdot 1 = 1$, and in all three cases we have $\sum\{o_{xbb'}e_{xbb'} : b, b' \in \{0,1\}\} \leq 3o_x e_x/4$ hence, $\sum\{P(xbb') : b, b' \in \{0,1\}\} \leq P(x)$. ◀

The proof of Proposition 5 follows the same structure.

▶ Proposition. For all $\varepsilon > 0$ and lower semicomputable odd and even online semimeasures Q_{odd} and Q_{ev}, there exist δ, a sequence ω, a lower semicomputable semimeasure P, and a partial computable F such that for all n

$$(Q_{\text{odd}}Q_{\text{ev}})(\omega_{1\dots n}) \leq (1-\delta)^n P(\omega_{1\dots n})^{2-2\varepsilon}$$

and $F(\omega_{1\dots 2n}, \omega_{2n+2}) = \omega_{2n+1}$.

Proof. We first consider the following variant for the game above on strings of length two. Alice should satisfy the weaker condition $\sum\{P(x) : |x| = 2\} \leq 1 - \delta$, where $\delta \ll \varepsilon$ will be determined later. She wins if

$$(P_{\text{odd}}P_{\text{ev}})(x) \leq (P(x))^{2-2\varepsilon}$$

for some x. The idea of the winning strategy is to start with a very small value somewhere, say $P(00) = \delta$. If $\varepsilon = 0$ then Bob could reply with $Q_{\rm odd}(0) = Q_{\rm ev}(00) = \delta$, (in fact he could win by always choosing $Q_{\rm odd}(x) = Q_{\rm ev}(x) = P(x)$). For $\varepsilon > 0$ and $\delta \ll \varepsilon$ one of the online semimeasures should exceed $\delta^{1-\varepsilon} = k\delta$ for $k = \delta^{-\varepsilon}$. k can be arbitrarily large if $\delta \ll \varepsilon$ is chosen sufficiently small. At his next move, (as before), Alice puts all his remaining measure, i.e. $1-2\delta$ in a leaf that does not belong to a branch where the corresponding online semimeasure is large. Note that $1-2\delta$ is close to 1 and taking a power $2 \geq 2-2\varepsilon$ we see that Bob needs at least $1-4\delta$ in each online semimeasure, but he already used $k\delta$ in one of them.

More precisely, the winning strategy for Alice is to set $P(00) = \delta$ and wait until $Q_{\rm odd}(0) > \delta^{1-\varepsilon}$ or $Q_{\rm ev}(00) > \delta^{1-\varepsilon}$. If these conditions are never satisfied, then Alice wins on $x = 00$. Suppose at some moment Alice observes that the first condition holds, then she plays $P(11) = 1-2\delta$, in the other case she plays $P(01) = 1-2\delta$. Afterwards she does not play anymore. Note that $\sum\{P(x) : |x| = 2\} \leq 1-\delta$. We show that Alice wins. Assume that $Q_{\rm odd}(0) > \delta^{1-\varepsilon}$ (the other case is similar). We know that $Q_{\rm ev}(11) \leq 1$ hence if Alice does not win, this implies $Q_{\rm odd}(1) > (1-2\delta)^{2-2\varepsilon}$. This is lower bounded by $(1-2\delta)^2 \geq 1-4\delta$. We choose $\delta = 2^{-2/\varepsilon}$. This implies

$$\delta^{1-\varepsilon} = 2^{-(2/\varepsilon)(1-\varepsilon)} = 2^{-2/\varepsilon+2} = 4\delta.$$

Hence $Q_{\rm odd}(0) + Q_{\rm odd}(1) > 4\delta + (1-4\delta) = 1$ and Bob would violate his restrictions. Therefore Alice wins. For later use notice that in the first case our argument implies $Q_{\rm odd}(1) \leq (1-2\delta)^{2-2\varepsilon}$.

In a similar way as before we adapt Alice's strategy to an inductive construction of ω and P: let O_x and E_x be defined as before using conditions $Q_{\rm odd}(x0) > o_n\delta^{1-\varepsilon}$ and $Q_{\rm ev}(x00) > e_n\delta^{1-\varepsilon}$. Let $\beta = (1-2\delta)^{2-2\varepsilon}$ and let ω, o_n and e_n be given by

$$(\omega_{2n+1}\omega_{2n+2}, o_{n+1}, e_{n+1}) = \begin{array}{llll} (11, & o_n\beta, & e_n) & \text{if } O_{\omega_{1\ldots 2n}} \text{ happens,} \\ (01, & o_n, & e_n\beta) & \text{if } E_{\omega_{1\ldots 2n}} \text{ happens,} \\ (00, & o_n\delta^{1-\varepsilon}, & e_n\delta^{1-\varepsilon}) & \text{otherwise.} \end{array}$$

This implies $o_n \geq Q_{\rm odd}(\omega_{1\ldots 2n})$ and $e_n \geq Q_{\rm ev}(\omega_{1\ldots 2n})$. F is defined and shown to satisfy the condition in exactly the same way. It remains to construct P such that

$$(1-\delta)^n P(\omega_{1\ldots 2n}) = (o_n e_n)^{1/(2-2\varepsilon)} \ ,$$

(the proposition follows after rescaling δ). In a similar way as before o_x and e_x are defined and let

$$P(x) = (1-\delta)^{-|x|/2}(o_x e_x)^{1/(2-2\varepsilon)} \, .$$

To show that P is indeed a semimeasure observe that $\sum\{P(xbb') : b, b' \in \{0,1\}\}$

$$\begin{aligned} &= (1-\delta)^{-|x|/2-1} \sum\{(o_{xbb'}e_{xbb'})^{1/(2-2\varepsilon)} : b, b' \in \{0,1\}\} \\ &\leq (1-\delta)^{-|x|/2-1} \left(\beta^{1/(2-2\varepsilon)} + \delta\right) (o_x e_x)^{1/(2-2\varepsilon)} \ , \end{aligned}$$

and because $\beta^{1/(2-2\varepsilon)} = 1-2\delta$ this equals

$$= (1-\delta)^{-|x|/2} \, (o_x e_x)^{1/(2-2\varepsilon)} = P(x) \, . \qquad \blacktriangleleft$$

Acknowledgements. The author is grateful to Alexander Shen, Nikolay Vereshchagin, Andrei Romashchenko, Mikhail Dektyarev, Ilya Mezhirov and Emmanuel Jeandel for extensive discussion and many useful suggestions. I also thank Ilya Mezhirov for implementing clever code to study some games. Especially thanks to Alexander Shen for encouragement after presenting earlier results and for arranging funding by grant NAFIT ANR-08-EMER-008-01. The author is also grateful to Mathieu Hoyrup who arranged a grant under which the work was finalized.

References

1 B. Bauwens and A. Shen. An additivity theorem for plain Kolmogorov complexity. *Theory Computing Systems*, 52(2):297–302, 2013.

2 Bruno Bauwens. *Computability in statistical hypotheses testing, and characterizations of independence and directed influences in time series using Kolmogorov complexity.* PhD thesis, Ghent University, May 2010.

3 A. Chernov, A. Shen, N. Vereshchagin, and V.Vovk. On-line probability, complexity and randomness. In *ALT'08: Proceedings of the 19th international conference on Algorithmic Learning Theory*, pages 138–153, Berlin, Heidelberg, 2008. Springer-Verlag.

4 U. Feldmann and J. Bhattacharya. Predictability improvement as an asymmetrical measure of interdependence in bivariate time series. *International Journal of Bifurcation and Chaos*, 14(2):505–514, 2004.

5 P. Gács. Lecture notes on descriptional complexity and randomness. `http://www.cs.bu.edu/faculty/gacs/papers/ait-notes.pdf`, 1988–2011.

6 C. W. J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37:424–438, 1969.

7 G. John. Inference and causality in economic time series models. In Z. Griliches and M. D. Intriligator, editors, *Handbook of Econometrics*, volume 2, chapter 19, pages 1101–1144. Elsevier, 1984.

8 M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer-Verlag, New York, 2008.

9 An. A. Muchnik. Algorithmic randomness and splitting of supermartingales. *Problems of Information Transmission*, 45(1):54–64, March 2009.

10 M. Palus and A. Stefanovska. Direction of coupling from phases of interacting oscillators: an information theoretic approach. *Physical Review E, Rapid Communications*, 67:055201(R), 2003.

11 A. Papana, C. Kyrtsou, D. Kugiumtzis, and C. Diks. Simulation study of direct causality measures in multivariate time series. *Entropy*, 15(7):2635–2661, 2013.

12 J. Pearl. *Causality: Models, Reasoning and Inference.* Cambridge University Press, 2000.

13 F.A. Razak. *Mutual information based measures on complex interdependent networks of neuro data sets.* PhD thesis, Imperial College London, March 2013.

14 M. G. Rosenblum and A. S. Pikovsky. Detecting direction of coupling in interacting oscillators. *Phys. Rev. E*, 64(4):045202, Sep 2001.

15 T. Schreiber. Measuring information transfer. *Physical Review Letters*, 85(2):461–464, Jul 2000.

16 V. A. Uspensky, N. K.Vereshchagin, and A. Shen. Kolmogorov complexity and algorithmic randomness. To appear.

17 V. A. Uspensky and A. Shen. Relations between varieties of Kolmogorov complexities. *Theory of Computing Systems*, 29(3):271–292, 1996.

18 M. Winterhalder, B. Schelter, W. Hesse, K. Schwab, L. Leistritz, R. Bauer, J. Timmer, and H. Witte. Comparison of linear signal processing techniques to infer directed interactions in multivariate neural systems. *Signal Processing*, 85:2137–2160, 2005.

19 A. K. Zvonkin and L. A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, 25(6:156):83–124, 1970.

Two-Page Book Embeddings of 4-Planar Graphs*

Michael A. Bekos[1], Martin Gronemann[2], and Chrysanthi N. Raftopoulou[3]

1 **Wilhelm-Schickhard-Institut für Informatik, Universität Tübingen, Germany**
bekos@informatik.uni-tuebingen.de
2 **Institut für Informatik, Universität zu Köln, Germany**
gronemann@informatik.uni-koeln.de
3 **School of Applied Mathematical & Physical Sciences, NTUA, Greece**
crisraft@mail.ntua.gr

Abstract

Back in the eighties, Heath [7] showed that every 3-planar graph is subhamiltonian and asked whether this result can be extended to a class of graphs of degree greater than three. In this paper we affirmatively answer this question for the class of 4-planar graphs. Our contribution consists of two algorithms: The first one is limited to triconnected graphs, but runs in linear time and uses existing methods for computing hamiltonian cycles in planar graphs. The second one, which solves the general case of the problem, is a quadratic-time algorithm based on the book embedding viewpoint of the problem.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Book Embedding, Subhamiltonicity, 4-Planar Graphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.137

1 Introduction

Book embeddings have a long history and arise in various application areas such as VLSI design [5]. In a *book embedding* the placement of nodes is restricted to a line, the *spine* of the book. The edges are assigned to different *pages* of the book. A page can be thought of as a half-plane bounded by the spine where the edges are drawn as circular arcs between their endpoints. We say that a graph admits a *k-page book embedding* if one can assign the edges to k pages and there exists a linear ordering of the nodes on the spine such that no two edges of the same page cross. The minimum number of pages required to construct such an embedding is the *book thickness* or *page number* of a graph and has received much attention in the past. Yannakakis [14] describes a linear-time algorithm to embed every planar graph into a book of four pages. Bernhart et al. [2] show that a graph is two-page embeddable iff it is subhamiltonian. A *subhamiltonian graph* is a subgraph of a planar hamiltonian graph. It is *NP*-complete to determine whether a graph is subhamiltonian [13]. Often referred to as *augmented hamiltonian cycle*, a *subhamiltonian cycle* is a cyclic sequence of nodes in a graph that would form a hamiltonian cycle when adding the missing edges without destroying planarity. The relation between subhamiltonian cycles and two-page book embeddings is

* Work on this problem began at Dagstuhl Seminar 13151. We thank the organizers, participants and Prof. Dr. M. Kaufmann. The work of M.A. Bekos is implemented within the framework of the Action "Supporting Postdoctoral Researchers" of the Operational Program "Education and Lifelong Learning" (Action's Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State.



© Michael A. Bekos, Martin Gronemann, and Chrysanthi N. Raftopoulou;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 137–148



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



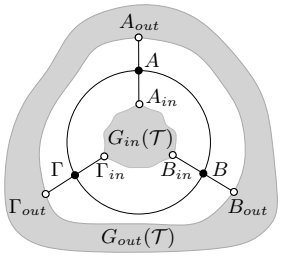


Figure 1 Triangle $\mathcal{T}$ separating $G_{in}(\mathcal{T})$ and $G_{out}(\mathcal{T})$ on removal.

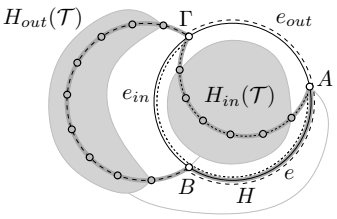


Figure 2 Merging $H_{in}(\mathcal{T})$ (dotted) and $H_{out}(\mathcal{T})$ (dashed) into H (bold gray).

quite intuitive. The order of the nodes on the spine is equivalent to the cyclic order of the subhamiltonian cycle. The edges are partitioned by whether they lie in the interior of the cycle or not.

An early result is due to Whitney [12], who proves that every maximal planar graph with no separating triangles is hamiltonian. Tutte [11] shows that every 4-connected planar graph has a hamiltonian cycle. Chiba et al. [4] provide a linear-time algorithm to find a hamiltonian cycle in a 4-connected planar graph. Chen [3] gives a proof that every maximal planar graph with at least five vertices and no separating triangles is 4-connected. Sanders [10] generalizes a theorem of Thomassen and shows that any 4-connected planar graph has a hamiltonian cycle that contains two arbitrarily chosen edges of the graph. Based on Whitney's theorem, Kainen et al. [9] show that every planar graph with no separating triangles is subhamiltonian. Another result is by Chen [3] who shows that if a maximal planar graph contains only one such triangle, then it is hamiltonian. Helden [8] improves this result further to two triangles. The aforementioned results are all related to the problem of embedding planar graphs into two pages. However, there is an extensive amount of literature on embedding various types of graphs into books; see e.g. [6]. One result that is interesting in our context is that of Heath [7], who describes a linear-time algorithm to embed any 3-planar graph into two pages.

We study the problem of embedding 4-planar graphs into books with two pages. We tackle this problem from two sides. The first approach is restricted to triconnected graphs (Section 2) but builds on existent results and is therefore of a simple nature compared to the second approach. Extending it to biconnected graphs is not straightforward, though. The algorithm of Section 3 –which is less efficient in terms of time complexity– exploits the degree restriction to construct a two-page book embedding. Due to space constraints, some of our proofs are only sketched, omitted details are given in [1].

2 Subhamiltonicity of Triconnected 4-Planar Graphs

In this section, we first investigate properties of separating triangles in 4-planar graphs and then we use those to derive a solution for a single separating triangle. Unlike Chen [3] and Helden [8], we are able to extend our approach to an unbounded number of triangles by exploiting the degree restriction. We say a subhamiltonian cycle H *crosses* a face if there are two consecutive vertices in H that are incident to the face but not adjacent to each other.

▶ **Lemma 1.** *Every triconnected planar graph with no separating triangles has a subhamiltonian cycle that crosses every face at most once and it can be computed in linear time.*

Proof. In the triconnected case, Kainen et al. [9] construct a maximal planar graph $G' = (V', E')$ by inserting a vertex into each non-triangular face of G and connect it to the vertices

of that face. Clearly, this takes linear time and G' is 4-connected. We can use the linear-time algorithm of Chiba et al. [4] to obtain a hamiltonian cycle H' for G'. Deleting the vertices of $V' - V$ yields a subhamiltonian cycle H for G that crosses each face at most once. ◀

Given an embedded triconnected 4-planar graph G with a fixed outerface and a separating triangle $\mathcal{T}$ with vertices $V(\mathcal{T}) = \{A, B, \Gamma\}$, we denote the subgraph of G contained in $\mathcal{T}$ by $G_{in}(\mathcal{T})$ and the subgraph of G outside $\mathcal{T}$ by $G_{out}(\mathcal{T})$. We also denote $\overline{G}_{in}(\mathcal{T}) = G - G_{out}(\mathcal{T})$ and $\overline{G}_{out}(\mathcal{T}) = G - G_{in}(\mathcal{T})$. Since G is triconnected and 4-planar, every vertex of $\mathcal{T}$ has degree four and is adjacent to exactly one vertex in $G_{in}(\mathcal{T})$ and $G_{out}(\mathcal{T})$, respectively. We denote these with $A_{in}, B_{in}, \Gamma_{in}$ and $A_{out}, B_{out}, \Gamma_{out}$, respectively (see Fig. 1).

▶ **Lemma 2.** *Given a 4-planar triconnected graph G and a separating triangle $\mathcal{T} = \{A, B, \Gamma\}$, then $A_{in}, B_{in}, \Gamma_{in} (A_{out}, B_{out}, \Gamma_{out})$ are pairwise distinct or all represent the same vertex.*

Proof. In the other case, where w.l.o.g. $A_{in} = B_{in} = v$ and $\Gamma_{in} \neq v$, there exists a separation pair (v, Γ) contradicting the triconnectivity of G. A symmetric argument applies to $A_{out}, B_{out}, \Gamma_{out}$. ◀

▶ **Lemma 3.** *In a 4-planar triconnected graph, every pair of distinct separating triangles $\mathcal{T}$ and $\mathcal{T}'$ is vertex disjoint, i.e. $V(\mathcal{T}) \cap V(\mathcal{T}') = \emptyset$.*

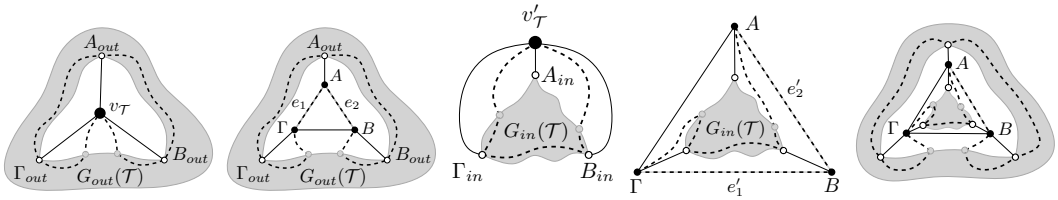
Proof. Assume to the contrary that $\mathcal{T}$ and $\mathcal{T}'$ share an edge or a vertex. In the first case, let w.l.o.g. $e = (u, v)$ be the common edge. The degree of both u and v is at least five, since three edges are required for $\mathcal{T}, \mathcal{T}'$ and two additional edges to connect $G_{in}(\mathcal{T})$ and $G_{in}(\mathcal{T}')$ to $\mathcal{T}$ and $\mathcal{T}'$, respectively. In the second case, let v denote the common vertex. Since v is part of two edge disjoint cycles and connected to $G_{in}(\mathcal{T})$ and $G_{in}(\mathcal{T}')$, it follows that $deg(v) \geq 6$. ◀

Consider now a 4-planar triconnected graph with a single separating triangle $\mathcal{T}$. Similar to Chen [3], the idea is to compute two cycles $H_{in}(\mathcal{T})$ and $H_{out}(\mathcal{T})$ for $\overline{G}_{in}(\mathcal{T})$ and $\overline{G}_{out}(\mathcal{T})$ and link them via the separating triangle together. The crucial observation is that if two cycles intersect as illustrated in Fig. 2, i.e., they contain two edges of the triangle but have only one of them in common, then we can always merge them into one cycle.

▶ **Lemma 4.** *Let G be a triconnected 4-planar graph, $\mathcal{T}$ a separating triangle, and $H_{in}(\mathcal{T})$ and $H_{out}(\mathcal{T})$ two subhamiltonian cycles for $\overline{G}_{in}(\mathcal{T})$ and $\overline{G}_{out}(\mathcal{T})$, resp. If $E(H_{in}(\mathcal{T})) \cap E(\mathcal{T}) = \{e_{in}, e\}$ and $E(H_{out}(\mathcal{T})) \cap E(\mathcal{T}) = \{e_{out}, e\}$ where $\{e, e_{in}, e_{out}\}$ are the edges of $\mathcal{T}$, then G is subhamiltonian.*

Proof. Let w.l.o.g. $e = (A, B)$, $e_{in} = (B, \Gamma)$ and $e_{out} = (A, \Gamma)$ as illustrated in Fig. 2. The result of removing the edges of $\mathcal{T}$ from both cycles are two paths $P_{out} = B \rightsquigarrow \Gamma$ and $P_{in} = \Gamma \rightsquigarrow A$. Joining them at Γ and inserting e yields a subhamiltonian cycle. ◀

It remains to show that we can always find two cycles that satisfy the requirements of Lemma 4. We neglect the degenerated case of Lemma 2, where $G_{out}(\mathcal{T})$ or $G_{in}(\mathcal{T})$ is a single vertex, because finding a cycle in that case is trivial. Consider for example $\overline{G}_{out}(\mathcal{T})$. To obtain $H_{out}(\mathcal{T})$, we temporarily replace $\mathcal{T}$ in $\overline{G}_{out}(\mathcal{T})$ with a single vertex $v_{\mathcal{T}}$ as depicted in Fig. 3a. The resulting graph $G^*_{out}(\mathcal{T})$ remains 4-planar and triconnected, because $deg(v_{\mathcal{T}}) = 3$ by construction and any path via $\mathcal{T}$ can use $v_{\mathcal{T}}$ instead. One may argue that this operation may introduce additional separating triangles. However, such a triangle must contain $v_{\mathcal{T}}$ and, therefore, $deg(v_{\mathcal{T}}) = 4$, a contradiction. Now let us assume that $H^*_{out}(\mathcal{T})$ is a subhamiltonian cycle for $G^*_{out}(\mathcal{T})$. The idea is to reinsert $\mathcal{T}$ and reroute $H^*_{out}(\mathcal{T})$ through $\mathcal{T}$ such that the resulting cycle $H_{out}(\mathcal{T})$ contains two edges $e_1, e_2 \in E(\mathcal{T})$.



(a) $v_\mathcal{T}$ in $G^*_{out}(\mathcal{T})$ **(b)** $\mathcal{T}$ in $\overline{G}_{out}(\mathcal{T})$ **(c)** $v'_\mathcal{T}$ in $G^*_{in}(\mathcal{T})$ **(d)** $\mathcal{T}$ in $\overline{G}_{in}(\mathcal{T})$ **(e)** G with $\mathcal{T}$ and H

Figure 3 (a) Subhamiltonian cycle $H^*_{out}(\mathcal{T})$ in $G^*_{out}(\mathcal{T})$ containing $v_\mathcal{T}$. (b) Augmenting $H^*_{out}(\mathcal{T})$ yields $H_{out}(\mathcal{T})$ containing edges $e_1 = (\Gamma, A)$ and $e_2 = (A, B)$. (c) Dummy vertex $v'_\mathcal{T}$ as replacement for $\mathcal{T}$ in $G^*_{in}(\mathcal{T})$ and a cycle $H^*_{in}(\mathcal{T})$. (d) Rerouting $H^*_{in}(\mathcal{T})$ through $\mathcal{T}$ resulting in $H_{in}(\mathcal{T})$ with edges $e'_1 = (\Gamma, B)$ and $e_2 = (A, B)$. (d) The result of merging $H_{in}(\mathcal{T})$ and $H_{out}(\mathcal{T})$ into a cycle H for G.

▶ **Lemma 5.** *Let G be a triconnected 4-planar graph, $\mathcal{T}$ a separating triangle. Furthermore, let $G^*_{out}(\mathcal{T})$ denote the graph resulting from replacing $\mathcal{T}$ by a vertex $v_\mathcal{T}$ in $\overline{G}_{out}(\mathcal{T})$. A subhamiltonian cycle $H^*_{out}(\mathcal{T})$ for $G^*_{out}(\mathcal{T})$ can be augmented to a subhamiltonian cycle $H_{out}(\mathcal{T})$ for $\overline{G}_{out}(\mathcal{T})$ such that it contains two edges of $\mathcal{T}$, i.e., $E(H_{out}(\mathcal{T})) \cap E(\mathcal{T}) = \{e_1, e_2\}$. If $H^*_{out}(\mathcal{T})$ crosses every face of $G^*_{out}(\mathcal{T})$ at most once, one may choose any pair $e_1, e_2 \in E(\mathcal{T})$ to lie on $H_{out}(\mathcal{T})$.*

Proof. We only sketch the proof. It is sufficient to consider every combination of e_1, e_2 and the location of the predecessor and successor of $v_\mathcal{T}$ in $H^*_{out}(\mathcal{T})$. It immediately becomes clear that in almost every situation $H^*_{out}(\mathcal{T})$ can be rerouted through $\mathcal{T}$ such that the resulting cycle $H_{out}(\mathcal{T})$ contains two prescribed edges e_1, e_2 of $\mathcal{T}$. Only in one case, where $H^*_{out}(\mathcal{T})$ crosses an incident face twice to visit $v_\mathcal{T}$, a specific combination of edges is required. ◀

In the single-separating triangle scenario, both $\overline{G}_{out}(\mathcal{T})$ and $\overline{G}_{in}(\mathcal{T})$ are free of separating triangles. Therefore, we may construct two graphs $G^*_{out}(\mathcal{T}), G^*_{in}(\mathcal{T})$ by replacing $\mathcal{T}$ with dummy vertices. Applying Lemma 1 to them yields two subhamiltonian cycles $H^*_{out}(\mathcal{T})$ and $H^*_{in}(\mathcal{T})$, both crossing every face of $G^*_{out}(\mathcal{T})$ and $G^*_{in}(\mathcal{T})$ at most once. Hence, we may augment them with the aid of Lemma 5 such that they contain each two edges of $\mathcal{T}$. By choosing the combination of edges such that $H_{out}(\mathcal{T})$ and $H_{in}(\mathcal{T})$ meet the requirements of Lemma 4, we can merge them into a single subhamiltonian cycle H for G.

While the property that $G^*_{out}(\mathcal{T})$ and $G^*_{in}(\mathcal{T})$ are both free of separating triangles enables us to conveniently choose two edges for each cycle $H_{out}(\mathcal{T}), H_{in}(\mathcal{T})$, this only works for a single separating triangle. However, a closer look reveals that it is sufficient to have a choice for either $H_{out}(\mathcal{T})$ or $H_{in}(\mathcal{T})$, not necessarily both of them. The idea is to first augment the cycle for which we do not have a choice to see which edges of $\mathcal{T}$ are part of it, then we choose the edges for the second cycle accordingly. We summarize the idea as the main result of this section and describe it in a more formal manner in form of a proof.

▶ **Theorem 6.** *Every triconnected 4-planar graph is subhamiltonian.*

Proof. Let G denote a triconnected 4-planar graph and $\tau(G)$ the number of separating triangles in G. We prove by induction and claim that for any $\tau(G) \geq 0$, we can compute a subhamiltonian cycle H for G. *Base case*: Since $\tau(G) = 0$, we can directly apply Lemma 1. *Inductive case:* For $\tau(G) > 0$, we pick a separating triangle $\mathcal{T}$ such that $\tau(\overline{G}_{in}(\mathcal{T})) = 0$. Let $G^*_{out}(\mathcal{T})$ be the result of replacing $\mathcal{T}$ by $v_\mathcal{T}$ in $\overline{G}_{out}(\mathcal{T})$. Notice that $\tau(G^*_{out}(\mathcal{T})) = \tau(G) - 1$ holds. Hence, by induction hypothesis, $G^*_{out}(\mathcal{T})$ has a subhamiltonian cycle $H^*_{out}(\mathcal{T})$. We

reinsert $\mathcal{T}$ and augment $H^*_{out}(\mathcal{T})$ such that the result $H_{out}(\mathcal{T})$ contains two (arbitrary) edges e_1, e_2 of $\mathcal{T}$. In a similar way, we replace $\mathcal{T}$ in $\overline{G}_{in}(\mathcal{T})$ by $v'_{\mathcal{T}}$ to obtain $G^*_{in}(\mathcal{T})$. Since $\tau(\overline{G}_{in}(\mathcal{T})) = \tau(G^*_{in}(\mathcal{T})) = 0$ holds, we can apply Lemma 1 to $G^*_{in}(\mathcal{T})$ and compute a cycle $H^*_{in}(\mathcal{T})$ that crosses each face at most once. With Lemma 5 we may obtain a cycle $H_{in}(\mathcal{T})$ for $\overline{G}_{in}(\mathcal{T})$ with two edges $e'_1, e'_2 \in E(\mathcal{T})$ of our choice. Choosing $e'_1 = e_1$ and $e'_2 \neq e_2$ yields two cycles $H_{out}(\mathcal{T}), H_{in}(\mathcal{T})$ that meet the requirements of Lemma 4 and we can merge them into one cycle H for G. ◀

The proof of Theorem 6 is constructive. Embedding G and identifying all separating triangles in G can be done in linear time. Augmenting a cycle and merging two of them takes constant time. Disjointness of separating triangles yields a linear number of subproblems and every edge occurs in at most one such subproblem. Hence, the total time spent for the subroutine of Lemma 1 is linear in the size of G.

▶ **Corollary 7.** *A subhamiltonian cycle of a triconnected 4-planar graph can be found in linear time.*

3 Two-Page Book Embeddings of General 4-Planar Graphs

In this section, we prove that any planar graph of maximum degree 4 admits a two-page book embedding. W.l.o.g. we assume that the input graph G is biconnected, since it is known that the page number of a graph equals the maximum of the page number of its biconnected components [2]. One can also neglect the exact geometry, as two edges that are drawn on the same page cross iff their endpoints alternate along the spine. We say that an edge e *nests a vertex* v iff one endpoint of e is to the left of v along the spine and the other endpoint of e to its right. We also say that an edge e *nests an edge* e' iff both e and e' are drawn on the same page and both endpoints of e' are nested by e. Observe that nested edges do not cross.

Our approach is as follows: First remove from G cycle C_{out} delimiting the outerface of G and *contract* each bridge-block of the remaining graph into a single vertex. Let F be the implied graph, which is a forest, as $G - C_{out}$ is not necessarily connected. C_{out} is embedded, s.t.: (i) the order of the vertices of C_{out} along the spine is fixed (and follows the one in which the vertices of C_{out} appear along C_{out}), and, (ii) all edges of C_{out} are on the same page, except for the one that connects its outermost vertices. Then we describe how to embed without crossings: (i) the chords of C_{out}, (ii) forest F, and, (iii) the edges between C_{out} and F. To obtain a two-page book embedding of G, we replace each vertex of F with a cycle (embedded similarly to C_{out}), whose length equals to the length of the cycle delimiting the outerface of the bridge-block it corresponds to in $G - C_{out}$, and recursively embed its interior.

More formally, consider an arbitrary simple cycle $C : v_1 \to v_2 \to \ldots \to v_k \to v_1$ of G. The removal of C results in two planar subgraphs $G_{in}(C)$ and $G_{out}(C)$ of G that are the components of $G - C$ that lie in the interior and exterior of C in G, resp. Note that $G_{in}(C)$ and $G_{out}(C)$ are not necessarily connected. Let $\overline{G}_{in}(C)$ ($\overline{G}_{out}(C)$, resp.) be the subgraph of G induced by C and $G_{in}(C)$ ($G_{out}(C)$, resp.). For the recursive step, we assume the following invariant properties:

IP-1: The order of the vertices of $\overline{G}_{out}(C)$ along the spine ℓ is fixed and the page in which each edge of $\overline{G}_{out}(C)$ is drawn (i.e., top- or bottom-drawn) is determined s.t. the book embedding of $\overline{G}_{out}(C)$ is planar. In other words, we assume that we have already produced a two-page book embedding for $\overline{G}_{out}(C)$, in which no edge crosses the spine.

IP-2: The combinatorial embedding of $\overline{G}_{out}(C)$ is consistent with a given planar combinatorial embedding of G.

IP-3: The vertices of C occupy consecutive positions along ℓ, s.t. v_1 (v_k, resp.) is the leftmost (rightmost, resp.) along ℓ. Moreover, all edges of C are on the same page, except for the one that connects v_1 and v_k. Say w.l.o.g. that (v_1, v_k) is on the top-page (or *top-drawn*), while the remaining edges of C, namely edges (v_i, v_{i+1}) for $1 \leq i < k$, are on the bottom-page (or *bottom-drawn*).

IP-4: If C is not identified with the cycle delimiting the outerface of G, the degree of either v_1 or v_k is at most 3 in $\overline{G}_{in}(C)$. Say w.l.o.g. that v_k is of degree at most 3.

IP-5: If vertex v_1 has degree 4 in $\overline{G}_{in}(C)$, then it is adjacent to zero or two chords of C.

We note that the combinatorial embedding specified in IP-2 is maintained throughout the whole drawing process. This combined with the fact that every edge entirely lies on one page is sufficient to ensure planarity. Note that we first present the recursive step of our algorithm and then its base, as this approach shows better how the different ideas flow one after the other. Let v_i be a vertex of C, $i = 1, \ldots, k$. Since G is of max-degree 4, v_i is incident to at most two undrawn edges. Assume that v_i has at least one undrawn edge. We refer to the edge incident to v_i that follows $(v_i, v_{(i+1) \bmod k})$ in the counterclockwise order of the edges around v_i (as defined by the combinatorial embedding specified by IP-2), as the *right edge* of v_i. If v_i is adjacent to two undrawn edges, then the one that is not identified with the right edge of v_i is its *left edge*; otherwise, the left and the right edge of v_i are identified.

Initially, we draw the chords of C on the top-page. By IP-2 and IP-3, no two chords intersect. We then draw $G_{in}(C)$ and the edges between C and $G_{in}(C)$. Note that $G_{in}(C)$ is not necessarily connected. Hence, its bridge-block trees form a forest. As already stated, we contract each bridge-block of $G_{in}(C)$ into a single vertex, which we call *block-vertex*; see Figs. 4a-4b. We distinguish two types of block-vertices: those adjacent to vertices of C (*anchors*) and those adjacent to other block-vertices only (*ancillaries*). From the contraction, it follows that an edge between C and a certain anchor can be of multiplicity at most two. Edges among block-vertices are always simple. We will first determine the positions of all anchors along ℓ. Consider an anchor c, then among the edges between c and C, we select and *mark* exactly one, s.t.: (i) the marked edge will be drawn on the bottom-page and (ii) all other edges incident to c (i.e., either edges between c and C that are not marked, or between c and block-vertices) will be drawn on the top-page. Let $v_{l,c}$ be the leftmost vertex of C adjacent to c along ℓ. If $(c, v_{l,c})$ is simple, we select and mark this edge. Otherwise, we mark the right edge of $v_{l,c}$. Hence, each anchor has exactly one marked edge and each vertex of C is incident to at most two marked edges. Let $v \in C$ be a vertex of C adjacent to at least one anchor through a marked edge. We distinguish two cases:

Case 1 *v is adjacent to exactly two anchors c and c' through two marked edges e and e', resp.*: Assume w.l.o.g. that e is the left edge of v and e' its right edge. Then, both c and c' are placed directly to the right of v and c precedes c' (see Fig. 4d). Note that v cannot be the rightmost vertex of C due to IP-4.

Case 2 *v is adjacent to one anchor c through a marked edge e*: If $deg(v) = 3$ in $\overline{G}_{in}(C)$, then we distinguish two sub-cases. If v is not the rightmost vertex of C, then c is placed directly to the right of v (see Fig. 4e). Otherwise, directly to its left (see Fig. 4f). It now remains to consider the case where $deg(v) = 4$ in $\overline{G}_{in}(C)$. In this case, by IP-4 it follows that v is not the rightmost vertex of C. Again, we distinguish two sub-cases:

- *If e is the right edge of v*, then c is placed directly to the right of v (see Fig. 4g).
- *If e is the left edge of v*, then c is placed directly to the left of v (see Fig. 4h); v cannot be the leftmost vertex of C, as the right edge of v would be a chord, violating IP-5.

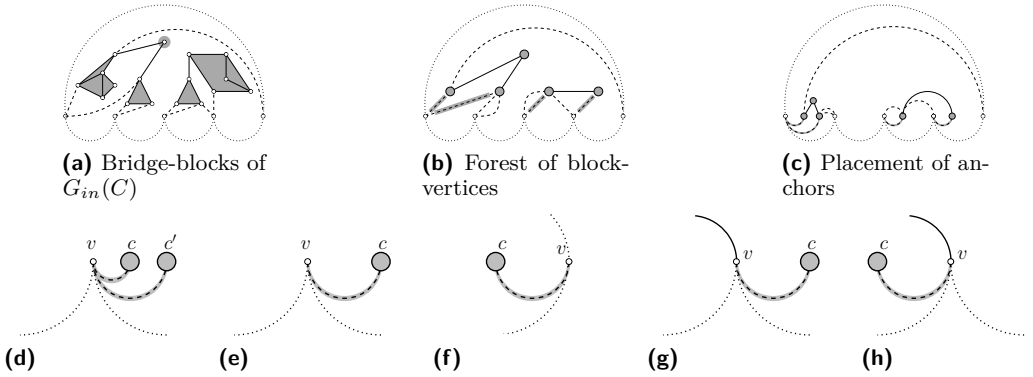


Figure 4 In all figures, the edges of C are drawn dotted, bridge-blocks are colored gray and edges between C and anchors are drawn dashed; marked edges are highlighted in gray.

All marked edges are bottom-drawn. Edges between anchors and C that are not marked are top-drawn; see Fig. 4c. Observe that we do not change the underlying combinatorial embedding of G, preserving IP-2. Hence, the book embedding constructed so far is planar.

Now observe that ancillaries form a new forest (*forest of ancillaries*), which is a subgraph of the initial forest containing all block-vertices. Let T be a tree of the forest of ancillaries and let $c_1, \ldots, c_t$ be anchors that (i) are adjacent to at least one ancillary of T, and (ii) c_i is to the left of c_{i+1}, $i = 1, \ldots, t-1$. We refer to $c_1, \ldots, c_t$ as the *anchors* of T, and to the tree formed by T and its anchors as the *anchored tree* of T, denoted by $\overline{T}$. We say that two anchors of $\overline{T}$ are *consecutive* iff there is no anchor of $\overline{T}$ between them.

▶ **Lemma 8.** *For anchored trees the following hold: (i) Two trees $\overline{T}$ and $\overline{T'}$ share at most a common anchor; (ii) $\overline{T}$ contains at least two anchors; and (iii) every leaf of $\overline{T}$ is an anchor of $\overline{T}$, and vice versa.*

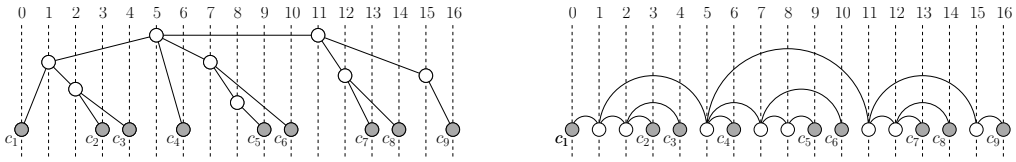
Proof. The assumption that one of the two properties does not hold contradicts either the connectivity or biconnectivity of G. ◀

Assume now that $\overline{T}$ is rooted at anchor c_1 (*rooted anchored tree*). For an anchor or ancillary c of $\overline{T}$, denote by $p(c)$ the parent of c in $\overline{T}$ and let $p(c_1)$ be any of the vertices of C adjacent to c_1. For an ancillary c of T (i.e., non-leaf in $\overline{T}$), we define an order for its children: if c' and c'' are children of c, then $c' < c''$ iff c' precedes c'' in the counterclockwise order of the edges around c (defined by the combinatorial embedding specified by IP-2), when starting from $(c, p(c))$. By this order, we label the vertices of $\overline{T}$ as they appear in the pre-order traversal of $\overline{T}$ (*labeled anchored tree*); see Fig.5a.

▶ **Lemma 9.** *For each ancillary c of a labeled anchored tree $\overline{T}$ there is (i) at least an anchor of $\overline{T}$ with label smaller than that of c and (ii) at least another with label greater than that of c*

Proof. The leftmost anchor (i.e. root) of $\overline{T}$ is zero labeled, which proves (i). The greatest labeled vertex of $\overline{T}$ is a leaf of $\overline{T}$ (due to pre-order traversal) and by Lemma 8(iii) an anchor of $\overline{T}$ which proves (ii). ◀

We first define the order in which the trees of the forest of ancillaries will be drawn. Let G^T_{aux} be an auxiliary graph whose vertices correspond to trees and there is a directed edge $(v_{T'}, v_T)$ in G^T_{aux} iff $\overline{T'}$ has an anchor between two consecutive anchors of $\overline{T}$. The desired order is defined by a topological sorting of G^T_{aux}, which exists due to the following lemma.



(a) A labeled anchored tree $\overline{T}$ **(b)** The placement of the ancillaries of $\overline{T}$

Figure 5 In both figures, anchors are colored gray; the indices of the vertical grid-lines denote the labeling of $\overline{T}$.

▶ **Lemma 10.** *Auxiliary graph G^T_{aux} is a directed acyclic graph.*

Proof. Assume to the contrary that there is a cycle $v_{T_1} \to \ldots v_{T_s} \to v_{T_1}$ in G^T_{aux}. Let I_i be the interval defined by the left/right-most anchors of $\overline{T_i}$. Edge $(v_{T_i}, v_{T_{i+1}})$ implies that there is an anchor of $\overline{T_i}$ between consecutive anchors of $\overline{T_{i+1}}$. However, in this case all anchors of $\overline{T_i}$ should be between the same two anchors of $\overline{T_{i+1}}$, as otherwise the embedding specified by IP-2 is not planar. So, $I_i \subseteq I_{i+1}$. By Lemma 8(i), it follows that $I_i \neq I_{i+1}$. Hence, $I_1 \subset \ldots \subset I_s \subset I_1$, a contradiction. ◀

Lemma 10 implies that drawing the trees in the order defined by a topological sorting of G^T_{aux}, assures that the tree T' will be drawn before T, if $\overline{T'}$ has an anchor that is between two consecutive anchors of $\overline{T}$ along ℓ. Now assume that we have drawn zero or more of these trees s.t. (i) all edges are top-drawn, (ii) there are no edge crossings, and (iii) the combinatorial embedding specified by IP-2 is preserved. Let T be the next tree to be drawn. The following lemma presents an important property of our drawing approach.

▶ **Lemma 11.** *Assume that all trees that precede T in a topological sorting of G^T_{aux} have been drawn on the top-page without edge crossings by preserving the combinatorial embedding specified by IP-2. If e is a top-drawn edge that does not belong to $\overline{T}$ and nests at least one anchor of $\overline{T}$, then it nests all anchors of $\overline{T}$.*

Proof. The detailed proof is based on a case-analysis on the type of edge e: (*i*) top-drawn edge of C; (*ii*) edge of an anchored tree $\overline{T'}$ drawn before $\overline{T}$; and (*iii*) not an edge of a previously drawn anchored tree (i.e., each endpoint of e is either a vertex of C or an anchor). ◀

We now describe how to draw T on the top page s.t. (i) there are no edge crossings, and, (ii) the combinatorial embedding specified by IP-2 is preserved. More precisely, we place each ancillary c of T between a pair of consecutive anchors of $\overline{T}$, s.t. the label of c is larger (smaller) than the label of the anchor to its left (right)[1]; for ancillaries placed between the same pair of anchors, the one with smaller label is to the left; all edges of $\overline{T}$ are top-drawn (see Fig.5b). Note that we have not fully specified the exact positions of the ancillaries of $\overline{T}$ along ℓ, since between consecutive anchors of $\overline{T}$ there may exist anchors that do not belong to $\overline{T}$ or vertices of C or anchors/ancillaries of trees that have already been drawn. Details will be given shortly. Notice that all ancillaries of $\overline{T}$ are placed between its left/right-most anchors, which by Lemma 11 implies that if a top-drawn edge (that does not belong to $\overline{T}$) nests at least one anchor of $\overline{T}$, then it nests the entire tree $\overline{T}$. By exploiting the correspondence between the left-to-right order of the vertices of $\overline{T}$ along ℓ and the labeling of $\overline{T}$, we can prove that the drawing of $\overline{T}$ is planar.

[1] Note that the existence of this pair of consecutive anchors of $\overline{T}$ is implied by Lemma 9.

▶ **Lemma 12.** *The drawing of the anchored tree $\overline{T}$ is planar.*

Proof. Assume to the contrary that $e = (c_1, c_2)$ and $e' = (c'_1, c'_2)$ of $\overline{T}$ cross. Since e and e' are top-drawn, their endpoints alternate along ℓ. Let the order on ℓ be $c_1 \to c'_1 \to c_2 \to c'_2$. Hence, c_1 is the parent of c_2, as the label of c_1 is smaller than that of c_2 and they are adjacent in $\overline{T}$. Similarly, c'_1 is the parent of c'_2. Since between c_1 and c_2 are drawn subtrees of $\overline{T}$ rooted at children of c_1 other than c_2, c'_1 and c'_2 belong to a subtree rooted at a child of c_1, different from c_2, which implies that the label of c'_2 is smaller than that of c_2, a contradiction. ◀

Recall that we have not fully specified the exact positions of the ancillaries of $\overline{T}$ along ℓ. Consider the following scenario. There is a path P of top-drawn edges (e.g., non-marked edges incident to C and/or edges of previously drawn trees) joining a pair of consecutive anchors of $\overline{T}$ and our algorithm must place an ancillary c of $\overline{T}$ between them. Since c is nested by an edge of P and all edges of $\overline{T}$ are top-drawn, an edge connecting c with an ancillary of $\overline{T}$ placed between another pair of consecutive anchors of $\overline{T}$ will cross P. The following lemma ensures that this scenario cannot occur, as such a path cannot exist.

▶ **Lemma 13.** *Let $u_0, u_1, \ldots, u_{l+1}$, $l \geq 0$, be vertices (anchors/ancillaries are treated as vertices) drawn on ℓ from left to right, s.t. u_0 and u_{l+1} are two consecutive anchors of $\overline{T}$. Assume that all trees anchored at $u_1, \ldots, u_l$ have been drawn on the top-page without edge crossings by preserving the combinatorial embedding specified by IP-2, while T has not been drawn. Then, there is an index $i \in \{0, 1, \ldots, l\}$, such that no two adjacent vertices u_k and u_m exist with $0 \leq k \leq i$, $i+1 \leq m \leq l+1$ and (u_k, u_m) is top-drawn.*

Proof. Assume to the contrary that for all $i \in \{0, \ldots, l\}$, there are two adjacent vertices u_k and u_m with $0 \leq k \leq i$, $i+1 \leq m \leq l+1$ and (u_k, u_m) is on the top-page. One can prove that there is a top-drawn path $P(u_0 \to u_{l+1}) : u_0 \to u_{j_1} \ldots u_{j_p} \to u_{l+1}$ consisting of vertices of $\{u_0, \ldots, u_{l+1}\}$, whose edges are top-drawn and for each edge of $P(u_0 \to u_{l+1})$ there is not a top-drawn edge with endpoints in $\{u_0, \ldots, u_{l+1}\}$ that nests it. However, the existence of $P(u_0 \to u_{l+1})$ implies that G should contain a vertex of degree five, a contradiction. ◀

We are now ready to specify the exact positions of the ancillaries of $\overline{T}$ along ℓ. Assume that a particular number of ancillaries of $\overline{T}$ should be drawn between two consecutive anchors c_i and c_{i+1} of $\overline{T}$, $i = 1, \ldots, t-1$. By Lemma 13, there is a pair of vertices that are between c_i and c_{i+1} along ℓ and there is not a top-drawn edge with endpoints between c_i and c_{i+1} nesting both of these vertices. We place between this pair of vertices all ancillaries of $\overline{T}$ that must reside between c_i and c_{i+1}, without changing their relative order, i.e., for ancillaries placed between c_i and c_{i+1}, the one with smaller label is to the left. Lemma 12 ensures the planarity of $\overline{T}$. It remains to prove that the combinatorial embedding specified by IP-2 is preserved.

▶ **Lemma 14.** *Assume that all trees that precede T in a topological sorting of G^T_{aux} have been drawn on the top-page without edge crossings by preserving the combinatorial embedding specified by IP-2. When $\overline{T}$ is drawn, the combinatorial embedding specified by IP-2 is also preserved.*

Proof. We only sketch the proof. Since the drawing of $\overline{T}$ preserves the order of the edges around all ancillaries, the combinatorial embedding specified by IP-2 is preserved for all ancillaries of $\overline{T}$. Then, one can prove that this property is propagated to all vertices of T. ◀

The following lemma focuses on the case where C contains a vertex with degree 2 in $\overline{G}_{in}(C)$ (other than its leftmost or rightmost vertex). We will utilize this lemma later.

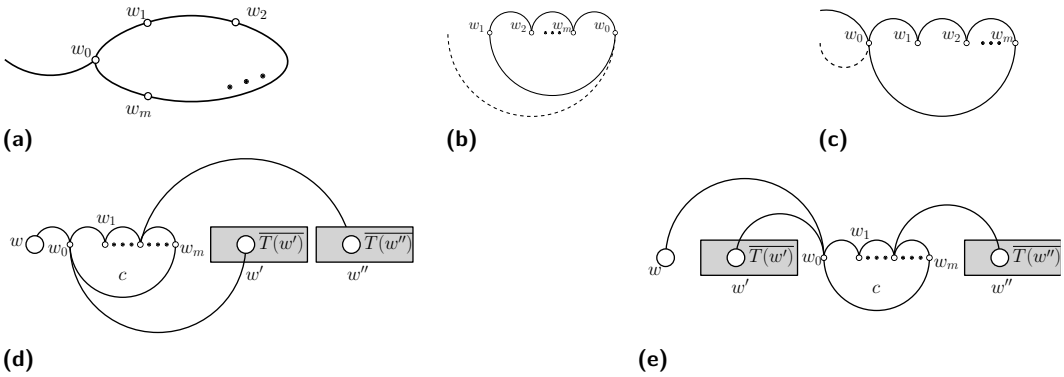


Figure 6 (a) The outerface of a block-vertex c. (b)–(c) different cases that occur when drawing the outerface of c, in the case where c is anchor. (d) Ancillary c needs to be repositioned. (e) Its placement is determined by Lemma 16.

▶ **Lemma 15.** *Let v be a vertex of C with degree 2 in $\overline{G}_{in}(C)$ that is not the left/right-most vertex of C. Let also v_r (v_l) be its next neighbor on C to its right (left resp.). Since edge (v, v_r) belongs to C, it is drawn on the bottom-page. However, it can also be drawn on the top-page without edge-crossings, while the combinatorial embedding specified by IP-2 is maintained.*

Proof. We only sketch the proof. First observe that if no block-vertex is drawn between v and v_r, then obviously (v, v_r) can be drawn on the top-page. Otherwise, one can move the block-vertices in between to the left of v, so that v and v_r are consecutive along ℓ. ◀

Up to now, we have drawn $\overline{G}_{in}(C)$, s.t., every bridge-block of $G_{in}(C)$ is contracted to a block-vertex that lies on ℓ and each edge is drawn either on the bottom (if it is a marked edge) or on the top-page (otherwise). Next, we describe how to recursively proceed. Let c be a block-vertex of $G_{in}(C)$ with outerface $\mathcal{F}_c$. Initially, assume that $\mathcal{F}_c$ is a simple cycle. If c is an anchor, denote by w_0 the vertex of $\mathcal{F}_c$ incident to the marked edge of c. If c is an ancillary, then c belongs to an anchored tree. In this case, w_0 denotes the vertex of $\mathcal{F}_c$ adjacent to the closest neighbor of c to its left, which is well-defined since c is always placed between two consecutive anchors of the anchored tree it belongs to. Let $w_0, w_1, \ldots, w_m$ be the vertices of $\mathcal{F}_c$, in the clockwise traversal of $\mathcal{F}_c$ from w_0 (see Fig. 6a).

If c is an anchor (i.e., w_0 is incident to a marked edge), then we place the vertices of $\mathcal{F}_c$ on ℓ as follows: (i) w_0 occupies the position of c and it is the rightmost vertex of $\mathcal{F}_c$ on ℓ, (ii) w_1 is the leftmost vertex of $\mathcal{F}_c$ on ℓ, (iii) w_i is to the left of w_{i+1} for $i = 1, \ldots, m-1$, and, (iv) there are no vertices in between; see Fig. 6b. All edges of $\mathcal{F}_c$ are top-drawn, except for (w_1, w_0). This placement is feasible, except for the case in which in the combinatorial embedding specified by IP-2 there is an edge incident to w_0 that is between (w_0, w_1) and the marked edge incident to w_0 in the counterclockwise order of the edges around w_0 when starting from (w_0, w_1); see Fig. 6c. In this case, we place w_0 to the left of $w_1, \ldots, w_m$, s.t. w_0 is the leftmost vertex of $\mathcal{F}_c$. So, (w_0, w_m) is the bottom-drawn edge of $\mathcal{F}_c$.

Suppose now that c is an ancillary. Let w be the closest neighbor of c to its left on ℓ. w is the parent of c in the tree in which c belongs to and (w_0, w) is top-drawn. We place the vertices of $\mathcal{F}_c$ as follows: (i) w_0 occupies the position of c and it is the leftmost vertex of $\mathcal{F}_c$ on ℓ, (ii) w_m is the rightmost vertex of $\mathcal{F}_c$ on ℓ, (iii) w_i is to the left of w_{i+1}, $i = 1, \ldots, m-1$, and, (iv) there are no vertices in between. All edges of $\mathcal{F}_c$ are top-drawn, except for (w_0, w_m). This placement is infeasible only when in the combinatorial embedding specified by IP-2 there

is an edge incident to w_0, say (w_0, w'), and between (w_0, w_m) and (w_0, w) in the clockwise order of the edges around w_0 when starting from (w_0, w_m); see Fig. 6d. Since c has only its parent to its left among the block-vertices of the anchored tree it belongs to, it follows that, w' is to the right of c. So, (w_0, w') cannot be drawn on the top-page, without deviating the combinatorial embedding specified by IP-2. Since G is biconnected, c is adjacent to at least another block-vertex, say w'', s.t. $w'' \notin \{w, w'\}$. The following lemma takes care of this case.

▶ **Lemma 16.** *Ancillary c can be repositioned on ℓ, s.t.: (i) c is placed between two consecutive anchors of $\overline{T}$. (ii) The combinatorial embedding specified by IP-2 is preserved and the edges (w_0, w), (w_0, w') and (c, w'') are top-drawn and crossing-free. (iii) w_0 is leftmost vertex of $\mathcal{F}_c$ and w_i is to the left of w_{i+1}, $i = 1, \ldots, m-1$; All edges of $\mathcal{F}_c$ are top-drawn, except for (w_0, w_m).*

Proof. w is the parent of c and w', w'' are children of c in $\overline{T}$, with w' being the first child of c. For our proof, w'' is its second child. So, (c, w), (c, w') and (c, w'') are consecutive around c as in Fig. 6d. Let $\overline{T(w')}$ and $\overline{T(w'')}$ be subtrees of $\overline{T}$ rooted at w' and w'', resp. c is to the left of all vertices of $\overline{T(w')}$, all vertices of $\overline{T(w')}$ are to the left of all vertices of $\overline{T(w'')}$ and there are no ancillaries of $\overline{T}$ in between. We place c between the rightmost (leftmost) anchor of $\overline{T(w')}$ ($\overline{T(w'')}$); see Fig. 6e. So, c is placed between two consecutive anchors of $\overline{T}$. If we place the vertices of $\mathcal{F}_c$, with w_0 being leftmost on $\mathcal{F}_c$ and w_i to the left of w_{i+1}, then (w_0, w), (w_0, w') and (c, w'') are drawn on the top-page and the embedding is preserved. ◀

If we process all ancillaries that have to be repositioned from right to left along ℓ, then by Lemma 16 we obtain a planar drawing in which the embedding specified by IP-2 is preserved once the outerface of each block-vertex is drawn and all edges that connect block-vertices are eventually drawn on the top-page. Initially, we assumed that $\mathcal{F}_c$ is simple. If not so, $\mathcal{F}_c$ consists of smaller simple *subcycles*, s.t. (i) any two subcycles share at most one vertex of $\mathcal{F}_c$ and (ii) any vertex of $\mathcal{F}_c$ is incident to at most two subcycles. Hence, the *"tangency graph"* of these subcycles (which has a vertex for each subcycle and an edge between every pair of subcycles that share a vertex) is a tree. Define w_0 as in the case of simple cycle and let the tangency tree be rooted at the cycle containing w_0. Due to degree restriction, w_0 cannot be incident to two subcycles. We draw the subcycles of $\mathcal{F}_c$ in the order implied by the Breadth First Search (BFS) traversal of the tangency tree. The first one (incident to w_0) is drawn as in the case of simple cycle. Each next subcycle is plugged into the drawing, as in Fig. 7.

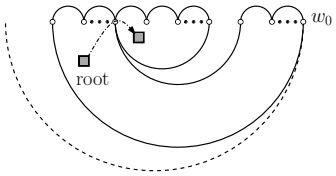


Figure 7 $\mathcal{F}_c$ is not simple.

It remains to ensure that IP-1 up to IP-5 are satisfied when a simple cycle, say C_s, is recursively drawn. IP-1 holds, since each edge is drawn either on the bottom (if it is a marked edge) or on the top-page (otherwise) and no two edges intersect. Lemma 14 implies IP-2. If C_s is the outerface of a block-vertex or a leaf in the tangency tree, then IP-3 trivially holds. If C_s is a non-leaf in the tangency tree, it contains at least one edge on the bottom-page (see Fig. 7). This violates IP-3. However, we can benefit from Lemma 15 since the edge which is improperly bottom-drawn is incident to a vertex (of degree four) that is not adjacent to any other vertex in the interior of C_s. For the sake of the recursion we assume that it is drawn on the top-page and once C_s is completely drawn, we redraw it on the bottom-page using Lemma 15. If C_s is the outerface of a block-vertex or root of the tangency tree of a non-simple outerface $\mathcal{F}_c$, then at least one vertex of C_s is adjacent to $G_{out}(C_s)$. If C_s is an internal node of the tangency tree of $\mathcal{F}_c$, then its leftmost vertex has two edges in $G_{out}(C_s)$. Hence, IP-4 also holds. Note that IP-5 does not necessarily hold. However, we can identify a

maximal separating path of chords of C_s adjacent to its leftmost vertex and use it to create two subinstances, which can be recursively drawn; refer to [1] for details.

The recursion begins by specifying a drawing of G with a chordless outerface $C_{out} : v_1 \to \ldots v_k \to v_1$. We place $v_1, \ldots, v_k$ in this order along ℓ and draw the edges of C_{out} as imposed by IP-3. If there is a vertex of C_{out} with degree less than four, then it is chosen as v_k and all invariant properties are satisfied. Otherwise, we appropriately augment our graph, so that IP-4 holds (the detailed proof is given in [1]). We are now ready to state our main theorem.

▶ **Theorem 17.** *Any planar graph of maximum degree 4 on n vertices admits a two-page book embedding, which can be constructed in $O(n^2)$ time.*

Proof. At each step, our algorithm performs a series of computations; the computation of the bridge-blocks, the topological sorting of G^T_{aux}, BFS-traversals on the tangency trees. All of these computations can be done in $O(n)$ time, resulting in $O(n^2)$ total time. ◀

4 Conclusions and Open Problems

Two approaches were proposed to embed a 4-planar graph into two pages. One reasonable question arising at this point is whether the result can be extended to 5-planar graphs.

References

1 Michael A. Bekos, Martin Gronemann, and Chrysanthi N. Raftopoulou. Two-page book embeddings of 4-planar graphs. Arxiv report arxiv.org/abs/1401.0684, 2013.

2 Frank Bernhart and Paul C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory*, 27(3):320–331, 1979.

3 Chiuyuan Chen. Any maximal planar graph with only one separating triangle is hamiltonian. *Journal of Combinatorial Optimization*, 7(1):79–86, 2003.

4 Norishige Chiba and Takao Nishizeki. The hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. *Journal of Algorithms*, 10(2):187–211, 1989.

5 Fan R. K. Chung, Frank T. Leighton, and Arnold L. Rosenberg. Embedding graphs in books: A layout problem with applications to VLSI design. *SIAM Journal on Algebraic and Discrete Methods*, 8(1):33–58, 1987.

6 Vida Dujmovic and David R. Wood. On linear layouts of graphs. *Discrete Mathematics & Theoretical Computer Science*, 6(2):339–358, 2004.

7 Lenwood S. Heath. *Algorithms for Embedding Graphs in Books*. PhD thesis, University of North Carolina, Chapel Hill, 1985.

8 Guido Helden. Each maximal planar graph with exactly two separating triangles is hamiltonian. *Discrete Applied Mathematics*, 155(14):1833–1836, 2007.

9 Paul C. Kainen and Shannon Overbay. Extension of a theorem of Whitney. *Applied Mathematics Letters*, 20(7):835–837, 2007.

10 Daniel P. Sanders. On paths in planar graphs. *Journal of Graph Theory*, 24(4):341–345, 1997.

11 William T. Tutte. A theorem on planar graphs. *Transactions of the American Mathematical Society*, 82(1):99–116, 1956.

12 Hassler Whitney. A theorem on graphs. *Annals of Mathematics*, 32(2):378–390, 1931.

13 Avi Wigderson. The complexity of the hamiltonian circuit problem for maximal planar graphs. Technical Report TR-298, EECS Department, Princeton University, 1982.

14 Mihalis Yannakakis. Embedding planar graphs in four pages. *Journal of Computer and System Sciences*, 38(1):36–67, 1989.

Palindrome Recognition In The Streaming Model

Petra Berenbrink[1], Funda Ergün[1,2], Frederik Mallmann-Trenn[1], and Erfan Sadeqi Azer[1]

1 **Simon Fraser University, Burnaby, Canada**
2 **Indiana University, Bloomington, US**

Abstract

A palindrome is defined as a string which reads forwards the same as backwards, like, for example, the string "racecar". In the *Palindrome Problem*, one tries to find all *palindromes* in a given string. In contrast, in the case of the *Longest Palindromic Substring Problem*, the goal is to find an arbitrary one of the longest palindromes in the string.

In this paper we present three algorithms in the streaming model for the the above problems, where at any point in time we are only allowed to use sublinear space. We first present a one-pass randomized algorithm that solves the *Palindrome Problem*. It has an *additive* error and uses $O(\sqrt{n})$ space. We also give two variants of the algorithm which solve related and practical problems. The second algorithm determines the exact locations of *all* longest palindromes using two passes and $O(\sqrt{n})$ space. The third algorithm is a one-pass randomized algorithm, which solves the *Longest Palindromic Substring Problem*. It has a *multiplicative* error using only $O(\log(n))$ space.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Palindromes, Streaming Model, Complementary Palindrome

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.149

1 Introduction

A palindrome is defined as a string which reads forwards the same as backwards, e.g., the string "racecar". In the *Palindrome Problem* one tries to find all *palindromes* (palindromic substrings) in an input string. A related problem is the *Longest Palindromic Substring Problem* in which one tries to find any one of the longest palindromes in the input.

In this paper we regard the streaming version of both problems, where the input arrives over time (or, alternatively, is read as a stream) and the algorithms are allowed space sub linear in the size of the input. Our first contribution is a one-pass randomized algorithm that solves the *Palindrome Problem*. It has an *additive* error and uses $O(\sqrt{n})$ space. The second contribution is a two-pass algorithm which determines the exact locations of all longest palindromes. It uses the first algorithm as the first pass and uses $O(\sqrt{n})$ space. The third is a one-pass randomized algorithm for the *Longest Palindromic Substring Problem*. It has a *multiplicative* error using $O(\log(n))$ space. We also give two variants of the first algorithm which solve other related practical problems.[1]

Palindrome recognition is important in computational biology. Palindromic structures can frequently be found in proteins and identifying them gives researchers hints about the structure of nucleic acids. For example, in *nucleic acid secondary structure prediction*, one is interested in complementary palindromes which are considered in the full version.

[1] The full version of this paper can be accessed at arXiv:1308.3466.



© Petra Berenbrink, Funda Ergün, Frederik Mallmann-Trenn, and Erfan Sadeqi Azer;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 149–161



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



Related work. While palindromes are well-studied, to the best of our knowledge there are no results for the streaming model. Manacher [5] presents a linear time online algorithm that reports at any time whether all symbols seen so far form a palindrome. The authors of [1] show how to modify this algorithm in order to find all palindromic substrings in linear time (using a parallel algorithm).

Some of the techniques used in this paper have their origin in the streaming pattern matching literature. In the *Pattern Matching Problem*, one tries to find all occurrences of a given pattern P in a text T. The first algorithm for pattern matching in the streaming model was shown in [6] and requires $O(\log(m))$ space. The authors of [3] give a simpler pattern matching algorithm with no preprocessing, as well as a related streaming algorithm for estimating a stream's Hamming distance to p-periodicity. Breslauer and Galil [2] provide an algorithm which does not report false negatives and can also be run in real-time. All of the above algorithms in the string model take advantage of Karp-Rabin fingerprints [4].

Our results. In this paper we present three algorithms, *ApproxSqrt*, *Exact*, and *ApproxLog* for finding palindromes and estimating their length in a given stream S of length n.

We assume that the workspace is bounded while the output space is unlimited.
Given an index m in stream S, $P[m]$ denotes the palindrome of maximal length cantered at index m of S. Our algorithms identify a palindrome $P[m]$ by its *midpoint* m and by its length $\ell(m)$. Our first algorithm outputs all palindromes in S and therefore solves the *Palindrome Problem*.

▶ **Theorem 1** (ApproxSqrt). *For any $\varepsilon \in [1/\sqrt{n}, 1]$* Algorithm ApproxSqrt$(S, \varepsilon)$ *reports for every palindrome $P[m]$ in S its midpoint m as well as an estimate $\tilde{\ell}(m)$ (of $\ell(m)$) such that w.h.p.*[2] *$\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \leq \ell(m)$. The algorithm makes one pass over S, uses $O(n/\varepsilon)$ time, and $O(\sqrt{n}/\varepsilon)$ space.*

The algorithm can easily be modified to report all palindromes $P[m]$ in S with $\ell(m) \geq t$ and no $P[m]$ with $\ell(m) < t - \varepsilon\sqrt{n}$ for some threshold $t \in \mathbb{N}$. For $t \leq \sqrt{n}$ one can modify the algorithm to report a palindrome $P[m]$ if and only if $\ell(m) \geq t$. Note, the algorithm is also $(1+\varepsilon)$-approximative.

Our next algorithm, *Exact*, uses two-passes to solve the *Longest Palindromic Substring Problem*. It uses *ApproxSqrt* as the first pass. In the second pass the algorithm finds the midpoints of all palindromes of length exactly $\ell_{\max}$ where $\ell_{\max}$ is the (initially unknown) length of the longest palindrome in S.

▶ **Theorem 2** (Exact). Algorithm Exact *reports w.h.p. ℓ_{max} and m for all palindromes $P[m]$ with a length of ℓ_{max}. The algorithm makes two passes over S, uses $O(n)$ time, and $O(\sqrt{n})$ space.*

Arguably the most significant contribution of this paper is an algorithm which requires only logarithmic space. In contrast to *ApproxSqrt* (Theorem 1) this algorithm has a multiplicative error and it reports only one of the longest palindromes (see *Longest Palindromic Substring Problem*) instead of all of them due to the limited space.

▶ **Theorem 3** (ApproxLog). *For any ε in $(0, 1]$,* Algorithm ApproxLog *reports w.h.p. an arbitrary palindrome $P[m]$ of length at least $\ell_{max}/(1+\varepsilon)$. The algorithm makes one pass over S, uses $O(\frac{n\log(n)}{\varepsilon\log(1+\varepsilon)})$ time, and $O(\frac{\log(n)}{\varepsilon\log(1+\varepsilon)})$ space.*

[2] We say an event happens *with high probability* (w.h.p.) if its probability is at least $1 - 1/n^c$ for $c \in \mathbb{N}$.

We also show two practical generalisations of our algorithms which can be run simultaneously. These results are presented in the next observation and the next lemma.

▶ Observation 4. For $\ell_{max} \geq \sqrt{n}$, there is an algorithm which reports w.h.p. the midpoints of all palindromes $P[m]$ with $\ell(m) > \ell_{max} - \varepsilon\sqrt{n}$. The algorithm makes one pass over S, uses $O(n/\varepsilon)$ time, and $O(\sqrt{n}/\varepsilon)$ space.

▶ **Lemma 5.** *For $\ell_{max} < \sqrt{n}$, there is an algorithm which reports w.h.p. ℓ_{max} and a $P[m]$ s.t. $\ell(m) = \ell_{max}$. The algorithm makes one pass over S, uses $O(n)$ time, and $O(\sqrt{n})$ space.*

In the full version of the paper we will show an almost matching bound for the additive error of *Algorithm ApproxSqrt*. In more detail, we will show that any randomized one-pass algorithm that approximates the length of the longest palindrome up to an additive error of $\varepsilon\sqrt{n}$ must use $\Omega(\sqrt{n}/\varepsilon)$ space.

2 Model and Definitions

Let $S \in \Sigma^n$ denote the input stream of length n over an alphabet Σ[3]. For simplicity we assume symbols to be positive integers, i.e., $\Sigma \subset \mathbb{N}$. We define $S[i]$ as the symbol at index i and $S[i,j] = S[i], S[i+1], \ldots S[j]$. In this paper we use the streaming model: In one *pass* the algorithm goes over the whole input stream S, reading $S[i]$ in *iteration* i of the pass. In this paper we assume that the algorithm has a memory of size $o(n)$, but the output space is unlimited. We use the so-called word model where the space equals the number of $O(\log(n))$ registers (See [2]).

S contains an *odd palindrome* of length ℓ with midpoint $m \in \{\ell, \ldots, n-\ell\}$ if $S[m-i] = S[m+i]$ for all $i \in \{1, \ldots, \ell\}$. Similarly, S contains an *even palindrome* of length ℓ if $S[m-i+1] = S[m+i]$ for all $i \in \{1, \ldots, \ell\}$. In other words, a palindrome is odd if and only if its length is odd. For simplicity, our algorithms assume palindromes to be even – it is easy to adjust our results for finding odd palindromes by apply the algorithm to $S[1]S[1]S[2]S[2]\cdots S[n]S[n]$ instead of $S[1,n]$.

The maximal palindrome (the palindrome of maximal length) in $S[1,i]$ with midpoint m is called $P[m,i]$ and the maximal palindrome in S with midpoint m is called $P[m]$ which equals $P[m,n]$. We define $\ell(m,i)$ as the maximum length of the palindrome with midpoint m in the substring $S[1,i]$. The maximal length of the palindrome in S with midpoint m is denoted by $\ell(m)$. Moreover, for $z \in \mathbb{Z} \setminus \{1, \ldots, n\}$ we define $\ell(z) = 0$. Furthermore, for $\ell^* \in \mathbb{N}$ we define $P[m]$ to be an ℓ^*-palindrome if $\ell(m) \geq \ell^*$. Throughout this paper, $\tilde{\ell}()$ refers to an estimate of $\ell()$.

We use the KR-Fingerprint, which was first defined by Karp and Rabin [4] to compress strings and was later used in the streaming pattern matching problem (see [6], [3], and [2]). For a string S' we define the forward fingerprint (similar to [2]) and its reverse as follows. $\phi^F_{r,p}(S') = \left(\sum_{i=1}^{|S'|} S'[i] \cdot r^i\right) \mod p$ $\phi^R_{r,p}(S') = \left(\sum_{i=1}^{|S'|} S'[i] \cdot r^{l-i+1}\right) \mod p,<$ where p is an arbitrary prime number in $[n^4, n^5]$ and r is randomly chosen from $\{1, \ldots, p\}$. We write ϕ^F (ϕ^R respectively) as opposed to $\phi^F_{r,p}$ ($\phi^R_{r,p}$ respectively) whenever r and p are fixed. We define for $1 \leq i \leq j \leq n$ the fingerprint $F^F(i,j)$ as the fingerprint of $S[i,j]$, i.e., $F^F(i,j) = \phi^F(S[i,j]) = r^{-(i-1)}(\phi^F(S[1,j]) - \phi^F(S[1,i-1])) \mod p$. Similarly, $F^R(i,j) = \phi^R(S[i,j]) = \phi^R(S[1,j]) - r^{j-i+1} \cdot \phi^R(S[1,i-1]) \mod p$. For every $1 \leq i \leq n - \sqrt{n}$ the

[3] All soundness, space, and time complexity analyses assumes $|\Sigma|$ to be polynomial. One can use a proper random hash function for bigger alphabets.

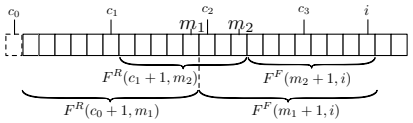


Figure 1 At iteration i two midpoints m_1 and m_2 are checked. Corresponding substrings are denoted by brackets. Note, the distance from c_0 to m_1 equals the distance from m_1 to i. Similarly, the distance from c_1 to m_2 equals the distance from m_2 to i.

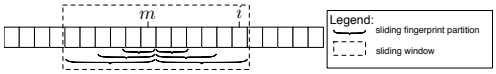


Figure 2 Illustration of the *Fingerprint Pairs* after iteration i of algorithm with $\sqrt{n}=6$, $\varepsilon=1/3$, and $m=i-\sqrt{n}$.

fingerprints $F^F(1,i-1-\sqrt{n})$ and $F^R(1,i-1-\sqrt{n})$ are called *Master Fingerprints*. Note that it is easy to obtain $F^F(i,j+1)$ by adding the term $S[j+1]r^{j+1}$ to $F^F(i,j)$. Similarly, we obtain $F^F(i+1,j)$ by subtracting $S[i]$ from $r^{-1}\cdot F^F(i,j)$. The authors of [2] observe useful properties which we state in the full version.

3 Algorithm Simple ApproxSqrt

In this section, we introduce a simple one-pass algorithm which reports all midpoints and length estimates of palindromes in S. Throughout this paper we use i to denote the current index which the algorithm reads. *Simple ApproxSqrt* keeps the last $2\sqrt{n}$ symbols of $S[1,i]$ in the memory.

It is easy to determine the exact length palindromes of length less than $\sqrt{n}$ since any such palindrome is fully contained in memory at some point. However, in order to achieve a better time bound the algorithm only *approximates* the length of short palindromes. It is more complicated to estimate the length of a palindrome with a length of at least $\sqrt{n}$. However, *Simple ApproxSqrt* detects that its length is at least $\sqrt{n}$ and stores it as an R_S-entry (introduced later) in a list L_i. The R_S-entry contains the midpoint as well as a length estimate of the palindrome, which is updated as i increases.
In order to estimate the lengths of the long palindromes the algorithm designates certain indices of S as *checkpoints*. For every checkpoint c the algorithm stores a fingerprint $F^R(1,c)$ enabling the algorithm to do the following. For every midpoint m of a long palindrome: Whenever the distance from a checkpoint c to m (c occurs before m) equals the distance from m to i, the algorithm compares the substring from c to m to the reverse of the substring from m to i by using fingerprints. We refer to this operation as *checking* $P[m]$ against checkpoint c. If $S[c+1,m]^R=S[m+1,i]$, then we say that $P[m]$ was *sucessfully checked* with c and the algorithm updates the length estimate for $P[m]$, $\tilde{\ell}(m)$. The next time the algorithm possibly updates $\tilde{\ell}(m)$ is after d iterations where d equals the distance between checkpoints. This distance d gives the additive approximation. See Figure 1 for an illustration.

We need the following definitions before we state the algorithm: For $k\in\mathbb{N}$ with $0\le k\le \lfloor\frac{\sqrt{n}}{\varepsilon}\rfloor$ checkpoint c_k is the index at position $k\cdot\lfloor\varepsilon\sqrt{n}\rfloor$ thus checkpoints are $\lfloor\varepsilon\sqrt{n}\rfloor$ indices apart. Whenever we say that an algorithm stores a checkpoint, this means storing the data belonging to this checkpoint. Additionally, the algorithm stores *Fingerprint Pairs*, fingerprints of size $\lfloor\varepsilon\sqrt{n}\rfloor, 2\lfloor\varepsilon\sqrt{n}\rfloor,\ldots$ starting or ending in the middle of the sliding window. In the following, we first describe the data that the algorithm has in its memory after reading $S[1,i-1]$, then we describe the algorithm itself. Let $R_S(m,i)$ denote the representation of $P[m]$ which is stored at time i. As opposed to storing $P[m]$ directly, the algorithm stores m, $\tilde{\ell}(m,i)$, $F^F(1,m)$, and $F^R(1,m)$.

Memory invariants. Just before algorithm *Simple ApproxSqrt* reads $S[i]$ it has stored the following information. Note that, for ease of referencing, during an iteration i data structures are indexed with the iteration number i.
That is, for instance, L_{i-1} is called L_i after $S[i]$ is read.

1. The contents of the sliding window $S[i-2\sqrt{n}-1, i-1]$.
2. The two *Master Fingerprints* $F^F(1, i-1)$ and $F^R(1, i-1)$.
3. A list of *Fingerprint Pairs*: Let r be the maximum integer s.t. $r \cdot \lfloor \varepsilon\sqrt{n} \rfloor < \sqrt{n}$. For $j \in \{\lfloor \varepsilon\sqrt{n} \rfloor, 2 \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \ldots, r \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \sqrt{n}\}$ the algorithm stores the pair $F^R((i-\sqrt{n})-j, (i-\sqrt{n})-1)$, and $F^F(i-\sqrt{n}, (i-\sqrt{n})+j-1)$. See Figure 2 for an illustration.
4. A list CL_{i-1} which consists of all fingerprints of prefixes of S ending at already seen checkpoints, i.e., $\mathrm{CL}_{i-1} = \left[F^R(1, c_1), F^R(1, c_2), \ldots, F^R\left(1, c_{\lfloor (i-1)/\lfloor \varepsilon\sqrt{n} \rfloor \rfloor}\right)\right]$
5. A list L_{i-1} containing representation of all $\sqrt{n}$-palindromes with a midpoint located in $S[1, (i-1)-\sqrt{n}]$. The j^{th} entry of L_{i-1} has the form $R_S(m_j, i-1) = (m_j, \tilde{\ell}(m_j, i-1), F^F(1, m_j), F^R(1, m_j))$ where
   (a m_j is the midpoint of the j^{th} palindrome in $S[1, (i-1)-\sqrt{n}]$ with a length of at least $\sqrt{n}$. Therefore, $m_j < m_{j+1}$ for $1 \le j \le |L_{i-1}| - 1$.
   (b $\tilde{\ell}(m_j, i-1)$ is the current estimate of $\ell(m_j, i-1)$.

In the following, we explain how the algorithm maintains the above invariants.

Maintenance. At iteration i the algorithm performs the following steps. It is implicit that L_{i-1} and $CLi-1$ become L_i and CL_i respectively.

1. Read $S[i]$, set $m = i - \sqrt{n}$. Update the sliding window to $S[m-\sqrt{n}, i] = S[i-2\sqrt{n}, i]$
2. Update the *Master Fingerprints* to be $F^F(1, i)$ and $F^R(1, i)$.
3. If i is a checkpoint (i.e., a multiple of $\lfloor \varepsilon\sqrt{n} \rfloor$), then add $F^R(1, i)$ to CL_i.
4. Update all *Fingerprint Pairs*: For $j \in \{\lfloor \varepsilon\sqrt{n} \rfloor, 2 \cdot \lfloor \varepsilon\sqrt{n} \rfloor, , \ldots, r \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \sqrt{n}\}$
   - Update $F^R(m-j, m-1)$ to $F^R(m-j+1, m)$ and $F^F(m, m+j-1)$ to $F^F(m+1, m+j)$.
   - If $F^R(m-j+1, m) = F^F(m+1, m+j)$, then set $\tilde{\ell}(m, i) = j$.
   - If $\tilde{\ell}(m, i) < \sqrt{n}$, output m and $\tilde{\ell}(m, i)$.
5. If $\tilde{\ell}(m, i) \ge \sqrt{n}$, add *add* $R_S(m, i)$ to L_i: $L_i = L_i \circ (m, \tilde{\ell}(m, i), F^F(1, m), F^R(1, m))$.
6. For all c_k with $1 \le k \le \lfloor \frac{i}{\lfloor \varepsilon\sqrt{n} \rfloor} \rfloor$ and $R_S(m_j, i) \in L_i$ with $i - m_j = m_j - c_k$, check if $\tilde{\ell}(m_j, i)$ can be updated:
   - If the left side of m_j is the reverse of the right side of m_j (i.e., $F^R(c_k+1, m_j) = F^F(m_j+1, i)$) then update $R_S(m_j, i)$ by updating $\tilde{\ell}(m_j, i)$ to $i - m_j$.
7. If $i = n$, then report L_n.

In all proofs in this paper which hold w.h.p. we assume that fingerprints do not fail as we take less than n^2 fingerprints and by using the following Lemma, the probability that a fingerprint fails is at most $1/n^{c+2}$.

▶ **Lemma 6.** *(Theorem 1 of [2]) For two arbitrary strings s and s' with $s \neq s'$ the probability that $\phi^F(s) = \phi^F(s')$ is smaller than $1/n^{c+2}$ for some $c \in \mathbb{N}$.*

Thus, by applying the union bound the probability that no fingerprint fails is at least $1-n^{-c}$. The following lemma shows that the *Simple ApproxSqrt* finds all palindromes along with the estimates as stated in Theorem 1. *Simple ApproxSqrt* does not fulfill the time and space bounds of Theorem 1; we will later show how to improve its efficiency. The proof can be found in the full version.

▶ **Lemma 7.** *For any ε in $[1/\sqrt{n}, 1]$* ApproxSqrt(S, ε) *reports for every palindrome $P[m]$ in S its midpoint m as well as an estimate $\tilde{\ell}(m)$ such that w.h.p. $\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \leq \ell(m)$.*

4 A space-efficient version

In this section, we show how to modify *Simple ApproxSqrt* so that it matches the time and space requirements of Theorem 1. The main idea of the space improvement is to store the lists L_i in a compressed form.

Compression. It is possible in the simple algorithm for L_i to have linear length. In such cases S contains many overlapping palindromes which show a certain *periodic* pattern as shown in Corollary 12, which our algorithm exploits to compress the entries of L_i. This idea was first introduced in [6], and is used in [3], and [2]. More specifically, our technique is a modification of the compression in [2]. In the following, we give some definitions in order to show how to compress the list. First we define a *run* which is a sequence of midpoints of overlapping palindromes.

▶ **Definition 8** (ℓ^*-Run)**.** Let ℓ^* be an arbitrary integer and $h \geq 3$. Let $m_1, m_2, m_3, \ldots, m_h$ be consecutive midpoints of ℓ^*-palindromes in S. $m_1, \ldots, m_h$ form an *ℓ^*-run* if $m_{j+1} - m_j \leq \ell^*/2$ for all $j \in \{1, \ldots, h-1\}$.

In Corollary 12 we show that $m_2 - m_1 = m_3 - m_2 = \cdots = m_h - m_{h-1}$. We say that a run is maximal if the run cannot be extended by other palindromes. More formally:

▶ **Definition 9** (Maximal ℓ^*-Run)**.** An ℓ^*-run over $m_1, \ldots, m_h$ is *maximal* it satisfies both of the following: i) $\ell(m_1 - (m_2 - m_1)) < \ell^*$, ii) $\ell(m_h + (m_2 - m_1)) < \ell^*$.

Simple ApproxSqrt stores palindromes explicitly in L_i, i.e., $L_i = [R_S(m_1, i); \ldots; R_S(m_{|L_i|}, i)]$ where $R_S(m_j, i) = (m_j, \tilde{\ell}(m_j, i), F^F(1, m_j), F^R(1, m_j))$, for all $j \in \{1, 2, \ldots, h\}$. The improved *Algorithm ApproxSqrt* stores these midpoints in a compressed way in list $\hat{L}_i$. *ApproxSqrt* distinguishes among three cases: Those palindromes which

1. are not part of a $\sqrt{n}$-run are stored explicitly as before. We call them R_S-entries. Let $P[m, i]$ be such a palindrome. After iteration i the algorithm stores $R_S(m, i)$.
2. form a maximal $\sqrt{n}$-run are stored in a data structure called R_F-entry. Let $m_1, \ldots, m_h$ be the midpoints of a maximal $\sqrt{n}$-run. The data structure stores the following information.
   - m_1, $m_2 - m_1$, h, $\tilde{\ell}(m_1, i)$, $\tilde{\ell}(m_{\lfloor \frac{1+h}{2} \rfloor}, i)$, $\tilde{\ell}(m_{\lceil \frac{1+h}{2} \rceil}, i)$, $\tilde{\ell}(m_h, i)$,
   - $F^F(1, m_1)$, $F^R(1, m_1)$, $F^F(m_1 + 1, m_2)$, $F^R(m_1 + 1, m_2)$
3. form a $\sqrt{n}$-run which is not maximal (i.e., it can possibly be extended) in a data structure called R_{NF}-entry. The information stored in an R_{NF}-entry is the same as in an R_F-entry, but it does not contain the entries: $\tilde{\ell}(m_{\lfloor \frac{1+h}{2} \rfloor}, i)$, $\tilde{\ell}(m_{\lceil \frac{1+h}{2} \rceil}, i)$, and $\tilde{\ell}(m_h, i)$.

The algorithm stores only the estimate (of the length) and the midpoint of the following palindromes explicitly.

- $P[m]$ for an R_S-entry (Therefore all palindromes which are not part of a $\sqrt{n}$-run)
- $P[m_1]$, $P[m_{\lfloor (h+1)/2 \rfloor}]$, $P[m_{\lceil (h+1)/2 \rceil}]$, and $P[m_h]$ for an R_F-entry
- $P[m_1]$ for an R_{NF}-entry.

In what follows we refer to the above listed palindromes as *explicitly stored* palindromes. We argue in Observation 15 that in any interval of length $\sqrt{n}$ the number of *explicitly stored* palindromes is bounded by a constant.

4.1 Algorithm ApproxSqrt

In this subsection, we describe some modifications of *Simple ApproxSqrt* in order to obtain a space complexity of $O(\frac{\sqrt{n}}{\varepsilon})$ and a total running time of $O(\frac{n}{\varepsilon})$. *ApproxSqrt* is the same as *Simple ApproxSqrt*, but it compresses the stored palindromes. *ApproxSqrt* uses the same memory invariants as *Simple ApproxSqrt*, but it uses $\hat{L}_i$ as opposed to L_i.

ApproxSqrt uses the first four steps of *Simple ApproxSqrt*. Step 5, Step 6, and Step 7 are replaced. The modified Step 5 ensures that there are at most two R_S-entries per interval of length $\sqrt{n}$. Moreover, Step 6 is adjusted since *ApproxSqrt* stores only the length estimate of explicitly stored palindromes.

5. If $\tilde{\ell}(m,i) \geq \sqrt{n}$, obtain $\hat{L}_i$ by adding the palindrome with midpoint $m(= i - \sqrt{n})$ to $\hat{L}_{i-1}$ as follows:

  a. The last element in $\hat{L}_i$ is the following R_{NF}-entry $\big(m_1, m_2 - m_1, h, \tilde{\ell}(m_1,i), F^F(1,m_1), F^R(1,m_1), F^F(m_1+1,m_2), F^R(m_1+1,m_2)\big)$.

  i. If the palindrome can be added to this run, i.e., $m = m_1 + h(m_2 - m_1)$, then we increment the h in the R_{NF}-entry by 1.

  ii. If the palindrome cannot be added: Store $P[m,i]$ as an R_S-entry: $\hat{L}_i = \hat{L}_i \circ (m, \tilde{\ell}(m,i), F^F(1,m), F^R(1,m))$. Moreover, convert the R_{NF}-entry into the R_F-entry by adding $\tilde{\ell}(m_{\lfloor \frac{1+h}{2} \rfloor}, i)$, $\tilde{\ell}(m_{\lceil \frac{1+h}{2} \rceil}, i)$ and $\tilde{\ell}(m_h, i)$: First we calculate $m_{\lfloor \frac{1+h}{2} \rfloor} = m_1 + \big(\lfloor \frac{1+h}{2} - 1 \rfloor\big)(m_2 - m_1)$. One can calculate $m_{\lceil \frac{1+h}{2} \rceil}$ similarly. For $m' \in \{m_{\lfloor \frac{1+h}{2} \rfloor}, m_{\lceil \frac{1+h}{2} \rceil}, m_h\}$ calculate $\tilde{\ell}(m',i) = \max_{i-2\sqrt{n} \leq j \leq i} \{j - m' \mid \exists c_k \text{ with } j - m' = m' - c_k \text{ and } F^R(c_k+1, m') = F^F(m'+1, j)\}$.

  b. The last two entries in $\hat{L}_i$ are stored as R_S-entries and together with $P[m,i]$ form a $\sqrt{n}$-run. Then remove the entries of the two palindromes out of $\hat{L}_{i-1}$ and add a new R_{NF}-entry with all three palindromes to $\hat{L}_{i-1}$: $m_1, \tilde{\ell}(m_1,i), F^F(1,m_1), F^F(1,m_2), F^R(1,m_1), F^R(1,m_2), m_2 - m_1, h = 3$. Retrieve $F^F(m_1+1, m_2)$ and $F^R(m_1+1, m_2)$.

  c. Otherwise, store $P[m,i]$ as an R_S-entry: $\hat{L}_i = \hat{L}_i \circ (m, \tilde{\ell}(m,i), F^F(1,m), F^R(1,m))$

6. This step is similar to step 6 of *Simple ApproxSqrt* the only difference is that we check only for explicitly stored palindromes if they can be extended outwards. [4]

7. If $i = n$. If the last element in $\hat{L}_i$ is an R_{NF}-entry, then convert it into an R_F-entry as in 5(a)ii. Report L_n.

[4] This step is only important for the running time.

4.2 Structural Properties

In this subsection, we prove structural properties of palindromes. These properties allow us to compress (by using R_S-entries and R_F-entries) overlapping palindromes $P[m_1], \ldots, P[m_h]$ in such a way that at any iteration i all the information stored $R_S(m_1, i), \ldots, R_S(m_h, i)$ is available. The structural properties imply, informally speaking, that the palindromes are either far from each other, leading to a small number of them, or they are overlapping and it is possible to compress them. Lemma 11 shows this structure for short intervals containing at least three palindromes. Corollary 12 shows a similar structure for palindromes of a run which is used by *ApproxSqrt*. We first give the common definition of periodicity.

▶ **Definition 10** (*period*)**.** A string S' is said to have period p if it consists of repetitions of a block of p symbols. Formally, S' has period p if $S'[j] = S[j+p]$ for all $j = 1, \ldots, |S'| - p$. [5]

▶ **Lemma 11.** *Let $m_1 < m_2 < m_3 < \cdots < m_h$ be indices in S that are consecutive midpoints of ℓ^*-palindromes for an arbitrary natural number ℓ^*. If $m_h - m_1 \leq \ell^*$, then*

(a) *$m_1, m_2, m_3, \ldots, m_h$ are equally spaced in S, i.e., $|m_2 - m_1| = |m_{k+1} - m_k|$ $\forall k \in \{1, \ldots, h-1\}$*

(b) $S[m_1+1, m_h] = \begin{cases} (ww^R)^{\frac{h-1}{2}} & h \text{ is odd} \\ (ww^R)^{\frac{h-2}{2}}w & h \text{ is even} \end{cases}$, *where $w = S[m_1+1, m_2]$.*

Proof. Given $m_1, m_2, \ldots, m_h$ and ℓ^* we prove the following stronger claim by induction over the midpoints $\{m_1, \ldots, m_j\}$. (a') $m_1, m_2, \ldots, m_j$ are equally spaced. (b') $S[m_1+1, m_j+\ell^*]$ is a prefix of ww^Rww^R... .

Base case $j = 2$: Since we assume m_1 is the midpoint of an ℓ^*-palindrome and $\ell^* \geq m_h - m_1 \geq m_2 - m_1 = |w|$, we have that $S[m_1 - |w| + 1, m_1] = w^R$. Recall that $\ell(m_2) \geq \ell^* \geq |w|$ and thus, $S[m_1+1, m_2+|w|] = ww^R$.

We can continue this argument and derive that $S[m_1+1, m_2+\ell^*]$ is a prefix of $ww^R \ldots ww^R$. (a') for $j = 2$ holds trivially.

Inductive step $j-1 \to j$: Assume (a') and (b') hold up to m_{j-1}. We first argue that $|m_j - m_1|$ is a multiple of $|m_2 - m_1| = |w|$. Suppose $m_j = m_1 + |w| \cdot q + r$ for some integers $q \geq 0$ and $r \in \{1, \ldots, |w| - 1\}$. Since $m_j \leq m_{j-1} + \ell^*$, the interval $[m_1 + 1, m_{j-1} + \ell^*]$ contains m_j. Therefore, by inductive hypothesis, $m_j - r$ is an index where either w or w^R starts. This implies that the prefix of ww^R(or w^Rw) of size $2r$ is a palindrome and the string ww^R(or w^Rw) has period $2r$. On the other hand, by consecutiveness assumption, there is no midpoint of an ℓ^*-palindrome in the interval $[m_1 + 1, m_2 - 1]$. does not have a period of $2p$, a contradiction. We derive that $m_j - m_1$ is multiple of $|w|$.

Hence, we assume $m_j = m_{j-1} + q \cdot |w|$ for some q. The assumption that m_j is a midpoint of an ℓ^*-palindrome beside the inductive hypothesis implies (b') for j. The structure of $S[m_{j-1}+|w|-\ell^*+1, m_{j-1}+|w|+\ell^*]$ shows that $m_{j-1}+|w|$ is a midpoint of an ℓ^*-palindrome. This means that $m_j = m_{j-1} + |w|$. This gives (a') and yields the induction step. ◀

Corollary 12 shows the structure of overlapping palindromes and is essential for the compression. The main difference between Corollary 12 and Lemma 11 is the required distance between the midpoints of a run. Lemma 11 assumes that every palindrome in the run overlaps with all other palindromes. In contrast, Corollary 12 assumes that every palindrome $P[m_j]$ overlaps with $P[m_{j-2}]$, $P[m_{j-1}]$, $P[m_{j+1}]$, and $P[m_{j+2}]$. It can be proven by an induction over the midpoints and using Lemma 11. The proof is in the full version.

[5] Here, p is called a period for S' even if $p > |S'|/2$.

▶ **Corollary 12.** *If $m_1, m_2, \ldots, m_h$ form an ℓ^*-run for an arbitrary natural number ℓ^* then*

(a) *$m_1, m_2, m_3, \ldots, m_h$ are equally spaced in S, i.e., $|m_2 - m_1| = |m_{k+1} - m_k|$ $\forall k \in \{1, \ldots, h-1\}$*

(b) $S[m_1+1, m_h] = \begin{cases} (ww^R)^{\frac{h-1}{2}} & h \text{ is odd} \\ (ww^R)^{\frac{h-2}{2}} w & h \text{ is even} \end{cases}$, *where $w = S[m_1+1, m_2]$.*

Lemma 13 shows the pattern for the lengths of the palindromes in each half of the run. This allows us to only store a constant number of length estimates per run. The proof can be found in the full version.

▶ **Lemma 13.** *At iteration i, let $m_1, m_2, m_3, ..., m_h$ be midpoints of a maximal ℓ^*-run in $S[1, i]$ for an arbitrary natural number ℓ^*. For any midpoint m_j, we have:*

$$\ell(m_j, i) = \begin{cases} \ell(m_1, i) + (j-1) \cdot (m_2 - m_1) & j < \frac{h+1}{2} \\ \ell(m_h, i) + (h-j) \cdot (m_2 - m_1) & j > \frac{h+1}{2} \end{cases}$$

4.3 Analysis of the Algorithm

We show that one can convert R_S-entries into a run and vice versa and *ApproxSqrt*'s maintenance of R_F-entries and R_{NF}-entries does not impair the length estimates. The following lemma shows that one can retrieve the length estimate of a palindrome as well as its fingerprint from an R_F-entry.

▶ **Lemma 14.** *At iteration i, the R_F-entry over $m_1, m_2, \ldots, m_h$ is a lossless compression of $[R_S(m_1, i); \ldots; R_S(m_h, i)]$*

Let *Compressed Run* be the general term for R_F-entry and R_{NF}-entry. We argue that in any interval of length $\sqrt{n}$ we only need to store at most two single palindromes and two *Compressed Runs*. Suppose there were three R_S-entries, then, by Corollary 12, they form a $\sqrt{n}$-run since they overlap each other. Therefore, the three R_S-entries would be stored in a *Compressed Run*. For a similar reason there cannot be more than two *Compressed Runs* in one interval of length $\sqrt{n}$. We derive the following observation.

▶ Observation 15. For any interval of length $\sqrt{n}$ there can be at most two R_S-entries and two *Compressed Runs* in L^*.

We now have what we need in order to prove Theorem 1; the proof is given in the full version.

5 Algorithm Exact

This section describes *Algorithm Exact* which determines the exact length of the longest palindrome in S using $O(\sqrt{n})$ space and two passes over S.

For the first pass this algorithm runs *ApproxSqrt* $(S, \frac{1}{2})$ (meaning that $\varepsilon = 1/2$) and the variant of *ApproxSqrt* described in Lemma 5 simultaneously. The first pass returns ℓ_{max} (Lemma 5) if $\ell_{max} < \sqrt{n}$. Otherwise, the first pass (Theorem 1) returns for every palindrome $P[m]$, with $\ell(m) \geq \sqrt{n}$, an estimate satisfying $\ell(m) - \sqrt{n}/2 < \tilde{\ell}(m) \leq \ell(m)$ w.h.p..

The algorithm for the second pass is determined by the outcome of the first pass. For the case $\ell_{max} < \sqrt{n}$, it uses the sliding window to find all $P[m]$ with $\ell(m) = \ell_{max}$. If $\ell_{max} \geq \sqrt{n}$, then the first pass only returns an additive $\sqrt{n}/2$-approximation of the palindrome lengths. We define the *uncertain intervals* of $P[m]$ to be: $I_1(m) = S[m - \tilde{\ell}(m) - \sqrt{n}/2 + 1, m - \tilde{\ell}(m)]$ and $I_2(m) = S[m + \tilde{\ell}(m) + 1, m + \tilde{\ell}(m) + \sqrt{n}/2]$.

The algorithm uses the length estimate calculated in the first pass to delete all R_S-entries (Step 3) which cannot be the longest palindromes. Similarly, the algorithm (Step 2) only

keeps the middle entries of R_F-entries since these are the longest palindromes of their run (A proof can be found in the full version). In the second pass, *Algorithm Exact* stores $I_1(m)$ for a palindrome $P[m]$ if it was not deleted. *Algorithm Exact* compares the symbols of $I_1(m)$ symbol by symbol to $I_2(m)$ until the first mismatch is found. Then the algorithm knows the exact length $\ell(m)$ and discards $I_1(m)$. The analysis will show, at any time the number of stored uncertain intervals is bounded by a constant.

First Pass. Run the following two algorithms simultaneously:

1. *ApproxSqrt* $(S, 1/2)$. Let L be the returned list.
2. Variant of *ApproxSqrt* (See Lemma 5) which reports ℓ_{max} if $\ell_{max} < \sqrt{n}$.

Second Pass

- $\ell_{max} < \sqrt{n}$: Use a sliding window of size $2\sqrt{n}$ and maintain two fingerprints $F^R[i - \sqrt{n} - \ell_{max} + 1, i - \sqrt{n}]$, and $F^F[i - \sqrt{n} + 1, i - \sqrt{n} + \ell_{max}]$. Whenever these fingerprints match, report $P[i - \sqrt{n}]$.
- $\ell_{max} \geq \sqrt{n}$: In this case, the algorithm uses a preprocessing phase first.

Preprocessing

1. Set $\tilde{\ell}_{max} = \max\{\tilde{\ell}(m) \mid P[m]$ is stored in L as an R_F or an R_S entry$\}$.
2. For every R_F-entry R_F in L with midpoints $m_1, \ldots, m_h$ remove R_F from L and add $R_s(m, i) = (m, \tilde{\ell}(m), F^F(1, m), F^R(1, m))$ to L, for $m \in \{m_{\lfloor (h+1)/2 \rfloor}, m_{\lceil (h+1)/2 \rceil}\}$. To do this, calculate $m_{\lfloor \frac{1+h}{2} \rfloor} = m_1 + (\lfloor \frac{1+h}{2} \rfloor - 1)(m_2 - m_1)$ and $m_{\lceil \frac{1+h}{2} \rceil} = m_1 + (\lceil \frac{1+h}{2} \rceil - 1)(m_2 - m_1)$. Retrieve $F^F(1, m)$ and $F^R(1, m)$ for $m \in \{m_{\lfloor (h+1)/2 \rfloor}, m_{\lceil (h+1)/2 \rceil}\}$.
3. Delete all R_S-entries $(m_k, \tilde{\ell}(m_k), F^F(1, m_k), F^R(1, m_k))$ with $\tilde{\ell}(m_k) \leq \tilde{\ell}_{max} - \sqrt{n}/2$ from L.
4. For every palindrome $P[m] \in L$ set $I_1(m) := (m - \tilde{\ell}(m) - 1/2\sqrt{n}, m - \tilde{\ell}(m)]$ and set $finished(m) :=$ false.

The resulting list is called L^*.

String processing. At iteration i the algorithm performs the following steps.

1. Read $S[i]$. If there is a palindrome $P[m]$ such that $i \in I_1(m)$, then store $S[i]$.
2. If there is a midpoint m such that $m + \tilde{\ell}(m) < i < m + \tilde{\ell}(m) + \frac{\sqrt{n}}{2}$, finished$(m)$ = false, and $S[m - (i - m) + 1] \neq S[i]$, then set finished$(m)$:= true and $\ell(m) = i - m - 1$.
3. If there is a palindrome $P[m]$ such that $i \geq \tilde{\ell}(m) + m + \frac{\sqrt{n}}{2}$, then discard $I_1(m)$.
4. If $i = n$, then output $\ell(m)$ and m of all $P[m]$ in L^* with $\ell(m) = \ell_{max}$.

We analyse *Exact* in the full version.

6 Algorithm ApproxLog

In this section, we present an algorithm which reports one of the longest palindromes and uses only logarithmic space. *ApproxLog* has a multiplicative error instead of an additive error term. Similar to *ApproxSqrt* we have special indices of S designated as checkpoints that we keep along with some constant size data in memory. The checkpoints are used to estimate the length of palindromes. However, this time checkpoints (and their data) are only stored for a limited time. Since we move from additive to multiplicative error we do not need checkpoints to be spread evenly in S. At iteration i, the number of checkpoints in any

interval of fixed length decreases exponentially with distance to i. The algorithm stores a palindrome $P[m]$ (as an R_S-entry or R_{NF}-entry) until there is a checkpoint c such that $P[m]$ was checked unsuccessfully against c. A palindrome is stored in the lists belonging to the last checkpoint with which is was checked successfully. In what follows we set $\delta \triangleq \sqrt{1+\varepsilon} - 1$ for the ease of notation. Every checkpoint c has an attribute called $level(c)$. It is used to determine the number of iterations the checkpoint data remains in the memory.

Memory invariants. After algorithm *ApproxLog* has processed $S[1, i-1]$ and before reading $S[i]$ it contains the following information:

1. Two *Master Fingerprints* up to index $i-1$, i.e., $F^F(1, i-1)$ and $F^R(1, i-1)$.
2. A list of checkpoints CL_{i-1}. For every $c \in \mathrm{CL}_{i-1}$ we have
   - $level(c)$ such that c is in CL_{i-1} iff $c \geq (i-1) - 2(1+\delta)^{level(c)}$.
   - $fingerprint(c) = F^R(1, c)$
   - a list L_c. It contains all palindromes which were successfully checked with c, but with no other checkpoint $c' < c$. The palindromes in L_c are either R_S-entries or R_{NF}-entries (See *Algorithm ApproxSqrt*).
3. The midpoint m^*_{i-1} and the length estimate $\tilde{\ell}(m^*_{i-1}, i-1)$ of the longest palindrome found so far.

The algorithm maintains the following property. If $P[m, i]$ was successfully checked with checkpoint c but with no other checkpoint $c' < c$, then the palindrome is stored in L_c. The elements in L_c are ordered in increasing order of their midpoint. The algorithm stores palindromes as R_S-entries and R_{NF}-entries. This time however, the length estimates are not maintained. Adding a palindrome to a current run works exactly (the length estimate is not calculated) as described in *Algorithm ApproxSqrt*.

Maintenance. At iteration i the algorithm performs the following steps.

1. Read S[i]. Update the *Master Fingerprints* to be $F^F(1, i)$ and $F^R(1, i)$.
2. For all $k \geq k_0 = {}^{\log(1/\delta)}/_{\log(1+\delta)}$(The algorithm does not maintain intervals of size 0.)
   - **a.** If i is a multiple of $\lfloor \delta(1+\delta)^{k-2} \rfloor$, then add the checkpoint $c = i$ (along with the checkpoint data) to CL_i. Set $level(c) = k$, $fingerprint(c) = F^R(1, i)$ and $L_c = \emptyset$.
   - **b.** If there exists a checkpoint c with $level(c) = k$ and $c < i - 2(1+\delta)^k$, then prepend L_c to $L_{c'}$ where $c' = max\{c'' \mid c'' \in \mathrm{CL}_i \text{ and } c'' > c\}$. Merge and create runs in L_c if necessary (Similar to step 5 of *ApproxSqrt*). Delete c and its data from CL_i.
3. For every checkpoint $c \in \mathrm{CL}_i$
   - **a.** Let m_c be the midpoint of the first entry in L_c and $c' = max\{c'' \mid c'' \in \mathrm{CL}_i \text{ and } c'' < c\}$. If $i - m_c = m_c - c'$, then we *check* $P[m]$ against c' by doing the following:
     - **i.** If the left side of m_c is the reverse of the right side of m_c (i.e., $F^R(c', m_c) = F^F(m_c, i)$) then move $P[m_c]$ from L_c to $L_{c'}$ by adding $P[m_c]$ to $L_{c'}$:
       - **A.** If $|L_{c'}| \leq 1$, store $P[m_c]$ as a R_S-entry.
       - **B.** If $|L_{c'}| = 2$, create a run out of the R_S-entries stored in $L_{c'}$ and $P[m_c]$.
       - **C.** Otherwise, add $P[m_c]$ to the R_{NF}-entry in $L_{c'}$.
     - **ii.** If the left side of m_c is *not* the reverse of the right side of m_c, then remove m_c from L_c.
     - **iii.** If $i - m_c > \tilde{\ell}(m^*_i)$, then set $m^*_i = m_c$ and set $\tilde{\ell}(m^*_i) = i - m_c$.
4. If $i = n$, then report m^*_i and $\tilde{\ell}(m^*_i)$.

6.1 Analysis

ApproxLog relies heavily on the interaction of the following two ideas. The pattern of the checkpointing and the compression which is possible due to the properties of overlapping palindromes (Lemma 11). On the one hand the checkpoints are close enough so that the length estimates are accurate (Lemma 19). The closeness of the checkpoints ensures that palindromes which are stored at a checkpoint form a run (Lemma 18) and therefore can be stored in constant space. On the other hand the checkpoints are far enough apart so that the number of checkpoints and therefore the required space is logarithmic in n.

We start off with an observation to characterise the checkpointing. Step 2 of the algorithm creates a checkpoint pattern: Recall that the level of a checkpoint is determined when the checkpoint and its data are added to the memory. The checkpoints of every level have the same distance. A checkpoint (along with its data) is removed if its distance to i exceeds a threshold which depends on the level of the checkpoint. Note that one index of S can belong to different levels and might therefore be stored several times. The following observation follows from Step 2 of the algorithm.

▶ Observation 16. At iteration i, $\forall k \geq k_0 = \left\lceil \frac{log(\frac{(1+\delta)^2}{\delta})}{log(1+\delta)} \right\rceil$. Let $C_{i,k} = \{c \in \mathrm{CL}_i \mid level(c) = k\}$.
1. $C_{i,k} \subseteq [i - 2(1+\delta)^k, i]$.
2. The distance between two consecutive checkpoints of $C_{i,k}$ is $\lfloor \delta(1+\delta)^{k-2} \rfloor$.
3. $|C_{i,k}| = \left\lceil \frac{2(1+\delta)^k}{\lfloor \delta(1+\delta)^{k-2} \rfloor} \right\rceil$.

This observation can be used to calculate the size of the checkpoint data which the algorithm stores at any time. The proof can also be found in the full version.

▶ **Lemma 17.** *At Iteration i of the algorithm the number of checkpoints is in $O\left(\frac{\log(n)}{\varepsilon \log(1+\varepsilon)}\right)$.*

The space bounds of Theorem 3 hold due to the following property of the checkpointing: If there are more than three palindromes stored in a list L_c for checkpoint c, then the palindromes form a run and can be stored in constant space as the following lemma shows.

▶ **Lemma 18.** *At iteration i, let $c \in CL_i$ be an arbitrary checkpoint. The list L_c can be stored in constant space.*

Proof. We fix an arbitrary $c \in \mathrm{CL}_i$. For the case that there are less than three palindromes belonging to L_c, they can be stored as R_S-entries in constant space. Therefore, we assume the case where there are at least three palindromes belonging to L_c and we show that they form a run. Let c' be the highest (index) checkpoint less than c, i.e., $c' = max\{c'' \mid c'' \in \mathrm{CL}_i \text{ and } c'' < c\}$. We disregard the case that the index of c is 1. Let k be the minimum value such that $(1+\delta)^{k-1} < i - c \leq (1+\delta)^k$. Recall that L_c is the list of palindromes which the algorithm has successfully checked against c and not against c' yet. Let $P[m]$ be a palindrome in L_c. Since it was successfully checked against c we know that $i - m \geq m - c$. Similarly, since $P[m]$ was not checked against c' we have $i - m < m - c'$. Thus, for every $P[m]$ in L_c we have $\frac{i+c'}{2} < m \leq \frac{i+c}{2}$. Therefore, all palindromes stored in L_c are in an interval of length less than $\frac{i+c}{2} - \frac{i+c'}{2} = \frac{c-c'}{2}$. If we show that $\ell(m) \geq \frac{c-c'}{2}$ for all $P[m]$ in L_c, then applying Lemma 11 with $\ell^* = \frac{c-c'}{2}$ on the palindromes in L_c implies that they are forming a run. The run can be stored in constant space in an R_{NF}-entry. Therefore, it remains to show that $\ell(m) \geq \frac{c-c'}{2}$. We first argue the following: $c - c' \underset{Obs.\ 16}{\leq} \delta(1+\delta)^{k-2} \underset{\delta \leq 1}{\leq} \frac{(1+\delta)^{k-1}}{2} \underset{Def.\ of\ k}{<} \frac{i-c}{2}$. Since $P[m]$ was successfully checked against c and since $m > \frac{i+c'}{2}$ we derive that $\ell(m) > \frac{i+c'}{2} - c$. Therefore, $\ell(m) > \frac{i+c'}{2} - c = \frac{i-c}{2} + \frac{c'-c}{2} \underset{(6.1)}{>} c - c' + \frac{c'-c}{2} = \frac{c-c'}{2}$. ◀

The following lemma shows that the checkpoints are sufficiently close in order to satisfy the multiplicative approximation. The proof is in the full version.

▶ **Lemma 19.** ApproxLog *reports a midpoint* m^* *such that w.h.p.* $\frac{\ell_{max}}{(1+\varepsilon)} \leq \tilde{\ell}(m^*) \leq \ell_{max}$.

We are ready to prove Theorem 3. The correctness follows from Lemma 19. Lemma 17 and Lemma 18 yield the claimed space. In every iteration the algorithm processes every checkpoint in CL_i in constant time. The number of checkpoints is bounded by Lemma 17. A full proof can be found in the full version.

References

1 Alberto Apostolico, Dany Breslauer, and Zvi Galil. Parallel detection of all palindromes in a string. *Theoretical Computer Science*, 141(1–2):163–173, 1995.

2 Dany Breslauer and Zvi Galil. Real-time streaming string-matching. In Raffaele Giancarlo and Giovanni Manzini, editors, *Combinatorial Pattern Matching*, volume 6661 of *LNCS*, pages 162–172. Springer Berlin Heidelberg, 2011.

3 Funda Ergün, Hossein Jowhari, and Mert Sağlam. Periodicity in streams. In *Proceedings of the 13th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, and 14th International Workshop on Randomization and Computation (APPROX/RANDOM'10)*, volume 6302 of *LNCS*, pages 545–559, Berlin, Heidelberg, 2010. Springer-Verlag.

4 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, March 1987.

5 Glenn Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, July 1975.

6 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS'09, pages 315–323, Washington, DC, USA, 2009. IEEE Computer Society.

New Bounds and Extended Relations Between Prefix Arrays, Border Arrays, Undirected Graphs, and Indeterminate Strings*

Francine Blanchet-Sadri[1], Michelle Bodnar[2], and Benjamin De Winkle[3]

1 **Department of Computer Science, University of North Carolina, Greensboro, USA**
`blanchet@uncg.edu`
2 **Department of Mathematics, University of California, San Diego, USA**
`mbodnar@ucsd.edu`
3 **Department of Mathematics, Tufts University, Medford, USA**
`benjamin.de_winkle@tufts.edu`

Abstract

We extend earlier works on the relation of prefix arrays of indeterminate strings to undirected graphs and border arrays. If integer array y is the prefix array for indeterminate string w, then we say w satisfies y. We use a graph theoretic approach to construct a string on a minimum alphabet size which satisfies a given prefix array. We relate the problem of finding a minimum alphabet size to finding edge clique covers of a particular graph, allowing us to bound the minimum alphabet size by $n + \sqrt{n}$ for indeterminate strings, where n is the size of the prefix array. When we restrict ourselves to prefix arrays for partial words, we bound the minimum alphabet size by $\lceil\sqrt{2n}\rceil$. Moreover, we show that this bound is tight up to a constant multiple by using Sidon sets. We also study the relationship between prefix arrays and border arrays. We show that the slowly-increasing property completely characterizes border arrays for indeterminate strings, whence there are exactly C_n distinct border arrays of size n for indeterminate strings (here C_n is the nth Catalan number). We also bound the number of prefix arrays for partial words of a given size using Stirling numbers of the second kind.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Indeterminate strings, Partial words, Prefix arrays, Border arrays, Undirected graphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.162

1 Introduction

Strings are sequences of letters from a given alphabet. They have been extensively studied and several generalizations have been proposed in the literature which include *indeterminate strings* and *strings with don't cares* [10, 1, 3]. An indeterminate string allows positions to be subsets of cardinality greater than one of a given alphabet, while a string with don't cares allows positions to be the given alphabet. For example, $a\{a,b\}bb\{a,c\}$ is an indeterminate string of length 5 on the alphabet $\{a,b,c\}$ and $a\{a,b,c\}bb\{a,b,c\}$ is a string with don't cares

* This material is based upon work supported by the National Science Foundation under Grant No. DMS–1060775.



© Francine Blanchet-Sadri, Michelle Bodnar, and Benjamin De Winkle; licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 162–173



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



of same length on that alphabet. Strings with don't cares are also referred to as *partial words* and the don't care symbol is often represented by the $\diamond$ symbol, or *hole* symbol, which represents the alphabet. An alternative way of writing our example $a\{a,b,c\}bb\{a,b,c\}$ is $a\diamond bb\diamond$. Strings where each position is a singleton subset are referred to as *regular strings*.

The fundamental concept of *border array* has played an important role in pattern matching for over four decades [6, 15]. If non-empty strings u_1, u_2, v_1, v_2 exist such that $w = u_1v_1 = v_2u_2$ and u_1 matches u_2, denoted by $u_1 \approx u_2$, then string w has a *border* of length $|u_1| = |u_2|$. The border array β corresponding to a string w of length n is an integer array of same length such that for $j \in 0..n-1$, $\beta[j]$ is the length of the longest border of $w[0..j]$. For example, $a\{a,b\}bb\{a,c\}$ and $a\diamond bb\diamond$ give rise to the border arrays 01231 and 01234, respectively.

For a regular string w, any border of a border of w is also a border of w; thus w's border array gives all the borders of every prefix of w. This desirable property is lost however, when we consider indeterminate strings or partial words, due to the lack of the transitivity of $\approx$ (e.g., $a \approx \{a,b\}$ and $\{a,b\} \approx b$, but $a \not\approx b$ implying that $w = a\{a,b\}b$ has a border of length 2 having a border of length 1, but w has no border of length 1). Smyth and Wang [16] showed that for indeterminate strings, the concept of *prefix array* provides more information than the one of border array and specifies all the borders of every prefix. The prefix array y corresponding to a string w of length n is an integer array of same length such that for $j \in 0..n-1$, $y[j]$ is the length of the longest prefix of $w[j..n)$ that matches a prefix of w. For example, $a\{a,b\}bb\{a,c\}$ and $a\diamond bb\diamond$ give rise to the prefix arrays 53001 and 54001, respectively. Main and Lorentz [13] described the first algorithm for computing the prefix array of any given regular string as a routine in their well-known algorithm for finding all repetitions in a regular string, and Smyth and Wang [16] described an algorithm for efficiently computing the prefix array of any given indeterminate string.

The reverse problem of the one of designing an algorithm that computes the prefix array of any given string is the problem of designing an algorithm that tests if an integer array is the prefix array of some string and, if so, constructs such a string. Clément et al. [5] described an $O(n)$ time algorithm that tests if an integer array of size n is the prefix array of some regular string and, if so, constructs the lexicographically least string having it as a prefix array, the alphabet size of the string being bounded by $\log_2 n$. Recently, Christodoulakis et al. [4] described an algorithm for computing an indeterminate string corresponding to a given feasible prefix array. Such algorithmic characterizations of prefix arrays are not only interesting from a theoretical point of view, but also from a practical point of view, e.g., they help in the design of methods for randomly generating prefix arrays for software testing.

Christodoulakis et al. [4] established quite unexpected connections between indeterminate strings, prefix arrays, and undirected graphs. Among them, they proved the surprising result that every feasible array is the prefix array of some string. In this paper, we extend connections between indeterminate strings, prefix/border arrays, and undirected graphs, which yield combinatorial insights. In Section 2, we review some basics. In Section 3, we revisit the problem of constructing an indeterminate string on a minimal alphabet satisfying a given feasible prefix array y. We describe two methods: the first one relies on a graph $\mathcal{Q}_y$ built from y's associated prefix graph $\mathcal{P}_y$, while the second one examines induced subgraphs of $\mathcal{P}_y$. It turns out that the minimum alphabet size is the chromatic number of $\mathcal{Q}_y$ and is also the size of the smallest induced positive edge cover of $\mathcal{P}_y$. We bound the minimum alphabet size for an array of size n by $n + \sqrt{n}$ using results of Alon, Erdős et al., and Lovász on edge clique covers. In Section 4, we explore the relationship between prefix arrays and border arrays. In particular, we show that every slowly-increasing array is the border array for some indeterminate string, whence the number of border arrays of size n for indeterminate strings

is the nth Catalan number. In Section 5, we restrict prefix arrays to partial words. We give a characterization of such prefix arrays y in terms of the prefix graph $\mathcal{P}_y$. Moreover, we give a method to construct a partial word on the smallest possible alphabet for a given prefix array. We bound the minimum alphabet size for an array of size n by $\left\lceil \sqrt{2n} \right\rceil$, this bound being tight (up to a constant multiple) using results of Erdős and Túran on Sidon sets. We also bound the number of prefix arrays of a given size valid for partial words using Stirling numbers of the second kind. Finally in Section 6, we conclude with some suggestions for future work.

2 Preliminaries

Throughout the paper, we use many graph theoretical concepts and constructions. We refer the reader to [11] for an introduction to these ideas.

A *string* w on alphabet A is a sequence of non-empty subsets of A, or may be empty. If A has cardinality μ, we also say that w is a string on μ letters. We call a 1-element subset of A a *regular* letter and larger subsets *indeterminate* letters. A string of all regular letters is called a *regular string* (also referred to as a *word*), and a string which contains at least one indeterminate letter is called an *indeterminate string*. A *hole*, denoted by $\diamond$, is an indeterminate letter which consists of the full alphabet, A. A *partial word* is a string which consists of only regular letters and holes. We denote the length of string w by $|w|$.

Two non-empty subsets of A, α and α', *match* if they have non-empty intersection. We denote this by $\alpha \approx \alpha'$. Similarly, two strings w and w' *match* if $|w| = |w'|$ and $w[i] \approx w'[i]$ for each $i \in 0..|w|-1$. As before, this is denoted by $w \approx w'$.

An integer array y of size n such that $y[0] = n$ and for every $i \in 1..n-1$, $0 \leq y[i] \leq n-i$, is called *feasible*. The *prefix array* of a string w of length n is an array of integers y such that $y[j]$ is the length of the longest prefix of $w[j..n)$ that matches a prefix of w. Note that $y[0]$ is the size of y for any prefix array y. If y is the prefix array of some regular string, then y is called *regular*. If y is the prefix array of some partial word, then y is called *valid for partial words*. If y is the prefix array of a string w, then w *satisfies* y.

▶ **Lemma 2.1.** *[4] An integer array y of size n is the prefix array of a string w of length n if and only if for each position $i \in 0..n-1$, the following two conditions hold: (1) $w[0..y[i]) \approx w[i..i+y[i])$ and (2) if $i + y[i] < n$, then $w[y[i]] \not\approx w[i+y[i]]$.*

The most important graph construction that we use is that of the *prefix graph*, which is introduced in [4]. The prefix graph of a prefix array, y, of size n is denoted by $\mathcal{P}_y$ and is constructed as follows. Its vertex set, $V(\mathcal{P}_y)$, is $[0..n)$. Its edge set consists of two types of edges. Let $i \in 1..n-1$. For $j \in 0..y[i]-1$, $\{j, i+j\}$ is a *positive edge*, while for $i + y[i] < n$, $\{y[i], i+y[i]\}$ is a *negative edge* (refer to Lemma 2.1).

Let $E^+(\mathcal{P}_y)$ be the set of positive edges of $\mathcal{P}_y$ and $E^-(\mathcal{P}_y)$ be the set of negative edges of $\mathcal{P}_y$ (note we may write just E^+ or E^-, respectively, when $\mathcal{P}_y$ is clear from context). We write $\mathcal{P}_y^+ = (V(\mathcal{P}_y), E^+(\mathcal{P}_y))$ (i.e., the graph with the same vertex set, but with only the positive edges) and $\mathcal{P}_y^- = (V(\mathcal{P}_y), E^-(\mathcal{P}_y))$ (same vertex set, only the negative edges). A string w *satisfies* negative edge $\{i, j\}$ if $w[i] \not\approx w[j]$ and w *violates* $\{i, j\}$ if $w[i] \approx w[j]$. Similarly, w *satisfies* positive edge $\{i, j\}$ if $w[i] \approx w[j]$ and w *violates* $\{i, j\}$ if $w[i] \not\approx w[j]$. The graph $\mathcal{P}_y$ encodes all the information of the prefix array y, that is to say, that string w satisfies y if and only if w satisfies all positive and negative edges of $\mathcal{P}_y$. Figure 1 shows an example.

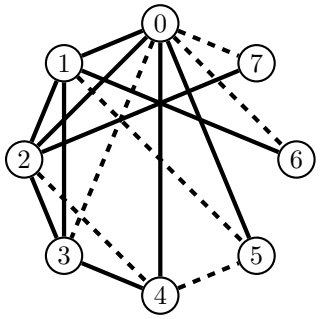


Figure 1 Prefix graph $\mathcal{P}_y$ for $y = 84201300$. Solid lines indicate positive edges and dashed lines indicate negative edges.

▶ **Lemma 2.2.** *Let y be a feasible prefix array and w be an indeterminate string satisfying y. If in $\mathcal{P}_y$, j_0 and j_k are joined by a negative edge and $j_0, j_1, \ldots, j_k$ is a path on only positive edges, then there exists some $i \in 0..k$ such that $w[j_i]$ is an indeterminate letter.*

3 Constructing Indeterminate Strings for Prefix Arrays

Returning to Figure 1, $\{a,c,e\}\{a,b\}\{a,b\}\{b,d\}\{c,d\}ebb$ satisfies the prefix array $y = 84201300$. It is constructed on an alphabet of five letters a, b, c, d, e. Is the alphabet size minimal? The answer is no since $\{a,b\}\{a,c\}\{b,c\}\{c,d\}\{a,d\}bcc$ also satisfies y but is constructed using only the four letters a, b, c, d. In this section, we describe two methods for constructing indeterminate strings on a minimum alphabet size that satisfy a given prefix array. For ease of notation, if y is a feasible prefix array, let $\mu(y)$ denote the minimum alphabet size for an indeterminate string that satisfies y.

Let us describe our first method. Let V^+ be the set of vertices of $\mathcal{P}_y$ which are incident to a positive edge. We construct a new graph $\mathcal{Q}$ from $\mathcal{P}_y$ as follows: $V(\mathcal{Q}) = E^+ \cup \{\{i,i\} \mid i \in V \setminus V^+\}$, and $\{\{i_1,j_1\},\{i_2,j_2\}\} \in E(\mathcal{Q})$ if and only if there exists some $\{r,s\} \in E^-$ such that $r \in \{i_1,j_1\}$ and $s \in \{i_2,j_2\}$. Since a prefix array y defines a unique $\mathcal{P}_y$, which in turn defines a unique $\mathcal{Q}$, we can call this graph $\mathcal{Q}_y$. We show how proper colorings of $\mathcal{Q}_y$ and indeterminate strings satisfying y are related.

Figure 2 gives an example. Since $\mathcal{Q}_y$ has chromatic number 2, associate a with vertices $\{0,2\}$ and $\{0,3\}$, and b with vertices $\{1,3\}$, $\{1,4\}$, and $\{5,5\}$. By assigning letters to each vertex in $\mathcal{P}_y$ corresponding to the letters associated with its incident positive edges, we obtain the indeterminate $\{a\}\{b\}\{a\}\{a,b\}\{b\}\{b\}$ which satisfies 602200.

▶ **Theorem 3.1.** *Let μ be the minimum alphabet size for a string satisfying a given feasible prefix array y. Then $\chi(\mathcal{Q}_y) = \mu$, where $\chi(\mathcal{Q}_y)$ denotes the chromatic number of $\mathcal{Q}_y$.*

Proof. Let $\mathcal{P} = \mathcal{P}_y$ and $\mathcal{Q} = \mathcal{Q}_y$. Suppose w is a string on μ letters which satisfies y, and associate a distinct color to each letter. For each edge $\{i,j\} \in E^+ \cup \{\{i,i\} \mid i \in V \setminus V^+\}$, color the vertex $\{i,j\}$ in $\mathcal{Q}$ with the color associated to the first element in $w[i] \cap w[j]$. The intersection is always non-empty because positive edges and loops represent matchings. Now suppose there is an edge connecting the vertices $\{i,j\}$ and $\{r,s\}$ in $\mathcal{Q}$. Then there is a negative edge in $\mathcal{P}$ connecting one endpoint of $\{i,j\}$ to an endpoint of $\{r,s\}$. Without loss of generality, assume $\{i,r\} \in E^-$. This implies $w[i] \cap w[r] = \emptyset$, so $\{i,j\}$ and $\{r,s\}$ must have different colors. Thus, we have obtained a proper coloring of $\mathcal{Q}$ with μ colors, so $\chi(\mathcal{Q}) \leq \mu$.

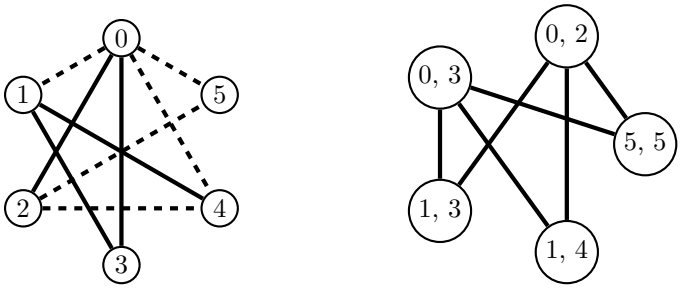


Figure 2 $\mathcal{P}_y$ (left) and $\mathcal{Q}_y$ (right) for the prefix array $y = 602200$, where solid lines indicate positive edges and dashed lines indicate negative edges.

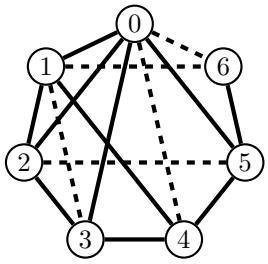


Figure 3 $\mathcal{P}_y$ for $y = 7612010$, where solid lines indicate positive edges and dashed lines indicate negative edges. One IPEC for $\mathcal{P}_y$ is given by the sets $V_0 = \{0, 1, 5\}$, $V_1 = \{0, 2, 3\}$, $V_2 = \{1, 2, 4\}$, and $V_3 = \{3, 4, 5, 6\}$. The construction in Theorem 3.2 gives the indeterminate string $w = \{a, b\}\{a, c\}\{b, c\}\{b, d\}\{c, d\}\{a, d\}d$, which satisfies y.

Now suppose we are given a proper coloring of $\mathcal{Q}$ using $\chi(\mathcal{Q})$ colors. We may think of each color as a letter and construct an indeterminate string w by assigning to $w[i]$ the color of each $\{k, j\} \in V(Q)$ such that $i = k$ or $i = j$. Let $\{i, j\}$ be any positive edge of $\mathcal{P}$. Then $w[i]$ and $w[j]$ both contain the color given to $\{i, j\}$, so it is satisfied. It remains to show that each negative edge is also satisfied. Suppose $\{i, j\} \in E^-$. Then $w[i] = \{c \mid c \text{ is the color on some } \{i, r\} \in E^+\}$, and $w[j] = \{c \mid c \text{ is the color on some } \{j, s\} \in E^+\}$. Moreover, if $\{i, r\}, \{j, s\} \in E^+$, then they are connected by an edge in $\mathcal{Q}_y$, so they have a different color. Hence $w[i] \cap w[j] = \emptyset$, so $\{i, j\}$ is satisfied. Therefore, w satisfies y. Moreover, w uses at most $\chi(\mathcal{Q})$ letters, which implies $\mu \leq \chi(\mathcal{Q})$. ◀

Let us describe our second method. Suppose indeterminate string w on alphabet $A = \{a_0, a_1, \ldots, a_{\mu-1}\}$ satisfies prefix array y, and define $V_i = \{j \mid a_i \in w[j]\}$. Notice that the subgraph of $\mathcal{P}_y$ induced by V_i contains no negative edges. Moreover, each positive edge is in the subgraph induced by some V_i. This observation motivates the following definitions.

A subgraph of $\mathcal{P}_y$ is *negative-free* if it does not contain any negative edges. We use the notation $\mathcal{P}_y[V_i]$ to denote the subgraph of $\mathcal{P}_y$ induced by V_i. A set $\{V_0, V_1, \ldots, V_k\}$, where $V_i \subset V(\mathcal{P}_y)$, is an *induced positive edge cover* (IPEC) of $\mathcal{P}_y$ if $\mathcal{P}_y[V_i]$ is negative-free for all $i \in 0..k$, each positive edge of $\mathcal{P}_y$ is in some $\mathcal{P}_y[V_i]$, and each vertex of $\mathcal{P}_y$ is in some $\mathcal{P}_y[V_i]$. Figure 3 gives an example of an IPEC.

▶ **Theorem 3.2.** *Let y be a feasible prefix array. The minimum alphabet size for an indeterminate string satisfying y is exactly the size of the smallest IPEC of $\mathcal{P}_y$.*

Proof. Let μ be the minimum alphabet size for an indeterminate string satisfying y, and let σ be the size of the smallest IPEC of $\mathcal{P}_y$. Suppose w is an indeterminate string that satisfies y

on the alphabet $\{a_0, a_1, \ldots, a_{\mu-1}\}$. Let V_i be as defined above. We claim $\{V_0, V_1, \ldots, V_{\mu-1}\}$ is an IPEC of $\mathcal{P}_y$. It is clear that each vertex is in some V_i, because each position of w is non-empty. Suppose $\{i, j\}$ is a negative edge of $\mathcal{P}_y$. Since w satisfies y, it must satisfy $\{i, j\}$, so $w[i] \cap w[j] = \emptyset$. Hence there is no V_k which contains both i and j, which implies $\{i, j\}$ is not in $\mathcal{P}_y[V_k]$ for any k. This holds for any negative edge, so each $\mathcal{P}_y[V_k]$ is negative-free. Now suppose $\{i, j\}$ is a positive edge of $\mathcal{P}_y$. As before, w must satisfy $\{i, j\}$, which implies there exists some $a_k \in w[i] \cap w[j]$. Then $i, j \in V_k$, so $\{i, j\}$ is in the subgraph induced by V_k. This proves our claim and shows $\sigma \leq \mu$.

Now suppose $\mathcal{C} = \{V_0, V_1, \ldots, V_{\sigma-1}\}$ is an IPEC of $\mathcal{P}_y$. Let $\{a_0, a_1, \ldots, a_{\sigma-1}\}$ be a collection of distinct letters and construct an indeterminate string w by setting $w[i] = \{a_j \mid i \in V_j\}$. Since each i is in some V_j, $w[i]$ is non-empty for all i. We claim w satisfies y. Suppose i and j are connected by a negative edge in $\mathcal{P}_y$. Then there is no $V_k \in \mathcal{C}$ which contains both i and j, so by construction, $w[i] \cap w[j] = \emptyset$. Hence all negative edges of $\mathcal{P}_y$ are satisfied. Now suppose i and j are connected by a positive edge. This edge is in $\mathcal{P}_y[V_k]$ for some $V_k \in \mathcal{C}$, which implies $a_k \in w[i] \cap w[j]$, satisfying the positive edge. Thus all edges of $\mathcal{P}_y$ are satisfied, which proves our claim. Note w uses σ letters, so $\mu \leq \sigma$. Therefore, $\mu = \sigma$. ◀

We can use this construction to bound $\mu(y)$, but first we introduce a few concepts. Given a graph G, an *edge clique cover* of G is a set of cliques in G such that each edge of G is in at least one of these cliques. The *edge clique cover number* of G is the size of the smallest edge clique cover of G, which we denote by $\theta(G)$. We denote the complement of G by $\overline{G}$, i.e., the graph defined by $V(\overline{G}) = V(G)$ and two vertices of $\overline{G}$ are joined by an edge if and only if they are not joined by an edge in G.

▶ **Lemma 3.3.** *Let y be a feasible prefix array of size n such that $y \neq n00\cdots0$ and $y \neq n(n-2)(n-3)\cdots0$. Then $\mu(y) \leq \theta\left(\overline{\mathcal{P}_y^-}\right)$.*

Edge clique covers have been well studied, and we can use results on them to bound $\mu(y)$. Specifically, we use the following results.

▶ **Theorem 3.4** ([8, Theorem 2])**.** *Let G be a graph on n vertices. Then $\theta(G) \leq \lfloor \frac{n^2}{4} \rfloor$.*

▶ **Theorem 3.5** ([12, Theorem 5])**.** *Let G be a graph on n vertices with $m \geq \lfloor \frac{n^2}{4} \rfloor$ edges. Further, set $k = \binom{n}{2} - m$ (i.e., k is the number of edges in $\overline{G}$) and let t be the greatest integer such that $t^2 - t \leq k$. Then $\theta(G) \leq k + t$.*

▶ **Theorem 3.6** ([2, Theorem 1.4])**.** *Let G be a graph on n vertices with maximum degree d. Then $\theta(\overline{G}) \leq 2e^2(d+1)^2 \ln n$, where e is the base of the natural logarithm.*

Now we can state a bound on $\mu(y)$.

▶ **Theorem 3.7.** *Let y be a feasible prefix array of size n, and let r be the number of negative edges in $\mathcal{P}_y$. Then $\mu(y) \leq \min\{r + \sqrt{r} + 1, 2e^2(d+1)^2 \ln n\}$, where e is the base of the natural logarithm and d is the maximum degree of a vertex in $\mathcal{P}_y^-$.*

Proof. We use Lemma 3.3, so we first check the cases $y = n00\cdots0$ and $y = n(n-2)(n-3)\cdots0$. Suppose $y = n00\cdots0$. Notice that y is satisfied by $abb\cdots b$. Similarly, if $y = n(n-2)(n-3)\cdots0$, then y is satisfied by $aa\cdots ab$. In both of these cases, any string satisfying y must have at least two letters. Moreover, in both of these cases $r = n - 1$, hence $\mu(y) = 2 \leq \min\{r + \sqrt{r} + 1, 2e^2(r+1)^2 \ln n\}$ and the result holds.

Thus, by Lemma 3.3, we may assume that $\mu(y) \leq \theta\left(\overline{\mathcal{P}_y^-}\right)$. Let $\theta = \theta\left(\overline{\mathcal{P}_y^-}\right)$. It follows from Theorem 3.6 that $\theta \leq 2e^2(d+1)^2 \ln n$. To get the $r + \sqrt{r} + 1$ bound, we apply

Theorem 3.5, but this requires that $\overline{\mathcal{P}_y^-}$ has at least $\lfloor \frac{n^2}{4} \rfloor$ edges. Note that $\overline{\mathcal{P}_y^-}$ has $\binom{n}{2} - r$ edges. Since $r < n$, the number of edges in $\overline{\mathcal{P}_y^-}$ is at least $\binom{n}{2} - (n-1) = \frac{n^2-3n+2}{2}$.

Define the function $f(n) = \frac{n^2-3n+2}{2} - \frac{n^2}{4} = \frac{n^2-6n+4}{4}$. Whenever f is positive, $\overline{\mathcal{P}_y^-}$ has at least $\frac{n^2}{4}$ edges. Note that $f(6) = 1 > 0$, and $f'(n) = \frac{n-3}{2}$ is positive for any $n > 3$. Hence f is positive for all $n \geq 6$. Note that the complement of $\overline{\mathcal{P}_y^-}$ is $\mathcal{P}_y^-$, which has r edges. Hence, by Theorem 3.5, if $n \geq 6$ and t is the greatest integer such that $t^2 - t \leq r$, then $\theta \leq r + t$. Notice that $t < \sqrt{r} + 1$, so $\theta < r + \sqrt{r} + 1$. This just leaves the cases where $n < 6$. Note that in these cases, $r + \sqrt{r} + 1 < 2e^2(d+1)^2 \ln n$, because $r - 1 < n$. Moreover, we can easily check that for each combination of n and r, either $\overline{\mathcal{P}_y^-}$ has at least $\frac{n^2}{4}$ edges, or $\frac{n^2}{4} < r + \sqrt{r} + 1$. In the former case, we can apply Theorem 3.5 to give $\theta < r + \sqrt{r} + 1$. In the latter case, Theorem 3.4 gives us $\theta \leq \frac{n^2}{4}$, implying $\theta < r + \sqrt{r} + 1$. ◀

Since $r \leq n - 1$, we get the following corollary.

▶ **Corollary 3.8.** *Let y be a feasible prefix array of size n. Then $\mu(y) \leq n + \sqrt{n}$.*

4 Connecting Prefix Arrays and Border Arrays

An indeterminate string, w, of length n has a *border* of length $\ell \in 0..n-1$ if $w[0..\ell) \approx w[n-\ell..n)$. The *border array*, $\beta[0..n)$, of an indeterminate string w is an integer array such that $\beta[0] = 0$ and for $i > 0$, $\beta[i]$ is the length of the longest border of $w[0..i]$. For example, $a\{a,b\}\{a,b\}bac$ has border array $\beta = 012330$. A border array β is *feasible* if there exists an indeterminate string such that β is its border array. A prefix array y *satisfies* a border array β if all strings with prefix array y also have border array β.

▶ **Theorem 4.1.** *Let β be a border array of size n. Then a prefix array y satisfies β if and only if the following two conditions hold: (1) $\beta[j] \leq y[j - \beta[j] + 1]$ for all $j \in 0..n-1$, and (2) $y[i] \leq j - i$ for all $i \leq j - \beta[j]$.*

Proof. Let y be a prefix array that satisfies β and w be any string with prefix array y. Since $w[0..j]$ has a maximal border of length $\beta[j]$, we have $w[0..\beta[j]) \approx w[j - \beta[j] + 1..j]$. Since $y[j - \beta[j] + 1]$ gives the length of the longest prefix of w that matches a prefix of $w[j - \beta[j] + 1..|w|)$, we have $y[j - \beta[j] + 1] \geq \beta[j]$ which gives (1). Now let $i \leq j - \beta[j]$ and $y[i] = j - i + r$ for some r. Then $w[0..j-i+r) \approx w[i..j+r)$. If $r > 0$ then $w[0..j-i] \approx w[i..j]$, so $w[0..j]$ has a border of length at least $j - i + 1$. However, since $\beta[j] \leq j - i$, $w[0..j]$ has a maximal border of length $j - i$, so $r \leq 0$. Therefore $y[i] \leq j - i$, so (2) must hold.

For the reverse implication, let y be a prefix array satisfying (1) and (2), and w be a string with prefix array y. We show that w must have border array β. Let $j \in 0..n-1$ be arbitrary. By (1) the factor of w of length $\beta[j]$ starting at position $j - \beta[j] + 1$ matches $w[0..\beta[j])$, so $w[0..j]$ has a border of length $\beta[j]$. To see that this border is maximal, suppose there exists a border of $w[0..j]$ with length $\beta[j] + r$ for some $r \geq 1$. Then $y[j - \beta[j] - r + 1] \geq \beta[j] + r$ which contradicts (2). Thus, y satisfies β. ◀

This characterization of prefix arrays which satisfy a given border array leads to a natural question: Given a border array, what degree of freedom do we have in creating a prefix array which satisfies it? The following corollary answers this question, but requires one property of border arrays referred to as the *slowly-increasing property*: for any border array $\beta = \beta[0..n)$ feasible by some indeterminate string w, $\beta[j+1] \leq \beta[j] + 1$ for all $j \in 0..n-2$.

▶ **Corollary 4.2.** *Let y be a prefix array that satisfies border array β. Then β completely determines $y[i]$, where $i > 0$ if and only if $\beta[i] = 0$ or there exists some j such that $i = j - \beta[j] + 1$. Moreover, if $i = j - \beta[j] + 1$ for some j, then $y[i] = \beta[j]$ for the largest j with this property.*

The slowly-increasing property characterizes border arrays of indeterminate strings.

▶ **Theorem 4.3.** *Every slowly-increasing array is the border array for some indeterminate string.*

Proof. Since every feasible prefix array is satisfied by some indeterminate string, it suffices to show that the set of prefix arrays which satisfy any slowly-increasing array is non-empty and feasible. We use the conditions given in Theorem 4.1. Recall that if a prefix array y is feasible, $y[i] \leq n - i$ for all $i \in 0..n-1$. Let β be a slowly-increasing array. For $j \in 0..n-1$ we have that $\beta[j] \leq n - (j - \beta[j] + 1)$, so satisfying (1) never forces y to be infeasible.

Now we check that (1) and (2) never force an empty set of possible assignments to a position of a prefix array. Suppose to the contrary that there exist such positions j_1 and j_2. Condition (1) gives $\beta[j_1] \leq y[j_1 - \beta[j_1] + 1]$ and Condition (2) gives $y[j_1 - \beta[j_1] + 1] \leq j_2 - j_1 + \beta[j_1] - 1$ for $j_1 - \beta[j_1] + 1 \leq j_2 - \beta[j_2]$. Since we assume no y can satisfy both of these, $j_2 - (j_1 - \beta[j_1] + 1) < \beta[j_1]$, or equivalently $j_2 \leq j_1$. This means that for $i = j_1 - \beta[j_1] + 1$, there is no possible assignment for $y[i]$ and thus no prefix array satisfying β. Condition (2) requires that $i \leq j_2 - \beta[j_2]$, so in this case we have $j_1 - \beta[j_1] + 1 \leq j_2 - \beta[j_2]$. However, rearranging this gives $\beta[j_2] + j_1 - j_2 + 1 \leq \beta[j_1]$. Since $j_2 \leq j_1$, this violates the slowly-increasing property of β, a contradiction. Thus, no such j_1 and j_2 can exist and we conclude that there exists a non-empty set of assignments to $y[i]$ for each i, so β is feasible. ◀

The following theorem counts the total number of slowly-increasing arrays of a given size.

▶ **Theorem 4.4.** *For all $n \geq 1$, the number of slowly-increasing arrays of size n is $C_n = \frac{1}{n+1}\binom{2n}{n}$, the nth Catalan number.*

Proof. This follows easily using basic enumerative combinatorics (see, e.g., [18, 17]). ◀

This gives us the following corollary.

▶ **Corollary 4.5.** *The number of distinct border arrays of size n for indeterminate strings is exactly the n^{th} Catalan number, $C_n = \frac{1}{n+1}\binom{2n}{n}$.*

5 Restricting Prefix Arrays to Partial Words

Since a hole matches any other letter, the following lemma follows directly from Lemma 2.1.

▶ **Lemma 5.1.** *Let y be the prefix array for some partial word w. Then $w[i]$ can be a hole if and only if there does not exist $j \in 0..n-1$ such that either (1) $y[j] = i$ and $i + j < n$ or (2) $j + y[j] = i$ if and only if i is not incident to any negative edges in $\mathcal{P}_y$.*

The next theorem gives a characterization of prefix arrays which can be satisfied by a partial word. It is similar to a characterization of regular prefix arrays given by [4, Lemma 10].

▶ **Theorem 5.2.** *Let y be a feasible prefix array and let V^- be the set of vertices in $\mathcal{P}_y$ which are incident to a negative edge. The following are equivalent: (1) the array y is the prefix array for some partial word, (2) every cycle in $\mathcal{P}_y$ which contains exactly one negative edge has at least one vertex which is not in V^-, and (3) every negative edge of $\mathcal{P}_y$ connects vertices in two different connected components of $\mathcal{P}_y^+[V^-]$.*

Proof. First we prove (1) implies (2) by contraposition. Assume $\{i, j\}$ is a negative edge in $\mathcal{P}_y$ which connects vertices i and j, where i and j lie in the same connected component of $\mathcal{P}_y^+[V^-]$. Then there exists a path, p, in $\mathcal{P}_y^+[V^-]$ from i to j. Further suppose w is an indeterminate string which satisfies y. Since each edge in path p is a positive edge, Lemma 2.2 implies there exists some k such that k is in this path and $w[k]$ is an indeterminate letter. However, $k \in V^-$, so by Lemma 5.1, $w[k]$ cannot be a hole. Since holes are the only indeterminate letters allowed in a partial word, w is not a partial word.

Next we prove (2) implies (3), again by contraposition. Suppose $\{i, j\}$ is a negative edge such that i and j are in the same connected component of $\mathcal{P}_y^+[V^-]$. Then there exists a path from j to i which lies in $\mathcal{P}_y^+[V^-]$. Note that all the edges in this path are positive, and all the vertices are in V^-. Then concatenating $\{i, j\}$ to this path creates a cycle in $\mathcal{P}_y$ which contains exactly one negative edge, but whose vertices all are in V^-.

Finally, we prove (3) implies (1). Assume every negative edge of $\mathcal{P}_y$ connects vertices in two different connected components of $\mathcal{P}_y^+[V^-]$. Let $C_0, C_1, \ldots, C_{\ell-1}$ be the connected components of $\mathcal{P}_y^+[V^-]$, and construct a partial word w on the alphabet $\{a_0, a_1, \ldots, a_{\ell-1}\}$ by setting $w[i] = a_j$ if $i \in C_j$ and $w[i] = \diamond$ if $i \notin V^-$. Note that this construction does indeed assign one letter (or hole) to each position of w, and we claim that w satisfies y.

Suppose $\{i, j\}$ is a negative edge in $\mathcal{P}_y$. By assumption, i and j are in different connected components of $\mathcal{P}_y^+[V^-]$, so $w[i] \not\approx w[j]$. Hence this edge is satisfied. Now suppose $\{i, j\}$ is a positive edge in $\mathcal{P}_y$. If $i \notin V^-$, then $w[i] = \diamond$, which implies $w[i] \approx w[j]$ and w satisfies $\{i, j\}$. A symmetric argument holds for $j \notin V^-$. Now assume $i, j \in V^-$. This implies i and j are in the same connected component of $\mathcal{P}_y^+[V^-]$, so $w[i] = w[j]$, satisfying $\{i, j\}$. Therefore w satisfies all edges of $\mathcal{P}_y$, proving that y is the prefix array for the partial word w. ◀

Based upon the construction given in the above proof, we define $V^-(\mathcal{P}_y)$ (or just V^- if $\mathcal{P}_y$ is clear from context) to be the set of vertices in $\mathcal{P}_y$ which are incident to a negative edge. Further, construct the graph $\mathcal{C}_y$ as follows: make one vertex in $\mathcal{C}_y$ for each connected component in $\mathcal{P}_y^+[V^-]$ and join two vertices in $\mathcal{C}_y$ if and only if there exists a negative edge in $\mathcal{P}_y$ between their corresponding components. Finally, if y is the prefix array for some partial word, $\mu_\diamond(y)$ will denote the minimum alphabet size for a partial word satisfying y.

▶ **Theorem 5.3.** *Let y be the prefix array for some partial word. Then $\mu_\diamond(y) = \chi(\mathcal{C}_y)$.*

Proof. Let $C_1, C_2, \ldots, C_\ell$ be the connected components of $\mathcal{P}_y^+[V^-]$ and assume we have a valid coloring of $\mathcal{C}_y$. We treat these colors as letters and construct w using a similar method to the one given in the proof of Theorem 5.2. We let $w[i]$ be the color of C_k if $i \in C_k$ and let $w[i] = \diamond$ otherwise. The proof that w satisfies y follows exactly the last part of the proof of Theorem 5.2. Hence $\mu_\diamond(y) \leq \chi(\mathcal{C}_y)$.

Let w be a partial word satisfying y on an alphabet, A, of minimum size. Suppose i and i' are in the same connected component of $\mathcal{P}_y^+[V^-]$. Then there exists a path, $i = j_1, j_2, \ldots, j_k = i'$, of positive edges from i to i' such that each vertex in this path is in V^-. By Lemma 5.1, $w[j_\ell]$ must be a regular letter for each $\ell \in 1..k$. Then, since j_ℓ and $j_{\ell+1}$ are joined by a positive edge for each $\ell \in 1..k-1$, it follows that $w[j_1] = w[j_2] = \cdots = w[j_k]$. Hence $w[i] = w[i']$. This implies that all positions of w associated with vertices in a given connected component of $\mathcal{P}_y^+[V^-]$ must have the same letter.

Now we give a coloring of $\mathcal{C}_y$ using the letters in A. Color a vertex, v, of $\mathcal{C}_y$ with $a \in A$ if there exists some i in the connected component of $\mathcal{P}_y^+[V^-]$ represented by v such that $w[i] = a$. It follows from the above discussion that this coloring operation is well-defined. We claim that this is a valid coloring. Suppose u and v are vertices of $\mathcal{C}_y$ joined by an edge. This edge corresponds to some negative edge $\{i, j\}$ in $\mathcal{P}_y$, where i is in the connected component

of $\mathcal{P}_y^+[V^-]$ represented by u and j is in the connected component of $\mathcal{P}_y^+[V^-]$ represented by v. Since i and j are connected by a negative edge, it must be that $w[i] \not\approx w[j]$ which implies u and v have different colors. Hence this is a valid coloring and $\chi(\mathcal{C}_y) \leq \mu_\diamond(y)$. ◀

It is well known that if a graph G has e edges, then $\frac{\chi(G)(\chi(G)-1)}{2} \leq e$. We can use this fact to bound $\mu_\diamond(y)$.

▶ **Corollary 5.4.** *Let y be the prefix array for some partial word such that $|y| = n$. Then $\mu_\diamond(y) \leq \left\lceil \sqrt{2n} \right\rceil$.*

We can show that this bound is tight within a constant multiple using Sidon sets. In number theory, a set $S = \{s_0, s_1, \ldots, s_{m-1}\}$ of natural numbers is a finite Sidon set, named after the Hungarian mathematician Simon Sidon, if the pairwise sums $s_i + s_j$, $i \leq j$, are all different. It is easy to show that Sidon sets have the property that the pairwise differences $|s_i - s_j|$, $i < j$, are also all different.

▶ **Proposition** 5.5. There exists a prefix array, y, of size n such that $\mu_\diamond(y) = (1 - o(1))\sqrt{n}$.

Proof. Erdős and Turán [7, 9] showed that there exists a Sidon set with $(1 - o(1))\sqrt{n}$ elements such that each element is less than n. Let $S = \{s_0, s_1, \ldots, s_{m-1}\}$ be such a set, where $m = (1 - o(1))\sqrt{n}$. Define $y = y[0..n)$ by

$$y[i] = \begin{cases} s_j & \text{if } i = s_k - s_j, \text{ for some } s_j, s_k \in S, s_j < s_k, \\ n - i & \text{otherwise.} \end{cases}$$

We show that $\mathcal{P}_y^-$ contains an m-clique. Consider $s_r, s_t \in S$, where $s_r < s_t$. This implies $y[s_t - s_r] = s_r$, and since $s_t - s_r + s_r = s_t < n$, it follows that $\{y[s_t - s_r], s_t - s_r + y[s_t - s_r]\} = \{s_r, s_t\} \in E^-$, where $\{s_r, s_t\}$ indicates an edge between the vertices indexed by s_r and s_t. Moreover, this holds for any pair of elements in S, so the vertices indexed by S form an m-clique in $\mathcal{P}_y^-$. This implies $\mu_\diamond(y) \geq m$.

Now construct a partial word w on the alphabet $A = \{a_0, a_1, \ldots, a_{m-1}\}$ by $w[i] = a_j$ if $i = s_j \in S$, while $w[i] = \diamond$ otherwise. We claim that w satisfies y. Since $i + y[i] = n$ whenever $y[i] \notin S$, the only negative edges of $\mathcal{P}_y$ are of the form $\{s_r, s_t\}$ where $s_r, s_t \in S$. Note that $w[s_r] = a_r \not\approx a_t = w[s_t]$, so w satisfies all of these negative edges. Moreover, since S is an m-clique in $\mathcal{P}_y^-$, any positive edge must be incident to some vertex i where $i \notin S$. Then $w[i] = \diamond$, which matches anything, so this positive edge must be satisfied. Therefore w satisfies y on m letters, implying $\mu_\diamond(y) \leq m$. ◀

Referring to the proof of Proposition 5.5, the Sidon set $\{0, 1, 4, 6\}$ determines the prefix array $y = 7041010$. Note that $\mathcal{P}_y^-$ contains the 4-clique $\{0, 1, 4, 6\}$.

Finally, two partial words w and w' of length n are *p-equivalent* if for all i and j such that $0 \leq i \leq j < n$ we have $w[i] \approx w[j]$ if and only if $w'[i] \approx w'[j]$. In other words, w' is just a relabeling of w. If two partial words are not p-equivalent, we say they are *p-distinct*. We use this notion to bound $\text{pref}_\diamond(n)$, the number of prefix arrays of size n valid for partial words.

▶ **Proposition** 5.6. For sufficiently large n,

$$\text{pref}_\diamond(n) \leq \left\lceil \sqrt{2n} \right\rceil \left\{ {n+1 \atop \left\lceil \sqrt{2n} \right\rceil + 1} \right\},$$

where $\left\{ {n+1 \atop \left\lceil \sqrt{2n} \right\rceil + 1} \right\}$ denotes a Stirling number of the second kind.

Proof. By Corollary 5.4, any prefix array valid for partial words can be satisfied by a partial word with at most $\left\lceil \sqrt{2n} \right\rceil$ letters. Partial words which are p-equivalent have the same prefix array, so different prefix arrays are necessarily p-distinct. We may bound the number of prefix arrays valid for partial words by the number of p-distinct partial words on at most $\left\lceil \sqrt{2n} \right\rceil$ letters. Using ideas from [14], the number of p-distinct partial words of length n with h holes and μ letters is $\binom{n}{h}\left\{ {n-h \atop \mu} \right\}$. Summing over all possible numbers of holes we have $\sum_{h=0}^{n-\mu} \binom{n}{h}\left\{ {n-h \atop \mu} \right\} = \sum_{j=\mu}^{n} \binom{n}{j}\left\{ {j \atop \mu} \right\} = \left\{ {n+1 \atop \mu+1} \right\}$.

Consider p-distinct partial words using less than $\left\lceil \sqrt{2n} \right\rceil$ letters. The following facts about Stirling numbers are useful. First, for sufficiently large μ and n we have $\left\{ {n \atop \mu} \right\} < \frac{\mu^n}{\mu!}$. Second, for fixed n, $\left\{ {n \atop \mu} \right\}$ is a unimodal sequence with mode asympototically approaching $\frac{n}{\log n}$. Thus for sufficiently large n, we have $\left\{ {n+1 \atop \lceil \sqrt{2n} \rceil + 1} \right\} \geq \left\{ {n+1 \atop j} \right\}$ for all $j < \left\lceil \sqrt{2n} \right\rceil$. Summing over each possible number of letters and using the unimodality of Stirling numbers,

$$\mathrm{pref}_{\diamond}(n) \leq \sum_{\mu=1}^{\lceil \sqrt{2n} \rceil} \left\{ {n+1 \atop \mu+1} \right\} \leq \lceil \sqrt{2n} \rceil \left\{ {n+1 \atop \lceil \sqrt{2n} \rceil + 1} \right\}.$$

◀

6 Conclusion and Future Work

In Section 3, we demonstrated two methods for constructing an indeterminate string which satisfies a given prefix array using the smallest alphabet possible. Moreover, we showed that the minimum alphabet size for an indeterminate string satisfying a prefix array y of size n is at most $n + \sqrt{n}$. Indeed, we showed that the minimum alphabet size is at most $r + \sqrt{r} + 1$, where r is the number of negative edges in $\mathcal{P}_y$. One suggestion for future work is to improve this bound or show it is tight. Since there are many results bounding chromatic numbers, we believe the method involving $\mathcal{Q}_y$ may be useful in lowering this bound. Examining many prefix arrays has led us to the following conjecture.

▶ **Conjecture** 6.1. Let y be a feasible prefix array of size n. Then $\mu(y) < n$.

Another suggestion for future work, as mentioned in [4], is to develop an efficient algorithm to compute a string on an alphabet of minimum size for a given prefix array.

In section 5, we restricted ourselves to considering prefix arrays for partial words. We gave a characterization of prefix arrays y valid for partial words in terms of the prefix graph $\mathcal{P}_y$. Moreover, we gave a method to construct a partial word on the smallest possible alphabet for a given prefix array. We showed that the minimum alphabet size for a partial word satisfying a given prefix array of size n is at most $\left\lceil \sqrt{2n} \right\rceil$, and we showed that this bound is tight (up to a constant multiple) using Sidon sets. We also bounded $\mathrm{pref}_{\diamond}(n)$, the number of prefix arrays of size n valid for partial words, for large enough n, in terms of Stirling numbers of the second kind. Based on experimental data, it seems that our bound is not tight, and we believe that there is actually an exponential upper bound for $\mathrm{pref}_{\diamond}(n)$.

▶ **Conjecture** 6.2. For all n, $\mathrm{pref}_{\diamond}(n) \leq 4^{n-1}$.

One final suggestion for future work is to develop an algorithm which enumerates all prefix arrays of a given size valid for partial words. In the case of regular strings, all prefix arrays of size n can be enumerated in constant time with respect to the output size [14].

In addition, we established a World Wide Web server interface at

`http://www.uncg.edu/cmp/research/arrays`

for automated use of a program that produces an indeterminate string with the minimum number of letters for a given prefix array.

Acknowledgements. We thank Professor W. F. Smyth for sending us his paper on indeterminate strings, prefix arrays, and undirected graphs. We also thank Brian Bowers and Nathan Fox for the slowly-increasing property, the number of slowly-increasing arrays of size n equalling the nth Catalan number, and the fact that the number of p-distinct partial words of length n with h holes and μ letters is $\binom{n}{h}\left\{ {n-h \atop \mu} \right\}$, where $\left\{ {n \atop \mu} \right\}$ denotes a Stirling number of the second kind.

References

1 K. Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16:1039–1051, 1987.
2 N. Alon. Covering graphs by the minimum number of equivalence relations. *Combinatorica*, 6:201–206, 1986.
3 F. Blanchet-Sadri. *Algorithmic Combinatorics on Partial Words*. Chapman & Hall/CRC Press, Boca Raton, FL, 2008.
4 M. Christodoulakis, P. J. Ryan, W. F. Smyth, and S. Wang. Indeterminate Strings, Prefix Arrays & Undirected Graphs. preprint, 2013.
5 J. Clément, M. Crochemore, and G. Rindone. Reverse engineering prefix tables. In S. Albers and J.-Y. Marion, editors, *STACS 2009, 26th International Symposium on Theoretical Aspects of Computer Science, Freiburg, Germany*, volume 3 of *LIPIcs*, pages 289–300. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2009.
6 M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.
7 P. Erdős. On a problem of Sidon in additive number theory and on some related problems. Addendum. *Journal of the London Mathematical Society, Second Series*, 19:208, 1944.
8 P. Erdős, A. W. Goodman, and L. Pósa. The representation of a graph by set intersections. *Canadian Journal of Mathematics*, 18:106–112, 1966.
9 P. Erdős and P. Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society, Second Series*, 16:212–215, 1941.
10 M. Fischer and M. Paterson. String matching and other products. In R. Karp, editor, *7th SIAM-AMS Complexity of Computation*, pages 113–125, 1974.
11 J. L. Gross and J. Yellen. *Handbook of Graph Theory*. CRC Press, 2004.
12 L. Lovász. On covering of graphs. In *Theory of Graphs (Proceedings of the Colloquium, Tihany, 1966)*, pages 231–236. Academic Press, New York, 1968.
13 M. G. Main and R. J. Lorentz. An O(nlog n) algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5:422–432, 1984.
14 D. Moore, W. F. Smyth, and D. Miller. Counting distinct strings. *Algorithmica*, 23:1–13, 1999.
15 W. F. Smyth. *Computing Patterns in Strings*. Pearson Addison-Wesley, 2003.
16 W. F. Smyth and S. Wang. New perspectives on the prefix array. In *15th Symposium on String Processing and Information Retrieval*, volume 5280 of *Lecture Notes in Computer Science*, pages 133–143. Springer-Verlag, Berlin, Heidelberg, 2008.
17 R. P. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge University Press, 2001.
18 R. P. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge Studies in Advanced Mathematics, 2011.

Online Bin Packing with Advice*

Joan Boyar[1], Shahin Kamali[2], Kim S. Larsen[1], and Alejandro López-Ortiz[2]

1 **University of Southern Denmark, Denmark**
{joan,kslarsen}@imada.sdu.dk
2 **University of Waterloo, Canada**
{s3kamali,alopez-o}@uwaterloo.ca

Abstract

We consider the online bin packing problem under the advice complexity model where the "online constraint" is relaxed and an algorithm receives partial information about the future requests. We provide tight upper and lower bounds for the amount of advice an algorithm needs to achieve an optimal packing. We also introduce an algorithm that, when provided with $\log n + o(\log n)$ bits of advice, achieves a competitive ratio of $3/2$ for the general problem. This algorithm is simple and is expected to find real-world applications. We introduce another algorithm that receives $2n + o(n)$ bits of advice and achieves a competitive ratio of $4/3 + \varepsilon$. Finally, we provide a lower bound argument that implies that advice of linear size is required for an algorithm to achieve a competitive ratio better than $9/8$.

1998 ACM Subject Classification F.1.2 Modes of Computation (online computation)

Keywords and phrases online algorithms, advice complexity, bin packing

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.174

1 Introduction

In the classical one-dimensional bin packing problem the goal is to pack a given sequence of *items* into a minimum number of *bins* with fixed and equal capacities. For convenience, it is assumed that items sizes are in the range $(0, 1]$ and the capacities of bins are 1. In the *online* version of the problem, the items are revealed one by one, and an algorithm must pack each item without any knowledge about future items. The decisions of an online algorithm are irrevocable, i.e., it is not possible to move an item from one bin to another after it is *packed* in a bin.

The online bin packing problem has many applications in practice, from loading trucks subject to weight limitations to creating file backups in removable media [10]. Heuristics that have been proposed for the problem include Next-Fit (NF), First-Fit (FF), Best-Fit (BF), and the Harmonic-based class of algorithms. NF maintains a single *open* bin and places an item in that bin; in the case the item does not fit, it *closes* the bin and opens a new one. FF keeps a list of bins in the order they are opened, packs an item in the first bin that has enough space, and opens a new bin if necessary. BF performs similarly to FF, except that the bins are ordered in increasing order of their remaining capacity. Harmonic-based algorithms are based on the idea of packing items of similar sizes together in a bin. For Harmonic$_K$, an

* The work of the first and third author was partially supported by the Danish Council for Independent Research, Natural Sciences and the Villum Foundation, and most of the work was carried out while these authors were visiting the University of Waterloo.



© Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 174–186



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



item has type i ($1 \leq i \leq K-1$) if it is in the range $(\frac{1}{i+1}, \frac{1}{i}]$, and type K if it is in the range $(0, \frac{1}{K}]$. The algorithm applies the NF strategy for items of each type separately.

As for other online problems, the standard method for comparing bin packing algorithms is competitive analysis. Under competitive analysis, the performance of an algorithm $\mathbb{A}$ is compared to that of OPT, which is the optimal offline algorithm. More precisely, the competitive ratio of an algorithm $\mathbb{A}$ is the asymptotically maximum ratio of the cost of $\mathbb{A}$ to that of OPT for serving the same sequence σ. FF and BF have the same competitive ratio of 1.7, while the best Harmonic-based algorithm has a competitive ratio of at most 1.58889 [22]. It is also known that no online algorithm can have a competitive ratio better than 1.54037 [3].

The total lack of information about the future is unrealistic in many real-world scenarios [13]. A natural approach for addressing this issue is to relax the problem by providing extra information about the input sequence. For the online bin packing problem, such relaxations have been studied in the contexts of *lookahead*, in which the online algorithm can look at the items arriving in the near future [16], and *closed bin packing*, in which the length of the request sequence is known to the online algorithm [1]. In both cases, the average performance of the online algorithm improves, compared to the online algorithms with no information about the future.

The advice complexity model for online algorithms is a more general framework under which the "no knowledge assumption" behind online algorithms is relaxed, and the algorithm receives some bits of *advice* about the future requests. The advice can be any information about the input sequence and is generated by an offline oracle which has unbounded computational power. Provided with the appropriate advice, the online algorithms are expected to achieve improved competitive ratios. The advice model has received significant attention since its introduction [8, 17, 13, 7, 18, 20, 9, 4, 11, 15, 19, 6, 5, 21].

In this paper, we study the advice complexity of the online bin packing problem. Our interest in studying the problem under this setting is mostly theoretical. Nevertheless, in many practical scenarios, it can be justified to allow a fast offline oracle to take a "quick look" at the input sequence and send some advice to the online algorithm. For example, it may be possible to take a quick look and count the number of items which are larger than 1/2 and smaller than 2/3 of the bin capacity. We show that this form of advice can be used to achieve an algorithm which outperforms all online algorithms.

1.1 Model

In the last few years, slightly different models of advice complexity have been proposed for online problems. All these models assume that there is an offline oracle with infinite computational power, which provides the online algorithm with some bits of advice. How these bits of advice are given to the algorithm is the source of difference between the models. In the first model, presented in [12], an online algorithm poses a series of questions which are answered by the offline oracle in *blocks of answers*. The total size of the answers, measured in the number of bits, defines the advice complexity. The problem with this model is that a lot of information can be encoded in the individual length of each block. To address this issue, another model is proposed in [13] which assumes that online algorithms receive a fixed number of bits of advice per request. We call this model the *advice-with-request model*. This model is studied for problems, such as metrical task systems and k-server, and the results tend to use at least a constant number of bits of advice per request [13, 20]. Nevertheless, there are many online problems for which a sublinear and even a constant number of bits of advice in total is sufficient to achieve good competitive ratios. However, under the advice-with-request

model, the possibility of sending a sublinear number of advice bits to the algorithm is not well defined. In [8, 7] another model of advice complexity is presented which assumes that the online algorithm has access to an *advice tape*, written by the offline oracle. At any time step, the algorithm may refer to the tape and read any number of advice bits. The advice complexity is the number of bits on the tape accessed by the algorithm. We refer to this model as *advice-on-tape model*. Since its introduction, the advice-on-tape model has been used to analyze the advice complexity of many online problems including paging [8, 17, 18], disjoint path allocation [8], job shop scheduling [8, 18], k-server [7, 20], knapsack [9], various coloring problems [4, 15, 5, 21], set cover [19, 6], maximum clique [6], and graph exploration [11].

Under the advice-on-tape model, we require a mechanism to infer how many bits of advice the algorithm should read at each time step. This could be implicitly derived during the execution of the algorithm or explicitly encoded in the advice string itself. For example, we may use a *self-delimited* encoding as used in [7], in which the value of a non-negative integer X is encoded by writing the value of $\lceil \log(\lceil \log(X+1) \rceil + 1) \rceil$ in unary (a string of 1's followed by a zero), the value of $\lceil \log(X+1) \rceil$ in binary [1], and the value of X in binary. These codes respectively require $\lceil \log(\lceil \log(X+1) \rceil + 1) \rceil + 1$, $\lceil \log(\lceil \log(X+1) \rceil + 1) \rceil$, and $\lceil \log(X+1) \rceil$ bits. Thus, the self-delimited encoding of X requires

$$e(X) = \lceil \log(X+1) \rceil + 2\lceil \log(\lceil \log(X+1) \rceil + 1) \rceil + 1$$

bits. The existence of self-delimited encodings at the beginning of the tape usually adds a lower-order term to the number of advice bits required by an algorithm.

Regarding notation, we use $\mathbb{A}(\sigma)$ to denote the costs of $\mathbb{A}$ for packing a request sequence σ. When σ follows from the context, we simply use $\mathbb{A}$ to denote this cost. We use similar notation for all algorithms, including Opt.

We consider the bin packing problem under the advice-on-tape model, which is formally defined as follows, based on the definition of the advice model in [7]:

▶ **Definition 1.** In the *online bin packing problem with advice*, the input is a sequence of items $\sigma = \langle x_1, \ldots, x_n \rangle$, revealed to the algorithm in an online manner ($0 < x_i \leq 1$). The goal is to pack these items in the minimum number of bins of unit size. At time step t, an online algorithm should pack item x_t into a bin. The decision of the algorithm to select the target bin is a function of $\Phi, x_1, \ldots, x_{t-1}$, where Φ is the content of the advice tape. An algorithm $\mathbb{A}$ is *c-competitive with advice complexity $s(n)$* if there exists a constant c_0 such that, for all n and for all input sequences σ of length at most n, there exists some advice Φ such that $\mathbb{A}(\sigma) \leq c\ \textsc{Opt}(\sigma) + c_0$, and at most the first $s(n)$ bits of Φ have been accessed by the algorithm. If $c = 1$ and $c_0 = 0$, then $\mathbb{A}$ is *optimal*.

1.2 Contribution

We answer different questions about the advice complexity of the online bin packing problem. First, we study how many bits of advice are required to achieve an optimal solution. We consider two different settings of the problem. When there is no restriction on the number of distinct items or their sizes, we present the easy result that $n\lceil \log \textsc{Opt}(\sigma) \rceil$ bits of advice are sufficient to achieve an optimal solution, where $\textsc{Opt}(\sigma)$ is the number of bins in an optimal packing. We also prove that at least $(n - 2\,\textsc{Opt}(\sigma)) \log \textsc{Opt}(\sigma)$ bits of advice are required to achieve an optimal solution.

[1] In this paper we use $\log n$ to denote $\log_2(n)$.

When there are m distinct items in the sequence, we prove that at least $(m-3)\log n - 2m\log m$ bits of advice are required to achieve an optimal solution. If m is a constant, there is a linear time online algorithm that receives $m\log n + o(\log n)$ bits of advice and achieves an optimal solution. We also show that, even if m is not a constant, there is a polynomial time online algorithm that receives $m\lceil\log(n+1)\rceil + o(\log n)$ bits of advice and achieves a packing with $(1+\varepsilon)\,\textsc{Opt}(\sigma)+1$ bins.

We also study a relevant question that asks how many bits of advice are required to perform strictly better than all online algorithms. We bound this by providing an algorithm which receives $\log n + o(\log n)$ bits of advice and achieves a competitive ratio of $3/2$. Recall that any online bin packing algorithm has a competitive ratio of at least 1.54037 [3]. Hence, our algorithm outperforms all online algorithms.

Moreover, we introduce an algorithm that receives $2n + o(n)$ bits of advice and achieves a competitive ratio of $4/3+\varepsilon$, for any fixed value of $\varepsilon > 0$. We also prove a lower bound that implies that a linear number of bits of advice are required to achieve a competitive ratio of $9/8-\delta$ for any fixed value of $\delta > 0$.

Due to space restrictions, many proofs have been removed. They will appear in the long version of the paper.

2 Optimal Algorithms with Advice

In this section we study the amount of advice required to achieve an optimal solution. We first investigate the theoretical setting in which there is no restriction on the number of distinct items or on their sizes. We observe that there is a simple algorithm that receives $n\lceil\log \textsc{Opt}(\sigma)\rceil$ bits of advice and achieves an optimal solution. Such an algorithm basically reads $\lceil\log \textsc{Opt}(\sigma)\rceil$ bits for each item, encoding the index of the bin that includes the item in an optimal packing. We show that the upper bound given by this algorithm is tight up to lower order terms, when $n - 2\,\textsc{Opt}(\sigma) \in \Theta(n)$.

▶ **Theorem 2.** *To achieve an optimal packing for a sequence of size n and optimal cost $\textsc{Opt}(\sigma)$, it is sufficient to receive $n\lceil\log \textsc{Opt}(\sigma)\rceil$ bits of advice. Moreover, any deterministic online algorithm requires at least $(n - 2\,\textsc{Opt}(\sigma))\log \textsc{Opt}(\sigma)$ bits of advice to achieve an optimal packing.*

Next, we consider a more realistic scenario where there are $m \in o(n)$ distinct items and the values of these items are known to the algorithm. Assume that the advice tape specifies the number of items of each size. If we are not concerned about the running time of the online algorithm, there is enough information to obtain an optimal solution. If we are concerned, we can use known results for solving the offline problem [2, 14, 23] to obtain the following:

▶ **Theorem 3.** *Consider the online bin packing problem in which there are m distinct items. If m is a constant, there is a (linear time) optimal online algorithm that receives $m\log n + o(\log n)$ bits of advice. If m is not a constant, there is a (polynomial time) online algorithm that reads $m\lceil\log(n+1)\rceil + o(\log n)$ bits of advice and achieves an almost optimal packing with at most $(1+\varepsilon)\,\textsc{Opt}(\sigma)+1$ bins, for any small but constant value of ε.*

We show that the above upper bound is asymptotically tight.

▶ **Theorem 4.** *At least $(m-3)\log n - 2m\log m$ bits of advice are required to achieve an optimal solution for the online bin packing problem on sequences of length n with m distinct items, each of size at least $\frac{1}{2m}$.*

3 An Algorithm with Sublinear Advice

In what follows we introduce an algorithm that receives $\log n + o(\log n)$ bits of advice and achieves a competitive ratio of $\frac{3}{2}$, for any instance of the online bin packing problem. An offline oracle can compute and write the advice on the tape in linear time, and the online algorithm runs as fast as First-Fit. Thus, the algorithm might be applied in practical scenarios in which it is allowed to have a "quick look" at the input sequence.

We call items *tiny*, *small*, *medium*, and *large* if their sizes lie in the intervals $(0, 1/3]$, $(1/3, 1/2]$, $(1/2, 2/3]$, and $(2/3, 1]$, respectively. The advice that the algorithm receives is the number of medium items, which we denote by α.

The algorithm reads the advice tape, obtains α, opens α bins, called *critical bins*, and reserves $2/3$ of the space in each of them. This reserved space will be used to pack a medium item in each of the critical bins, and these bins have a *virtual level* of size $2/3$ at the beginning. All other bins have virtual level zero when they are opened. The algorithm serves an item x in the following manner:

- If x is a large item, open a new bin for it. Set the virtual level to its size.
- If x is a medium item, put it in the reserved space of a critical bin B. Update the virtual level to the actual level. (B will not have any reserved space now.)
- If x is small or tiny, use the First Fit (FF) strategy to put it into any of the open bins, based on virtual levels (open a new bin if required). Add the size of the item to the virtual level.

Note that the critical bins appear first in the ordering maintained by the algorithm as they are opened before other bins.

▶ **Theorem 5.** *There is an online algorithm which receives* $\log n + o(\log n)$ *bits of advice and has cost* $3/2\,\textsc{Opt}(\sigma) + 3$ *for serving any sequence* σ *of size* n.

Proof. We prove that the algorithm described above has the desired property. The value of α is encoded in $X = \lceil \log(n+1) \rceil$ bits of advice. In order to read this properly from the tape, the algorithm needs to know the value of X. This can be done by adding the self-delimited encoding of X in $e(X) = \lceil \log X \rceil + 2\lceil \log \log(X) \rceil + 2$ bits at the beginning of the tape. Consequently the number of advice bits used by the algorithm is $X + O(\log X)$, which is $\log n + o(\log n)$ as stated by the theorem.

Consider the final packing of the algorithm for serving a sequence σ. There are two cases. In the first case, there is a critical bin B so that no other item, except a medium item, is packed in it. Since all tiny items are smaller than $1/3$ and can fit in B, all the non-critical bins that are opened after B include small and large items only. More precisely, they include either a single large item or two small items (except the last one which might have a single small item). Let L, M, and S denote the number of large, medium, and small items. The cost of the algorithm is at most $L + M + S/2 + 1$. Now, if $S \leq M$, this would be at most $L + 3/2M + 1$. Since $L + M$ is a lower bound on the cost of OPT, the cost of the algorithm is at most $3/2\,\textsc{Opt}(\sigma) + 1$ and we are done. If $S > M$, OPT should open $L + M$ bins for large and medium items, and in the best case, it packs M small items together with medium ones. For the other $S - M$ bins, OPT has to open at least $(S - M)/2$ bins. Hence the cost of OPT is at least $L + M + (S - M)/2 = L + M/2 + S/2$, and we have $3/2\,\textsc{Opt}(\sigma) \geq 3L/2 + 3M/4 + 3S/4 > L + M + S/2$. Thus, the cost of the algorithm is at most $3/2\,\textsc{Opt}(\sigma) + 1$.

In the second case, we assume that all critical bins include another item in addition to the medium item. We claim that at the end of serving a sequence all bins, except possibly

two, have level at least 2/3. First, we verify this for non-critical bins (bins without medium items). If a non-critical bin is opened by a large item, it clearly has level higher than 2/3. All other non-critical bins only include items of size at most 1/2. Hence, these bins, except possibly the last one, include at least two items. Among the non-critical bins that include two items, consider two bins b_i and b_j $(i < j)$ that have levels smaller than 2/3. Since b_j contains at least two items, at least one of them has size smaller than 1/3. This item could fit in b_i by the FF property. We conclude that all non-critical bins, except possibly two, have level at least 2/3. Now, suppose two critical bins b_i and b_j have levels smaller than 2/3. Consider the first non-medium item x which is packed in b_j (in the second case, such an item exists). Since a medium item is packed in the bin, x should be either tiny or small. If x is small, then the level of b_j is at least $1/2 + 1/3$, which contradicts the level of b_j being smaller than 2/3. Similarly, x cannot be a tiny item of size larger than 1/6 (since $1/2 + 1/6 \geq 2/3$). Hence, x is a tiny item of size at most 1/6. This implies that at the time the online algorithm packs x, bin b_i has a virtual level of at least 5/6. The virtual level is at most 1/6 larger than the actual level (the final level). Hence, the actual level of b_i is at least $5/6 - 1/6 = 2/3$. We conclude that at most one critical bin has level smaller than 2/3. To summarize, at most three bins have level smaller than 2/3. Hence, the cost of the algorithm is at most $3/2\,\text{OPT}(\sigma) + 3$. ◀

4 An Algorithm with Linear Advice

In this section, we present an algorithm that receives $2n + o(n)$ bits of advice and achieves a competitive ratio of $4/3 + \varepsilon$ for any sequence of size n, and arbitrarily small (but constant) values of ε. Consider an algorithm that receives an *approximate size* for each sufficiently large item x encoded using k bits. The approximate size of x would be larger than its *actual size* by at most an additive term of $1/2^k$. The algorithm can optimally pack items by their approximate sizes and achieve an *approximate packing* which includes a reserved space of size $x + \varepsilon$ $(\varepsilon \leq 1/2^k)$ for each item. Precisely, for each sufficiently large item x, the approximate packing includes a reserved space of size $x + \varepsilon$ $(\varepsilon \leq 1/2^k)$ for x. This enables the algorithm to place x in the reserved space for it in the approximate packing. Smaller items are treated differently and the algorithm does not reserve any space for them. In the reminder of this section, we elaborate this idea to achieve a 4/3-competitive algorithm.

Notice that the cost of an approximate packing can be as large as $\frac{3}{2}$ times the cost of OPT. To see that, consider a sequence which is a permutation of $\langle \frac{1}{2} + \varepsilon_1, \frac{1}{2} - \varepsilon_1, \frac{1}{2} + \varepsilon_2, \frac{1}{2} - \varepsilon_2, \ldots, \frac{1}{2} + \varepsilon_{n/2}, \frac{1}{2} - \varepsilon_{n/2} \rangle$, where $\varepsilon_i < 1/2^n (1 \leq i \leq n/2)$. Since OPT packs all bins tightly, an increase in the sizes of items by a constant (small) ε results in opening a new bin for each two bins OPT uses. Hence the cost of the optimal approximate packing can be as bad as $\frac{3}{2}$ OPT. This example suggests that using approximate packings is not good for the bins in which a small number of large items are tightly packed. To address this issue we divide the bins of OPT into two groups:

▶ **Definition 6.** Consider an optimal packing of a sequence σ. Given a small parameter $\varepsilon' < 1/60$, define *good bins* to be those where the total size of the items smaller than 1/4 in the bin is at least $5\varepsilon'$. Define all other bins to be *bad bins*.

A part of the advice received for each item x indicates if x is packed by OPT in a good bin or in a bad bin. This enables us to treat items packed in these two groups separately.

▶ **Lemma 7.** *Consider sequences for which all bins in the optimal packing are good (as defined above). There is an online algorithm that receives $o(n)$ bits of advice and achieves a competitive ratio of 4/3.*

Proof. Call an item *small* if it is smaller than or equal to 1/6 and *large* otherwise. The advice bits define the approximate sizes of all large items with a precision of ε'. The amount of advice will be roughly $2^{1/\varepsilon'} \log n$ which is $o(n)$ for constant values of ε'. The online algorithm $\mathbb{A}$ can build the optimal approximate packing of large items. In such a packing, there is a reserved space of size at most $x + \varepsilon'$ for any large item of size x. The algorithm considers this packing as a partial packing and initializes the level of each bin to be the total sizes of approximated items in that bin. For packing an item x, if x is large, $\mathbb{A}$ packs it in the space reserved for it in the approximate packing. It also updates the level of the bin to reflect the actual size of x. If x is small, $\mathbb{A}$ simply applies the First-Fit strategy to pack x in a bin of the partial packing (and opens a new bin for it if necessary). We prove that $\mathbb{A}$ is 4/3-competitive. In the final packing by $\mathbb{A}$, call a bin "red" if all items packed in it are small items and call it "blue" otherwise (the blue bins constitute the approximated packing at the beginning). There are two cases to consider.

In the first case, there is no red bin in the final packing of $\mathbb{A}$, i.e., all small items fit in the remaining space of the bins in the approximate packing of large items. Let σ' be a copy of the input sequence in which the sizes of large items are approximated, i.e., increased by at most ε'; also let X be the number of bins for the optimal packing of σ'. Since there is no red bin in the final packing of $\mathbb{A}$, the cost of $\mathbb{A}$ is equal to X. Consider the optimal packing of the actual input sequence σ. Since all bins are good, one can transfer a subset of items to provide an available space of size at least $5\varepsilon'$ in each bin. After such a transfer, we can increase the sizes of large items to their approximate sizes. Since there are at most 5 large items in each bin and also available space of size at least $5\varepsilon'$, the packing constructed this way is a valid packing for the sequence σ'. Since the size of the transferred items for each bin is at most 1/4, the transferred items from each group of four bins can fit in one new bin. Consequently the number of bins in the new packing is at most $5/4\,\textsc{Opt}(\sigma)$. We know that the final packing by $\mathbb{A}$ is the optimal packing for σ' (with cost X), and in particular not worse than the packing constructed above. Hence, the cost of $\mathbb{A}$ is not more than $5/4\,\textsc{Opt}(\sigma)$.

In the second case, there is at least one red bin in the final packing of $\mathbb{A}$. We claim that all bins in the final packing of $\mathbb{A}$, except possibly the last, have levels larger than 3/4. The claim obviously holds for the red bins since the levels of all these bins (excluding the last one) are larger than 5/6. Moreover, since there is a bin which is opened by a small item, all blue bins have levels larger than 5/6, i.e., the total size of packed items and reserved space for the large items is larger than 5/6. Since there are at most 5 large items in each bin, the actual level of each bin in the final packing of $\mathbb{A}$ is at least $5/6 - 5\varepsilon'$, which is not smaller than 3/4 for $\varepsilon' \leq 1/60$. So, all bins, except possibly one, have levels larger than 3/4. Consequently, the algorithm is 4/3-competitive. ◀

It remains to address how to deal with bad bins. The next three lemmas do this.

▶ **Lemma 8.** *Consider sequences for which all bins in the optimal packing include precisely two items. There is an algorithm that receives* 1 *bit of advice per request and achieves an optimal packing.*

▶ **Lemma 9.** *Consider a sequence σ for which all items have sizes larger than* 1/4 *and for which each bin in* Opt*'s packing includes precisely three items. The cost of the Harmonic algorithm is at most $4/3\,\textsc{Opt}(\sigma) + 3$ for serving such a sequence.*

▶ **Lemma 10.** *Consider a sequence σ for which all bins in the optimal packing are bad bins (as defined earlier). There is an algorithm that receives two bits of advice for each request, and opens at most $(4/3 + \frac{5\varepsilon'}{1-5\varepsilon'})\,\textsc{Opt}(\sigma) + 3$ bins.*

Proof. By the definition of bad bins, for any bin in the optimal packing, all items are either smaller than $5\varepsilon'$ or larger than $1/4$. We call the former group of items *tiny* items and pack them separately using the FF strategy. We refer to other items as *normal items*. Consider an offline packing P which is the same as OPT's packing, except that all tiny items are removed from their bins and packed separately in new bins using the FF strategy. This implies that the cost of P is larger than $\text{OPT}(\sigma)$ by a multiplicative factor of at most $1+\frac{5\varepsilon'}{1-5\varepsilon'}$. Let Q be the optimal packing for normal items. Since all normal items are larger than $1/4$, each bin of Q contains at most three items. We say a bin of Q has type i ($i \in \{1,2,3\}$), if it contains i normal items. Similarly, we say an item x has type i if it is packed in a type i bin. All items in type 3 bins have sizes smaller than $1/2$ (otherwise one will have size at most $1/4$ which contradicts the assumption). Moreover, the sizes of the items in all type 1 bins (except possibly the last one) are larger than $1/2$ (otherwise a better packing is achieved by pairing two of them). With two bits of advice, we can detect the type of an item as follows: Let b denote the two bits of advice with item x. If b is "01" and $x > 1/2$, then x has type 1; if b is "01" and $x \leq 1/2$, then x has type 3; and if b is "10" or b is "11", then x has type 2. Note that the code "00" is not used at this point (this is used later on), and the use of "10" and "11" is still to be detailed.

Let X_i denote the number of bins of type i ($1 \leq i \leq 3$). Hence, the cost of Q is $X_1+X_2+X_3$, and consequently the cost of P is at least $X_1+X_2+X_3+X'$, where X' is the number of bins filled by tiny items. Consider an algorithm $\mathbb{A}$ that performs as follows. If an item x has type 1, $\mathbb{A}$ simply opens a new bin for x. If x has type 2, $\mathbb{A}$ applies the strategy of Lemma 8 to place it in one of the bins maintained for items of type 2. Recall that the advice in this case is either "10" or "11", so the second bit provides the advice required by Lemma 8. If x has type 3, $\mathbb{A}$ applies the Harmonic strategy to pack the item in a set of bins maintained for type 3 items. By Lemma 9, the cost of $\mathbb{A}$ for these items is at most $4/3X_3+3$. Finally, $\mathbb{A}$ uses the FF strategy to pack tiny items in separate bins. Consequently, the cost of the algorithm is at most $X_1+X_2+4/3X_3+X'+3 \leq (1+\frac{5\varepsilon'}{1-5\varepsilon'})\,\text{OPT}(\sigma)+X_3/3+3 \leq (4/3+\frac{5\varepsilon'}{1-5\varepsilon'})\,\text{OPT}(\sigma)+3$. ◀

Provided with the above results, we arrive at the following result:

▶ **Theorem 11.** *There is an online algorithm which receives two bits of advice per request, plus an additive lower order term, and achieves a competitive ratio of $4/3+\varepsilon$, for any positive value of ε.*

Proof. Define ε' to be $\frac{11\varepsilon}{60}$. For $\varepsilon < 1/11$, we have $\varepsilon' < 1/60$. Moreover, we have $\frac{5\varepsilon'}{1-5\varepsilon'} \leq \frac{5\varepsilon'}{1-1/12} = \frac{60\varepsilon'}{11} = \varepsilon$. In an optimal packing, divide bins into good and bad bins using Definition 6. Also, let Gd and Bd respectively denote the number of good and bad bins. Use advice bits to distinguish items which are packed in good and bad bins, and pack them in separate lists of bins. More precisely, let the two bits of advice for an item x be "00" if it is packed by OPT in a good bin, and apply Lemma 7 to pack these items in at most $4/3Gd$ bins. Similarly, apply Lemma 10 to pack items from bad bins in at most $(4/3+\frac{5\varepsilon'}{1-5\varepsilon'})Bd+3 \leq (4/3+\varepsilon)Bd+3$ bins, using bits of advice of the form "01", "10", or "11", as discussed in the proof of Lemma 10. Consequently, the cost of the algorithm will be at most $4/3Gd+(4/3+\varepsilon)Bd+3 \leq (4/3+\varepsilon)\,\text{OPT}(\sigma)+3$. ◀

5 A Lower Bound for Linear Advice

The *GMP problem* [13] and the *String Guessing Problem* [6] both contain a core special case of guessing a binary sequence. We use their results to show that an online algorithm needs a linear number of bits of advice to achieve a competitive ratio better than 9/8 for bin packing.

▶ **Definition 12** ([13, 6]). The *Binary String Guessing Problem with known history (2-SGKH)* is the following online problem. The input $I = (n, \sigma = \langle x_1, x_2, \ldots, x_n \rangle)$ consists of n items that are either "0" or "1" and that are revealed one by one. For each item x_t, the online algorithm $\mathbb{A}$ must guess if it is a "0" or a "1". After the algorithm has made a guess, the value of x_t is revealed to the algorithm.

▶ **Lemma 13** ([6]). *On any input of length n, any deterministic algorithm for 2-SGKH that is guaranteed to guess correctly on more than αn bits, for $1/2 \leq \alpha < 1$, needs to read at least $(1 + (1-\alpha)\log(1-\alpha) + \alpha \log \alpha)n$ bits of advice.*

Since the number of bits needed to express the number of "0"s in the input is at most $\lceil \log(n+1) \rceil \leq \log n + 1$, and this number can be given as advice by an oracle, if it is not given to the algorithm otherwise, we easily obtain the following lemma. Recall that the definition of e, the length of the encoding function, is given in Section 1.1.

▶ **Lemma 14.** *Consider instances of size n of the 2-SGKH problem in which the number of "0"s is given to the algorithm as part of the input. For these instances, any deterministic algorithm that is guaranteed to guess correctly on more than αn bits, for $1/2 \leq \alpha < 1$, needs to read at least $(1 + (1-\alpha)\log(1-\alpha) + \alpha \log \alpha)n - e(n)$ bits of advice.*

In order to relate the Binary String Guessing Problem to the online bin packing problem, we introduce another problem called the Binary Separation Problem.

▶ **Definition 15.** The *Binary Separation Problem* is the following online problem. The input $I = (n_1, \sigma = \langle y_1, y_2, \ldots, y_n \rangle)$ consists of $n = n_1 + n_2$ positive values which are revealed one by one. There is a fixed partitioning of the set of items into a subset of n_1 *large* items and a subset of n_2 *small* items, so that all large items are larger than all small items. Upon receiving an item y_i, an online algorithm for the problem must guess if y belongs to the set of small or large items. After the algorithm has made a guess, it is revealed to the algorithm whether y_i actually belongs to class of small or large items.

We provide reductions from the modified Binary String Guessing Problem to the Binary Separation Problem, and from the Binary Separation Problem to the online bin packing problem. In order to reduce a problem P_1 to another problem P_2, given an instance of P_1 defined by a sequence σ_1 and a set of parameters η_1 (such as the length of σ_1 or the number of "0"s in it), we create an instance of P_2 which is defined by a sequence σ_2 and also a set of parameters η_2. In our reductions, we assume η_2 is derived from η_1, and since σ_1 is revealed in an online manner, σ_2 is created in an online manner by looking only at η_1 and the revealed items of σ_1.

▶ **Lemma 16.** *Assume that there is an online algorithm that solves the Binary Separation Problem on sequences of length n with $b(n)$ bits of advice, and makes at most $r(n)$ mistakes. Then there is also an algorithm that solves the Binary String Guessing Problem on sequences of length n, assuming the number of "0"s is given as a part of input, so that the algorithm receives $b(n)$ bits of advice and makes at most $r(n)$ errors.*

Proof. We assume that we have an algorithm Bsa that solves the Binary Separation Problem under the conditions of the lemma statement. Using that algorithm, we define the number n_1 of large items to be the number of "0"s in the instance of the Binary String Guessing Problem. Then, we implement our algorithm Bsga for the Binary String Guessing Problem as outlined in Algorithm 1, which defines the reduction. This Bsga implementation, defined in Algorithm 1, functions as an adversary for Bsa, e.g., in Line 4, Bsga gives Bsa its next

Algorithm 1 Implementing Binary String Guessing via Binary Separation.

The Binary Guessing algorithm knows the number of "0"s (n_1) and passes it as a parameter (the number of large items) to the Binary Separation algorithm

```
small = 0; large = 1
repeat
    mid = (large − small) / 2
    class_guess = SeparationAlgorithm.ClassifyThis(mid)
    if class_guess = "large" then
        bit_guess = 0
    else
        bit_guess = 1
    actual_bit = Guess(bit_guess) {The actual value is received after guessing (2-SGKH).}

    if actual_bit = 0 then
        large = mid {We let "large" be the correct decision.}
    else
        small = mid {We let "small" be the correct decision.}
until end of sequence
```

request. Notice that we ensure that the BSGA makes a correct guess if and only if BSA makes a correct guess. The advice tape is filled with bits of advice for this combined algorithm. The BSGA uses the BSA as a sub-routine, but all the questions are effectively coming from the BSA.

The set-up, reminiscent of binary search, is carried out as specified in the algorithm with the purpose of ensuring that when the BSA is informed of the actual class of the item it considered, no result can contradict information already obtained. Specifically, the next item for the BSA to consider is always in between the largest item which has previously been deemed "small" and the smallest item which has previously been deemed "large". The fact that we give the middle item from that interval is unimportant; any value chosen from the open interval would work. ◀

Now, we prove that if we can solve a special case of the bin packing problem, we can also solve the Binary Separation Problem.

▶ **Lemma 17.** *Consider the bin packing problem on sequences of length $2n$ for which* OPT *opens n bins. Assume that there is an online algorithm $\mathbb{A}$ that solves the problem on these instances with $b(n)$ bits of advice and opens at most $n + r(n)/4$ bins. Then there is also an algorithm* BSA *that solves the Binary Separation Problem on sequences of length n with $b(n)$ bits of advice and makes at most $r(n)$ errors.*

Proof. In the reduction, we encode requests for the BSA as items for bin packing. Assume we are given an instance $I = (n_1, \sigma = \langle y_1, y_2, \ldots, y_n \rangle)$ of the Binary Separation problem, in which n_1 is the number of large items ($n_1 + n_2 = n$), and the values of y_ts are revealed in an online manner ($1 \leq t \leq n$). We create an instance of the bin packing problem which has length $2n$. Algorithm 2 shows the details of the reduction. The bin packing sequence starts with n_1 items of size $\frac{1}{2} + \varepsilon_{min}$ (in Algorithm 2, the variable "NumberOfLargeItems" is n_1 from the Binary Separation Problem). Any algorithm needs to open a bin for each of these n_1 items. We create the next n items in an online manner, so that we can use the result of their packing to guess the requests for the Binary Separation Problem. Let $\tau = y_t$

$(1 \leq t \leq n)$ be a requested item of the Binary Separation Problem; we ask the bin packing algorithm to pack an item whose size is an increasing function of τ, and slightly less than $\frac{1}{2}$. Depending on the decision of the bin packing algorithm for opening a new bin or placing the item in one of the existing bins, we decide the type of τ as being consecutively small or large. The last n_2 items of the bin packing instance are defined as complements of the items in the bin packing instance associated with small items in the binary separation instance (the complement of item x is $1 - x$). We do not need to give the last items complementing the small items in order to implement the algorithm, but we need them for the proof of the quality of the correspondence that we are proving.

Call an item in the bin packing sequence "large" if it is associated with large items in the Binary Separation Problem, and "small" otherwise. For the bin packing sequence produced by the reduction, an optimal algorithm pairs each of the large items with one of the first n_1 items (those with size $\frac{1}{2} + \varepsilon_{min}$), placing them in the first n_1 bins. OPT pairs the small items with their complements, starting one of the next n_2 bins with each of these small items. Hence, the cost of an optimal algorithm is $n_1 + n_2 = n$. The values ε_{min} and ε_{max} in Algorithm 2 must be small enough so that no more than two of any of the items given in the algorithm can fit together in a bin. No other restriction is necessary.

We claim that each extra bin used by the bin packing algorithm, but not by OPT, results in at most four mistakes made by the derived algorithm on the given instance of the Binary Separation Problem. Consider an extra bin in the final packing of $\mathbb{A}$. This bin is opened by a large item which is incorrectly guessed as being small (bins which are opened by small items also appear in OPT's packing). Note that large items do not fit in the same bins as complements of small items. The extra bin has enough space for another large item. Moreover, there are at most two small items which are incorrectly guessed as being large and placed in the space dedicated to the large items of the extra bin. Hence, there is an overhead of at least one for four mistakes. To summarize, $\mathbb{A}$ has to decide if a given item is small or large and performs accordingly, and it pays a cost of at least 1/4 for each incorrect decision. If $\mathbb{A}$ opens at most $n + r(n)/4$ bins, the algorithm derived from $\mathbb{A}$ for the Binary Separation Problem makes at most $r(n)$ mistakes. ◀

▶ **Theorem 18.** *Consider the online bin packing problem on sequences of length n. To achieve a competitive ratio of c $(1 < c < 9/8)$, an online algorithm needs to receive at least* $(n(1+(4c-4)\log(4c-4)+(5-4c)\log(5-4c)) - (\lceil \log(n+1) \rceil + 2\lceil \log(\lceil \log(n+1) \rceil + 1) \rceil + 1))/2$ *bits of advice.*

Proof. Consider a bin packing algorithm $\mathbb{A}$ that receives $b(n)$ bits of advice and achieves a competitive ratio of c. This algorithm opens at most $(c-1)\,\text{OPT}(\sigma)$ bins more than OPT, so when $\text{OPT}(\sigma) = n/2$, it opens at most $(c-1)n/2$ more bins. By Lemma 17, the existence of such an algorithm implies that there is an algorithm $\mathbb{A}$ that solves the Binary Separation Problem on sequences of length $n/2$ with b bits of advice and makes at most $2(c-1)n$ errors. By Lemma 16, this implies that there is an algorithm $\mathbb{B}$ that solves the Binary String Guessing Problem on sequences of length $n/2$ with b bits of advice and makes at most $2(c-1)n$ mistakes, i.e., it correctly guesses the other $n/2 - 2(c-1)n = (5-4c)n/2$ items. Let $\alpha = 5 - 4c$, and note that α is in the range $[1/2, 1)$ when c is in the range $(1, 9/8]$. Lemma 14 implies that in order to correctly guess more than $\alpha n/2$ of the items in the binary sequence, we must have $b(n)$ larger than or equal to $((1 + (1-\alpha)\log(1-\alpha) + \alpha \log \alpha)n - e(n))/2$. Replacing α with $5 - 4c$ completes the proof. ◀

Thus, to obtain a competitive ratio strictly better than 9/8, a linear number of bits of advice is required. For example, to achieve a competitive ratio of 17/16, at least $0.188n$ bits of advice are required asymptotically.

Algorithm 2 Implementing Binary Separation via Special Case Bin Packing.

Choose ε_{min} and ε_{max} so that $0 < \varepsilon_{min} < \varepsilon_{max} < \frac{1}{6}$
Choose a decreasing function $f\colon \mathbb{R} \to (\varepsilon_{min}..\varepsilon_{max})$
for i = 1 to NumberOfLargeItems **do**
  BinPacking.Treat($\frac{1}{2} + \varepsilon_{min}$) {The decision can only be to open a bin.}
repeat
  Let τ be the next request
  decision = BinPacking.Treat($\frac{1}{2} - f(\tau)$)
  if decision = "packed with an $\frac{1}{2} + \varepsilon_{min}$ item" **then**
    class_guess = "large"
  else
    class_guess = "small"
  actual_class = Guess(class_guess)
  if actual_class = "small" **then**
    SmallItems.append($\frac{1}{2} - f(\tau)$) {Collecting small items for later.}
until end of request sequence
for i = 1 to len(SmallItems) **do**
  BinPacking.Treat(1 − SmallItems[i]) {The decision is not used.}

▶ **Corollary 19.** *Consider the bin packing problem for packing sequences of length n. To achieve a competitive ratio of $9/8 - \delta$, in which δ is a small, but fixed positive number, an online algorithm needs to receive $\Omega(n)$ bits of advice.*

6 Concluding Remarks

We conjecture that a sublinear number of bits of advice is enough to achieve competitive ratios smaller than 4/3. Note that our results imply that we cannot hope for ratios smaller than 9/8 with sublinear advice.

References

1 E. Asgeirsson, U. Ayesta, E. Coffman, J. Etra, P. Momcilovic, D. Phillips, V. Vokhshoori, Z. Wang, and J. Wolfe. Closed on-line bin packing. *Acta Cybernet.*, 15(3):361–367, 2002.
2 J. Baewicz and K. Ecker. A linear time algorithm for restricted bin packing and scheduling problems. *Oper. Res. Lett.*, 2(2):80–83, 1983.
3 János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. *Theoret. Comput. Sci.*, 440–441:1–13, 2012.
4 Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovic, and Lucia Keller. Online coloring of bipartite graphs with and without advice. In *COCOON '12*, volume 7434 of *LNCS*, pages 519–530, 2012.
5 Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovic, Sacha Krug, and Björn Steffen. On the advice complexity of the online $L(2,1)$-coloring problem on paths and cycles. In *COCOON '13*, volume 7936 of *LNCS*, pages 53–64, 2013.
6 Hans-Joachim Böckenhauer, Juraj Hromkovic, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. In *COCOON '13*, volume 7936 of *LNCS*, pages 493–505, 2013.
7 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On advice complexity of the k-server problem. In *ICALP '1s1*, volume 6755 of *LNCS*, pages 207–218, 2011.

8 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In *ISAAC '09*, volume 5878 of *LNCS*, pages 331–340, 2009.

9 Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, and Peter Rossmanith. On the advice complexity of the knapsack problem. In *LATIN '12*, volume 7256 of *LNCS*, pages 61–72, 2012.

10 Edward G. Coffman, Michael R. Garey, and David S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation algorithms for NP-hard Problems*. PWS Publishing Co., 1997.

11 Stefan Dobrev, Rastislav Královic, and Euripides Markou. Online graph exploration with advice. In *SIROCCO '12*, volume 7355 of *LNCS*, pages 267–278, 2012.

12 Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO Inform. Theor. Appl.*, 43(3):585–613, 2009.

13 Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. *Theoret. Comput. Sci.*, 412(24):2642 – 2656, 2011.

14 W. Fernandez de la Vega and G. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

15 Michal Forišek, Lucia Keller, and Monika Steinová. Advice complexity of online coloring for paths. In *LATA '12*, volume 7183 of *LNCS*, pages 228–239, 2012.

16 Edward F. Grove. Online binpacking with lookahead. In *SODA '95*, pages 430–436, 1995.

17 Juraj Hromkovič, Rastislav Královič, and Richard Královič. Information complexity of online problems. In *MFCS '10*, volume 6281 of *LNCS*, pages 24–36, 2010.

18 D. Komm and R. Královič. Advice complexity and barely random algorithms. *RAIRO Inform. Theor. Appl.*, 45(2):249–267, 2011.

19 Dennis Komm, Richard Královic, and Tobias Mömke. On the advice complexity of the set cover problem. In *CSR '12*, volume 7353 of *LNCS*, pages 241–252, 2012.

20 Marc P. Renault and Adi Rosén. On online algorithms with advice for the k-server problem. In *WAOA '11*, volume 7164 of *LNCS*, pages 198–210, 2011.

21 Sebastian Seibert, Andreas Sprock, and Walter Unger. Advice complexity of the online coloring problem. In *CIAC '13*, volume 7878 of *LNCS*, pages 345–357, 2013.

22 Steven S. Seiden. On the online bin packing problem. *J. ACM*, 49:640–671, 2002.

23 Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2004.

Balls into bins via local search: cover time and maximum load*

Karl Bringmann[1], Thomas Sauerwald[2], Alexandre Stauffer[3], and He Sun[1]

1 **Max Planck Institute for Informatics, Saarbrücken, Germany**
2 **University of Cambridge, UK**
3 **University of Bath, UK**

Abstract

We study a natural process for allocating m balls into n bins that are organized as the vertices of an undirected graph G. Balls arrive one at a time. When a ball arrives, it first chooses a vertex u in G uniformly at random. Then the ball performs a local search in G starting from u until it reaches a vertex with local minimum load, where the ball is finally placed on. Then the next ball arrives and this procedure is repeated. For the case $m = n$, we give an upper bound for the maximum load on graphs with bounded degrees. We also propose the study of the *cover time* of this process, which is defined as the smallest m so that every bin has at least one ball allocated to it. We establish an upper bound for the cover time on graphs with bounded degrees. Our bounds for the maximum load and the cover time are tight when the graph is vertex transitive or sufficiently homogeneous. We also give upper bounds for the maximum load when $m \geqslant n$.

1998 ACM Subject Classification G.3 Mathematics of Computing: Probability and Statistics

Keywords and phrases Balls and Bins, Stochastic Process, Randomized Algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.187

1 Introduction

A very simple procedure for allocating m balls into n bins is to place each ball into a bin chosen independently and uniformly at random. We refer to this process as *1-choice process*. It is well known that, when $m = n$, the maximum load for the 1-choice process (i.e., the maximum number of balls allocated to any single bin) is $\Theta\left(\frac{\log n}{\log \log n}\right)$ [10]. Alternatively, in the d-choice process, balls arrive sequentially one after the other, and when a ball arrives, it chooses d bins independently and uniformly at random, and places itself in the bin that currently has the smallest load among the d bins (ties are broken uniformly at random). It was shown by Azar et al. [2] and Karp et al. [7] that the maximum load for the d-choice process with $m = n$ and $d \geqslant 2$ is $\Theta\left(\frac{\log \log n}{\log d}\right)$. The constants omitted in the Θ are known and, as shown by Vöcking [11], they can be reduced with a slight modification of the d-choice process. Berenbrink et al. [3] extended these results to the case $m \gg n$.

In some applications, it is important to allow each ball to choose bins in a *correlated* way. For example, such correlations occur naturally in distributed systems, where the bins

* Karl Bringmann is a recipient of the *Google Europe Fellowship in Randomized Algorithms*, and this research is supported in part by this Google Fellowship. The research of Alexandre Stauffer is supported in part by a Marie Curie Career Integration Grant PCIG13-GA-2013-618588 DSRELIS. The research of He Sun has partially been funded by the Cluster of Excellence "Multimodal Computing and Interaction" within the Excellence Initiative of the German Federal Government.



© Karl Bringmann, Thomas Sauerwald, Alexandre Stauffer, and He Sun;
licensed under Creative Commons License CC-BY





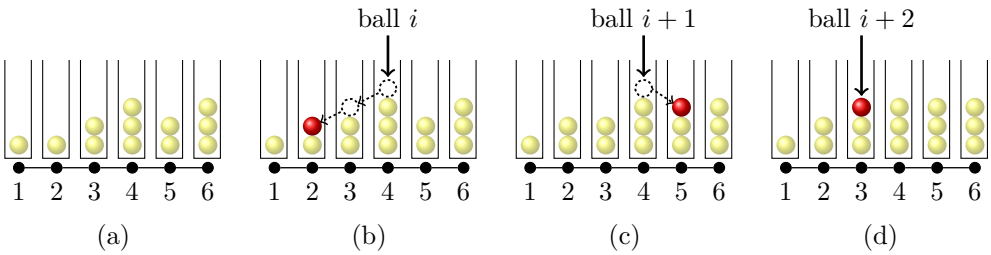


Figure 1 Illustration of the local search allocation. Black circles represent the vertices 1–6 arranged as a path, and yellow circles represent the balls of the process (the most recently allocated ball is marked red). Figure (a) shows the configuration after placing $i-1$ balls. In Figure (b), ball i born at vertex 4 has two choices in the first step of the local search (vertices 3 or 5) and is finally allocated to vertex 2. Figures (c) and (d) show the placement of balls $i+1$ and $i+2$.

represent processors that are interconnected as a graph and the balls represent tasks that need to be assigned to processors. From a practical point of view, letting each task choose d independent random bins may be undesirable, since the cost of accessing two bins which are far away in the graph may be higher than accessing two bins which are nearby. Furthermore, in some contexts, tasks are actually created by the processors, which are then able to forward tasks to other processors to achieve a more balanced load distribution. In such settings, allocating balls close to the processor that created them is certainly very desirable as it reduces the costs of probing the load of a processor and allocating the task.

With this motivation in mind, Bogdan et al. [4] introduced a natural allocation process called *local search allocation*. Consider that the bins are organized as the vertices of a graph $G=(V,E)$ with $n=|V|$. At each time step a ball is "born" at a vertex chosen independently and uniformly at random from V, which we call the birthplace of the ball. Then, starting from its birthplace, the ball performs a local search in G, where the ball repeatedly moves to the adjacent vertex with the smallest load, provided that this load is strictly smaller than the load of its current vertex. We assume that ties are broken independently and uniformly at random. The local search ends when the ball visits the first vertex that is a local minimum, which is a vertex for which no neighbor has a smaller load. After that, the next ball is born and the procedure above is repeated. See Figure 1 for an illustration.

The main result in [4] establishes that when G is an expander graph with bounded maximum degree, the maximum load after n balls have been allocated is $\Theta(\log\log n)$. Hence, local search allocation on bounded-degree expanders achieves the same maximum load (up to constants) as in the d-choice process, but has the extra benefit of requiring only local information during the allocation. In [4], it was also established that the maximum load is $\Theta\left(\left(\frac{\log n}{\log\log n}\right)^{\frac{1}{d+1}}\right)$ on d-dimensional grids, and $\Theta(1)$ on regular graphs of degrees $\Omega(\log n)$.

1.1 Results

In this paper, we derive new upper and lower bounds for the maximum load and propose the study of another natural quantity, which we refer to as the *cover time*. In order to state our results, we need to introduce the following two quantities that are related to the local neighborhood growth of G:

$$R_1 = R_1(G) = \min\{r \in \mathbb{N}\colon r|B_u^r|\log r \geqslant \log n \text{ for all } u \in V\}$$

and

$$R_2 = R_2(G) = \min\{r \in \mathbb{N} \colon r|B_u^r| \geqslant \log n \text{ for all } u \in V\},$$

where B_u^r denotes the set of vertices within distance r from vertex u. Note that $R_1 \leqslant R_2$ for all G. For the sake of clarity, we state our results here for *vertex-transitive* graphs only. In later sections we state our results in full generality, which will require a more refined definition of R_1 and R_2. We also highlight that for all the results below (and throughout this paper) we assume that ties are broken independently and uniformly at random; the impact of tie-breaking procedures in local search allocation was investigated in [4, Theorem 1.5].

Maximum load

We derive an upper bound for the maximum load after n balls have been allocated. Our bound holds for *all* bounded-degree graphs, and is tight for vertex-transitive graphs (and, more generally, for graphs where the neighborhood growth is sufficiently homogeneous across different vertices).

▶ **Theorem 1.1** (Maximum load when $m = n$). *Let G be any vertex-transitive graph with bounded degrees. Then, with probability at least $1 - n^{-1}$, the maximum load after n balls have been allocated is $\Theta(R_1)$.*

Theorem 1.1 is a special case of Theorem 3.1, which gives a more precise version of the result above and generalizes it to non-transitive graphs; in particular, we obtain that for any graph with bounded degrees the maximum load is $\mathcal{O}(R_1)$ with high probability. We state and prove Theorem 3.1 in Section 3.

Note that for bounded-degree expanders we have $R_1 = \Theta(\log \log n)$, and for d-dimensional grids we have $R_1 = \Theta\left(\left(\frac{\log n}{\log \log n}\right)^{\frac{1}{d+1}}\right)$. Hence the results for bounded-degree graphs in [4] are special cases of Theorems 1.1 and 3.1. Furthermore, the proof of Theorems 1.1 and 3.1 uses different techniques (it follows by a subtle coupling with the 1-choice process) and is substantially shorter than the proofs in [4].

Our second result establishes an upper bound for the maximum load when $m \geqslant n$. We point out that all other results known so far were limited to the case $m = n$. We establish that, when $m = \Omega(R_2 n)$, the maximum load is of order $\Theta(m/n)$ (i.e., the same order as the average load). We note that the difference between the maximum load and the average load for the local search allocation is always bounded above by the diameter of the graph. This is in some sense similar to the d-choice process, where the difference between the maximum load and the average load does not depend on m [3].

▶ **Theorem 1.2** (Maximum load when $m \geqslant n$). *Let G be any graph with bounded degrees. Then for any $m \geqslant n$, with probability at least $1 - n^{-1}$, the maximum load after m balls have been allocated is $\mathcal{O}(\frac{m}{n} + R_2)$.*

Cover time

We propose to study the following natural quantity related to any process based on allocating balls into bins. Define the *cover time* as the first time at which all bins have at least one ball allocated to them. This is in analogy with cover time of random walks on graphs, which is the first time at which the random walk has visited all vertices of the graph. Note that for the 1-choice process, the cover time corresponds to the time of a *coupon collector* problem, which is known to be $n \log n + \Theta(n)$ [9, Section 2.4.1]. For the d-choice process with $d = \Theta(1)$, we obtain that the cover time is also of order $n \log n$.

We show that for the local search allocation the cover time can be much smaller than $n \log n$: Our next theorem establishes that the cover time for vertex-transitive bounded-degree graphs is $\Theta(R_2 n)$ with high probability. Since $R_2 = \mathcal{O}(\sqrt{\log n})$ for all connected graphs, it follows that the cover time for any connected, bounded-degree graph is at most $\mathcal{O}(n\sqrt{\log n})$, which is significantly smaller than the cover time of the d-choice process for any $d = \Theta(1)$. In particular, we have $R_2 = \Theta(\log \log n)$ for bounded-degree expanders, and $R_2 = \Theta\Big((\log n)^{\frac{1}{d+1}}\Big)$ for d-dimensional grids.

▶ **Theorem 1.3** (Cover time for bounded-degree graphs). *Let G be any vertex-transitive graph with bounded degrees. Then, with probability at least $1 - n^{-1}$, the cover time of local search allocation on G is $\Theta(R_2 n)$.*

The theorem above is a special case of Theorem 4.2, which we state and prove in Section 4.

Our final result provides a general upper bound on the cover time for dense graphs. Theorem 1.4 below is a special case of Theorem 4.3, which gives an upper bound on the cover time for all regular graphs. We state and prove Theorem 4.3 in Section 4.

▶ **Theorem 1.4** (Cover time for dense graphs). *Let G be any d-regular graph with $d = \Omega(\log n \log \log n)$. Then, with probability at least $1 - n^{-1}$, the cover time is $\Theta(n)$.*

Due to space limitations, we skip some proofs. The full version can be found in [5].

2 Key technical argument

Aside from Theorem 1.4, we assume throughout this paper that G has bounded degrees; i.e., the maximum degree Δ is bounded above by a constant independent of n. We also assume that, in the local search allocation, ties are broken independently and uniformly at random.

For each $m \geqslant 0$ and vertex $v \in V$, let $X_v^{(m)}$ denote the load of v (i.e., the number of balls allocated to v) after m balls have been allocated. Initially we have $X_v^{(0)} = 0$ for all $v \in V$ and, for any $m \geqslant 0$, we have $\sum_{v \in V} X_v^{(m)} = m$. Denote by $X_{\max}^{(m)}$ the maximum load after m balls have been allocated; i.e., $X_{\max}^{(m)} = \max_{v \in V} X_v^{(m)}$. Also, denote by $T_{\text{cov}} = T_{\text{cov}}(G)$ the *cover time* of G, which we define as the first time at which all vertices have load at least 1. More formally, $T_{\text{cov}} = \min\{m \geqslant 0 \colon X_v^{(m)} \geqslant 1 \text{ for all } v \in V\}$.

Let $U_i \in V$ denote the birthplace of ball i and, for each $m \geqslant 0$ and $v \in V$, let $\overline{X}_v^{(m)}$ denote the load of v after m balls have been allocated according to the 1-choice process. Let $\overline{X}_{\max}^{(m)}$ denote the maximum load for the 1-choice process. More formally,

$$\overline{X}_v^{(m)} = \sum_{i=1}^{m} \mathbf{1}\,(U_i = v) \quad \text{and} \quad \overline{X}_{\max}^{(m)} = \max_{v \in V} \overline{X}_v^{(m)} \tag{2.1}$$

We now prove a key technical result (Lemma 2.2 below) that will play a central role in our proofs later. Let $\mu \colon V \to \mathbb{Z}$ be any integer function on the vertices of G that satisfies the following property:

$$\text{for any two neighbors } u, v \in V, \text{ we have } |\mu(u) - \mu(v)| \leqslant 1. \tag{2.2}$$

We see μ as an initial attribution of weights to the vertices of G. Then, for any $m \geqslant 1$, after m balls are allocated, we define the weight of vertex v by

$$W_v^{(m)} = X_v^{(m)} + \mu(v). \tag{2.3}$$

Note that for any $m \geqslant 1$ and $v \in V$, we have that W_v can increase by at most one after each step; i.e., $W_v^{(m)} \in \{W_v^{(m-1)}, W_v^{(m-1)} + 1\}$. The lemma below establishes that a ball cannot be allocated to a vertex with larger weight than the vertex where the ball is born.

▶ **Lemma 2.1.** *Let $m \geqslant 1$ and denote by v the vertex where ball m is born (i.e., $v = U_m$). Let v' be the vertex where ball m is allocated. Then, $W_{v'}^{(m-1)} \leqslant W_v^{(m-1)}$.*

Proof. Assume that $v \neq v'$, thus the local search of ball m visits at least two vertices. Let w be the second vertex visited during the local search. Since v and w are neighbors in G, we have

$$W_w^{(m-1)} = X_w^{(m-1)} + \mu(w) = X_v^{(m-1)} - 1 + \mu(w) \leqslant X_v^{(m-1)} + \mu(v) = W_v^{(m-1)}.$$

Proceeding inductively for each step of the local search, we obtain $W_{v'}^{(m-1)} \leqslant W_v^{(m-1)}$. ◀

For vectors $A = (a_1, a_2, \ldots, a_n)$ and $A' = (a'_1, a'_2, \ldots, a'_n)$ such that $\sum_{i=1}^n a_i = \sum_{i=1}^n a'_i$, we say that A *majorizes* A' if, for each $\kappa = 1, 2, \ldots, n$, the sum of the κ largest entries of A is at least the sum of the κ largest entries of A'. More formally, if $j_1, j_2, \ldots, j_n$ are distinct numbers such that $a_{j_1} \geqslant a_{j_2} \geqslant \cdots \geqslant a_{j_n}$ and $j'_1, j'_2, \ldots, j'_n$ are distinct numbers such that $a'_{j'_1} \geqslant a'_{j'_2} \geqslant \cdots \geqslant a'_{j'_n}$, then A majorizes A' if

$$\sum_{i=1}^{\kappa} a_{j_i} \geqslant \sum_{i=1}^{\kappa} a'_{j'_i} \quad \text{for all } \kappa = 1, 2, \ldots, n. \tag{2.4}$$

Let $\overline{W}_v^{(m)}$ be the weight of vertex v after m balls are allocated according to the 1-choice process; i.e., $\overline{W}_v^{(m)} = \overline{X}_v^{(m)} + \mu(v)$ for all $v \in V$. The lemma below establishes that $\overline{W}^{(m)}$ majorizes $W^{(m)}$ for any m.

▶ **Lemma 2.2.** *For any fixed $m \geqslant 0$, we can couple $W^{(m)}$ and $\overline{W}^{(m)}$ so that, with probability 1, $\overline{W}^{(m)}$ majorizes $W^{(m)}$.*

For the proof of this lemma, we need the following result from [2].

▶ **Lemma 2.3** ([2, Lemma 3.4])**.** *Let $v = (v_1, v_2, \ldots, v_n)$, $u = (u_1, u_2, \ldots, u_n)$ be two vectors such that $v_1 \geqslant v_2 \geqslant \cdots \geqslant v_n$ and $u_1 \geqslant u_2 \geqslant \cdots \geqslant u_n$. If v majorizes u, then also $v + e_i$ majorizes $u + e_i$, where e_i is the ith unit vector.*

Proof of Lemma 2.2. The proof is by induction on m. Clearly, for $m = 0$, we have $W_v^{(0)} = \overline{W}_v^{(0)} = \mu(v)$ for all $v \in V$. Now, assume that we can couple $W^{(m-1)}$ with $\overline{W}^{(m-1)}$ so that $\overline{W}^{(m-1)}$ majorizes $W^{(m-1)}$. Let $i_1, i_2, \ldots, i_n$ be distinct elements of V so that

$$W_{i_1}^{(m-1)} \geqslant W_{i_2}^{(m-1)} \geqslant \cdots \geqslant W_{i_n}^{(m-1)}.$$

Similarly, let $j_1, j_2, \ldots, j_n$ be distinct elements of V so that

$$\overline{W}_{j_1}^{(m-1)} \geqslant \overline{W}_{j_2}^{(m-1)} \geqslant \cdots \geqslant \overline{W}_{j_n}^{(m-1)}.$$

Let ℓ be a uniformly random integer from 1 to n. Then, for the process $(W_v^{(m)})_{v \in V}$, let the birthplace of ball m be vertex i_ℓ and for the process $(\overline{W}_v^{(m)})_{v \in V}$, let the birthplace of ball m be j_ℓ. For the process $(W_v^{(m)})_{v \in V}$, ball m may not necessarily be allocated at vertex i_ℓ, so let us define ι as the integer so that i_ι is the vertex to which ball m is allocated.

In order to prove that $\overline{W}^{(m)}$ majorizes $W^{(m)}$, let us define by $\widetilde{W}^{(m)}$ the vector which is obtained from $W^{(m-1)}$ by allocating ball m to vertex i_ℓ (the birthplace of ball m). Applying Lemma 2.3 gives that $\overline{W}^{(m)}$ majorizes $\widetilde{W}^{(m)}$, since by the induction hypothesis $\overline{W}^{(m-1)}$ majorizes $W^{(m-1)}$. Next observe that

$$W^{(m)} = \widetilde{W}^{(m)} - e_{i_\ell} + e_{i_\iota},$$

so we obtain the vector $W^{(m)}$ from $\widetilde{W}^{(m)}$ by removing one ball from vertex i_ℓ and adding one ball to vertex i_ι. By Lemma 2.1, we have $W^{(m-1)}_{i_\iota} \leqslant W^{(m-1)}_{i_\ell}$. This implies $\widetilde{W}^{(m)}_{i_\ell} = W^{(m-1)}_{i_\ell} + 1 \geqslant W^{(m-1)}_{i_\iota} + 1$ and in turn that $\widetilde{W}^{(m)}$ majorizes $W^{(m)}$. Combining this with the insight that $\overline{W}^{(m)}$ majorizes $\widetilde{W}^{(m)}$ implies that $\overline{W}^{(m)}$ majorizes $W^{(m)}$. This completes the induction and the proof. ◀

Now we illustrate the usefulness of the above result by relating the probability of a vertex to have a certain load to the probability that balls are born in a neighborhood around a vertex. For any two vertices $u, v \in V$, we denote by $d_G(u, v)$ their distance on G.

▶ **Lemma 2.4.** *For any $v \in V$, and any $\ell, m \geqslant 1$, we have*

$$\mathbf{Pr}\left[\, X^{(m)}_v \geqslant \ell \,\right] \geqslant \mathbf{Pr}\left[\, \bigcap\nolimits_{w \in B^{\ell-1}_v} \left\{ \overline{X}^{(m)}_w \geqslant \ell - d_G(v,w) \right\} \right]$$

and

$$\mathbf{Pr}\left[\, X^{(m)}_v \geqslant \ell \,\right] \leqslant \mathbf{Pr}\left[\, \bigcup\nolimits_{w \in V} \left\{ \overline{X}^{(m)}_w \geqslant \ell + d_G(v,w) \right\} \right].$$

Proof. For the first inequality, set $\mu(w) = d_G(v, w)$ for all $w \in V$. Let $\mathcal{A}^{(m)}$ be the event that all vertices have weight at least ℓ after m balls are allocated, and let $\overline{\mathcal{A}}^{(m)}$ be the same event for the 1-choice process. In symbols $\mathcal{A}^{(m)} = \{\min_{u \in V} W^{(m)}_u \geqslant \ell\}$ and $\overline{\mathcal{A}}^{(m)} = \{\min_{u \in V} \overline{W}^{(m)}_u \geqslant \ell\}$. By Lemma 2.2, we have that $\mathbf{Pr}\left[\, \mathcal{A}^{(m)} \,\right] \geqslant \mathbf{Pr}\left[\, \overline{\mathcal{A}}^{(m)} \,\right]$. Clearly, we have that $\mathcal{A}^{(m)}$ implies $\{X^{(m)}_v \geqslant \ell\}$, but the two events are in fact equal since, by the smoothness of the load vector ([4, Lemma 2.2]), $\{X^{(m)}_v \geqslant \ell\}$ implies $\mathcal{A}^{(m)}$. The proof is then complete since $\overline{\mathcal{A}}^{(m)} = \bigcap_{w \in B^\ell_v} \left\{ \overline{X}^{(m)}_w \geqslant \ell - d_G(v,w) \right\}$.

For the second inequality, set $\mu(w) = -d_G(v, w)$ for all $w \in V$. Then define $\mathcal{B}^{(m)}$ to be the event that there exists at least one vertex with weight at least ℓ after m balls are allocated, and let $\overline{\mathcal{B}}^{(m)}$ be the corresponding event for the 1-choice process. Thus, $\mathcal{B}^{(m)} = \{\max_{u \in V} W^{(m)}_u \geqslant \ell\}$ and $\overline{\mathcal{B}}^{(m)} = \{\max_{u \in V} \overline{W}^{(m)}_u \geqslant \ell\}$. Similarly as for the event $\mathcal{A}^{(m)}$, we have that the events $\{X^{(m)}_v \geqslant \ell\}$ and $\mathcal{B}^{(m)}$ are identical. Applying Lemma 2.2 we obtain that $\mathbf{Pr}\left[\, \mathcal{B}^{(m)} \,\right] \leqslant \mathbf{Pr}\left[\, \overline{\mathcal{B}}^{(m)} \,\right] = \mathbf{Pr}\left[\, \bigcup_{w \in V} \left\{ \overline{X}^{(m)}_w \geqslant \ell + d_G(v,w) \right\} \right]$. ◀

▶ Remark. The lemma above states that one can *couple* $\{X^{(m)}_v\}_{v \in V}$ and $\{\overline{X}^{(m)}_v\}_{v \in V}$ so that if $\overline{X}^{(m)}_w \geqslant \ell - d_G(v, w)$ for all $w \in B^{\ell-1}_v$, then $X^{(m)}_v \geqslant \ell$. However, this is not necessarily achieved with the "trivial" coupling where each ball is born at the same vertex for both processes $\{X^{(m)}_v\}_{v \in V}$ and $\{\overline{X}^{(m)}_v\}_{v \in V}$. In other words, knowing that the number of balls born at vertex w is at least $\ell - d_G(v, w)$ for all $w \in B^\ell_v$ does *not* imply that $X^{(m)}_v \geqslant \ell$.

Now we extend the proof of Lemma 2.4 to derive an upper bound on the load of a subset of vertices. The proof of this proposition can be found in the full version [5].

▶ Proposition 2.5. Let $S \subset V$ be fixed and Δ be the maximum degree in G. Then, for all $m \geqslant n$ and $\ell \geqslant \frac{300\Delta m}{n}$ we have $\mathbf{Pr}\left[\sum_{v\in S} X_v^{(m)} \geqslant \ell|S|\right] \leqslant 4\exp\left(-\frac{|S|\ell}{14}\log\left(\frac{\ell n}{m}\right)\right) + \exp\left(-\frac{m}{4}\right)$. Moreover, for any given $u \in V$, it holds that $\mathbf{Pr}\left[X_u^{(m)} \geqslant 2\ell\right] \leqslant 4\exp\left(-\frac{|B_u^\ell|\ell}{14}\log\left(\frac{\ell n}{m}\right)\right) + \exp\left(-\frac{m}{4}\right)$.

In many of our proofs we analyze a continuous-time variant where the number of balls is *not* fixed, but is given by a Poisson random variable with mean m. Equivalently, in this variant balls are born at each vertex according to a Poisson process of rate $1/n$. We refer to this as the *Poissonized* version. We will use the Poissonized versions of both the local search allocation and the 1-choice process in our proofs. Since the probability that a mean-m Poisson random variable takes the value m is of order $\Theta(m^{-1/2})$ we obtain the following relation.

▶ **Lemma 2.6.** *Let $\mathcal{A}$ be an event that holds for the Poissonized version of the local search allocation (respectively, 1-choice process) with probability $1-\varepsilon$ for some $\varepsilon \in (0,1)$. Then, the probability that $\mathcal{A}$ holds for the non-Poissonized version of the local search allocation (respectively, 1-choice process) is at least $1 - \mathcal{O}(\varepsilon\sqrt{m})$.*

3 Maximum Load

We start stating a stronger version of Theorem 1.1 which also holds for non-transitive graphs. For $\gamma \in (0, 1/2]$, let

$$R_1^{(\gamma)} = R_1^{(\gamma)}(G) = \max\big\{r \in \mathbb{N}\colon \text{there exists } S \subseteq V \text{ with } |S| \geqslant n^{\frac{1}{2}+\gamma} \text{ such that } r|B_u^r|\log r < \log n \text{ for all } u \in S\big\}.$$

Note that $R_1^{(\gamma)}$ is non-increasing with γ. Also, when G is vertex transitive, we have $R_1 = R_1^{(\gamma)} + 1$ for all $\gamma \in (0, 1/2]$, because in this case, for any given r, the size of B_u^r is the same for all $u \in V$. The theorem below establishes that, for any bounded-degree graph, if there exists a $\gamma \in (0, 1/2]$ for which $R_1^{(\gamma)} = \Theta(R_1)$, then the maximum load when $m = n$ is $\Theta(R_1)$. In the following, $\omega(1)$ stands for a term that goes to ∞ as $n \to \infty$.

▶ **Theorem 3.1** (General version of Theorem 1.1)**.** *Let G be any graph with bounded degrees. For any $\gamma \in (0, 1/2]$ and $\alpha \geqslant 1$, we have*

$$\mathbf{Pr}\left[X_{\max}^{(n)} < \frac{\gamma R_1^{(\gamma)}}{4}\right] \leqslant n^{-\omega(1)} \quad \text{and} \quad \mathbf{Pr}\left[X_{\max}^{(n)} \geqslant 56\alpha R_1\right] \leqslant 5n^{-\alpha}.$$

Proof. We start establishing a lower bound for $X_{\max}^{(n)}$. Let A be a Poisson random variable with mean 1. We first consider the Poissonized versions of the local search allocation and the 1-choice process (recall the definition of these variants from the paragraph preceding Lemma 2.6). For any $v \in V$ and any $\ell > 0$, Lemma 2.4 gives that

$$\mathbf{Pr}\left[X_v^{(n)} \geqslant \ell\right] \geqslant \prod_{r=0}^{\ell-1} \left(\mathbf{Pr}\left[A \geqslant \ell - r\right]\right)^{|N_v^r|} \geqslant \prod_{r=0}^{\ell-1} \left(\mathrm{e}^{-1}(\ell-r)^{-\ell+r}\right)^{|N_v^r|},$$

where N_v^r is the set of vertices at distance r from v so that $B_v^\ell = \bigcup_{r=0}^{\ell} N_v^r$. Hence,

$$\mathbf{Pr}\left[X_v^{(n)} \geqslant \ell\right] \geqslant \exp\left(-|B_v^\ell| - \ell|B_v^\ell|\log(\ell)\right) \geqslant \exp\left(-2\ell|B_v^\ell|\log(\ell)\right),$$

where the last step follows for all $\ell \geqslant 2$. Given $\gamma > 0$, set $\ell = \frac{\gamma R_1^{(\gamma)}}{4}$. Since $|B_v^r| \log r$ is increasing with r, there exists a set S with $|S| = \lceil n^{\frac{1}{2}+\gamma} \rceil$ such that

$$\mathbf{Pr}\left[X_v^{(n)} \geqslant \frac{\gamma R_1^{(\gamma)}}{4} \right] \geqslant \exp\left(-\frac{\gamma R_1^{(\gamma)} |B_v^{R_1^{(\gamma)}}| \log(R_1^{(\gamma)})}{2} \right) \geqslant n^{-\gamma/2} \quad \text{for all } v \in S. \quad (3.1)$$

Let $Y = Y(\gamma)$ be the random variable defined as the number of vertices v satisfying $X_v^{(n)} \geqslant \frac{\gamma R_1^{(\gamma)}}{4}$. Let K be the total number of balls allocated in the Poissonized version of the local search allocation. Note that $\mathbf{E}[K] = n$ and by standard tail bounds, $\mathbf{Pr}[K > 2en] \leqslant 2^{1-2ne}$. Regard Y as a function of the K independently chosen birthplaces $U_1, U_2, \ldots, U_K$. Then, for any given K, Y is 1-Lipschitz by [4, Lemma 2.5], and (3.1) implies that

$$\mathbf{E}[Y \mid K \leqslant 2en] \geqslant n^{\frac{1}{2}+\gamma} \cdot \left(\frac{n^{-\gamma/2} - \mathbf{Pr}[K > 2en]}{\mathbf{Pr}[K \leqslant 2en]} \right) \geqslant \frac{n^{\frac{1}{2}+\frac{\gamma}{2}}}{2}.$$

With this, we apply the method of bounded differences [8, Lemma 1.2] to obtain

$$\begin{aligned}
&\mathbf{Pr}\left[X_{\max}^{(n)} < \frac{\gamma R_1^{(\gamma)}}{4} \right] \\
&\leqslant \mathbf{Pr}\left[|Y - \mathbf{E}[Y \mid K \leqslant 2en]| \geqslant \frac{1}{2}\mathbf{E}[Y \mid K \leqslant 2en] \mid K \leqslant 2en \right] + \mathbf{Pr}[K > 2en] \\
&\leqslant n^{-\omega(1)} + 2^{1-2ne} = n^{-\omega(1)}.
\end{aligned}$$

This result can then be translated to the non-Poissonized model via Lemma 2.6.

Now we establish the upper bound, where we consider the non-Poissonized process. For any fixed $u \in V$, we have from the second part of Proposition 2.5 (with $m = n$) that

$$\begin{aligned}
\mathbf{Pr}\left[X_u^{(n)} \geqslant 56\alpha R_1 \right] &\leqslant 4\exp\left(-\frac{28\alpha R_1 |B_u^{28\alpha R_1}|}{14} \log(28\alpha R_1) \right) + \exp\left(-\frac{n}{4}\right) \\
&\leqslant 4\exp\left(-2\alpha R_1 |B_u^{R_1}| \log R_1\right) + \exp\left(-\frac{n}{4}\right) \leqslant 5n^{-2\alpha}.
\end{aligned}$$

Taking the union bound over u we obtain that $\mathbf{Pr}\left[X_{\max}^{(n)} \geqslant 56\alpha R_1 \right] \leqslant 5n^{-2\alpha+1} \leqslant 5n^{-\alpha}$. ◀

Proof of Theorem 1.2. Applying Proposition 2.5 with $\ell = \left(\frac{m}{n} + R_2\right) c$ for any constant $c \geqslant 300\Delta$, we obtain

$$\begin{aligned}
&\mathbf{Pr}\left[\sum_{u \in B_u^{R_2}} X_u^{(m)} \geqslant \left(\frac{m}{n} + R_2\right) c \cdot |B_u^{R_2}| \right] \\
&\leqslant 4\exp\left(-\left(\frac{m}{n} + R_2\right) \frac{c|B_u^{R_2}|}{14} \log c \right) + \exp\left(-\frac{m}{4}\right) \\
&\leqslant 4\exp\left(-\frac{cR_2|B_u^{R_2}|}{14} \log c \right) + \exp\left(-\frac{m}{4}\right),
\end{aligned}$$

where $B_u^{R_2}$ denotes the set of vertices within distance R_2 from u. By setting $c > 0$ sufficiently large, the right-hand side above can be made smaller than n^{-2}. If u has load k, then the number of balls allocated to vertices in $B_u^{R_2}$ is at least

$$\sum_{i=0}^{R_2} (k-i)|N_u^i| \geqslant (k - R_2)|B_u^{R_2}|.$$

Therefore, on the event $\sum_{u \in B_u^{R_2}} X_u^{(m)} \leqslant \left(\frac{m}{n} + R_2\right) c|B_u^{R_2}|$, we have $X_u^{(m)} \leqslant c\left(\frac{m}{n} + R_2\right) + R_2 \leqslant 2c\left(\frac{m}{n} + R_2\right)$. Taking a union bound over all $u \in V$ completes the proof. ◀

4 Cover time

The proposition below gives an upper bound for the cover time.

▶ Proposition 4.1. Let G be a graph with bounded degrees. Then for any $\alpha > 1$ there exists a $C = C(\alpha) > 0$ such that for all $m \geqslant CR_2 n$ we have $\mathbf{Pr}\left[X^{(m)}_{\min} < \frac{m}{224n\log\Delta}\right] \leqslant n^{-\alpha}$, where $X^{(m)}_{\min} = \min_{v\in V} X^{(m)}_v$.

Proof. Fix an arbitrary vertex $u \in V$. We will use the concept of weights defined in Section 2. Define $\mu(v) = d_G(u,v)$ and $W^{(m)}_v = X^{(m)}_v + \mu(v)$. Similarly, for the 1-choice process, define $\overline{W}^{(m)}_v = \overline{X}^{(m)}_v + \mu(v)$. Let $Y := \min_{v\in V} \overline{W}^{(m)}_v$ be the minimum weight of all vertices in V in the 1-choice process. Let $\ell = \frac{m}{28n\log\Delta}$ and recall that B^r_u is the set of vertices within distance r from u. We have

$$\begin{aligned}\mathbf{Pr}\left[Y < \ell\right] &= \mathbf{Pr}\left[\bigcup\nolimits_{v\in B^{\ell-1}_u}\left\{\overline{W}^{(m)}_v < \ell\right\}\right] \leqslant |B^\ell_u|\mathbf{Pr}\left[\overline{X}^{(m)}_u < \ell\right]\\ &\leqslant |B^\ell_u|\mathbf{Pr}\left[\left|\overline{X}^{(m)}_u - \mathbf{E}\left[\overline{X}^{(m)}_u\right]\right| > \frac{m}{n}\left(1 - \frac{1}{28\log\Delta}\right)\right].\end{aligned}$$

Using a variant of Hoeffding's inequality, we obtain

$$\begin{aligned}\mathbf{Pr}\left[Y < \ell\right] &\leqslant |B^\ell_u|\exp\left(-\frac{\frac{m^2}{n^2}\left(1-\frac{1}{28\log\Delta}\right)^2}{\frac{7m}{3n}}\right)\\ &\leqslant |B^\ell_u|\exp\left(-\frac{3m}{28n}\right) \leqslant \exp\left(\frac{m}{28n} - \frac{3m}{28n}\right) \leqslant \frac{1}{2},\end{aligned}$$

where the last inequality holds since $m/n \geqslant CR_2 = \omega(1)$ for bounded degree graphs. Now define $\overline{Z}$ as the sum of the $|B^{R_2}_u|$ smallest values of $\left\{\overline{W}^{(m)}_v : v \in V\right\}$ and Z as the sum of the $|B^{R_2}_u|$ smallest values of $\left\{W^{(m)}_v : v \in V\right\}$. By Lemma 2.2, we can couple $W^{(m)}$ and $\overline{W}^{(m)}$ so that, with probability 1, $Z \geqslant \overline{Z}$. Further, $\mathbf{E}\left[\overline{Z}\right] \geqslant \frac{\ell|B^{R_2}_u|}{2}$. We now apply Azuma's inequality [6, Theorem 6.1] in order to show that $\overline{Z}$ is likely to be at least $\frac{\ell|B^{R_2}_u|}{4}$. Let $A_1, A_2, \ldots, A_m$ be the martingale adapted to the filtration $\mathcal{F}_i$ generated by $U_1, U_2, \ldots, U_i$; i.e., $A_i = \mathbf{E}\left[\overline{Z} \mid \mathcal{F}_i\right]$. Since changing the birthplace of ball i (and keeping all other birthplaces the same) can change Z by at most one [4, Lemma 2.5], we have that $\mathbf{E}\left[A_i - A_{i-1} \mid \mathcal{F}_{i-1}\right] \leqslant 1$.

Now fix i. Let ζ_u be the value of A_i when $U_i = u$ and let $\overline{\zeta} = \frac{1}{n}\sum_{u\in V}\zeta_u$. Then we have

$$\mathbf{E}_{U_i}\left[(A_i - A_{i-1})^2 \,\middle|\, \bigcap\nolimits_{j=1}^{i-1}\{U_j = u_j\}\right] = \frac{1}{n}\sum_{u\in V}(\zeta_u - \overline{\zeta})^2,$$

where the expectation above is taken with respect to U_i. Since $|\zeta_u - \zeta_{u'}| \leqslant 1$ for all $u, u' \in V$, we can write

$$\frac{1}{n}\sum_{u\in V}(\zeta_u - \overline{\zeta})^2 \leqslant \frac{1}{n}\sum_{u\in V}|\zeta_u - \overline{\zeta}| = \frac{1}{n}\sum_{u\in V}\left|\sum_{u'\in V}\frac{1}{n}(\zeta_u - \zeta_{u'})\right| \leqslant \frac{1}{n^2}\sum_{u\in V}\sum_{u'\in V}|\zeta_u - \zeta_{u'}|.$$

Note that, for any realization of $U_1, U_2, \ldots, U_{i-1}, U_{i+1}, \ldots, U_m$, ζ_u and $\zeta_{u'}$ only differ if exactly one of u or u' is among the $|B^{R_2}_u|$ smallest loads. Hence, $\sum_{u\in V}\sum_{u'\in V}|\zeta_u - \zeta_{u'}| \leqslant 2|B^{R_2}_u|n$. Consequently, $\mathbf{E}_{U_i}\left[(A_i - A_{i-1})^2 \,\middle|\, \bigcap_{j=1}^{i-1}\{U_j = u_j\}\right] \leqslant \frac{2|B^{R_2}_u|}{n}$. Now, Azuma's

inequality [6, Theorem 6.1] gives

$$\mathbf{Pr}\left[\overline{Z} < \frac{\ell|B_u^{R_2}|}{4}\right] \leqslant \mathbf{Pr}\left[\,|\overline{Z} - \mathbf{E}\left[\,\overline{Z}\,\right]| \geqslant \frac{1}{2}\mathbf{E}\left[\,\overline{Z}\,\right]\right] \leqslant \exp\left(-\frac{\left(\frac{1}{2}\mathbf{E}\left[\,\overline{Z}\,\right]\right)^2}{4\cdot\frac{|B_u^{R_2}|}{n}\cdot m + \frac{1}{6}\mathbf{E}\left[\,\overline{Z}\,\right]}\right).$$

Clearly, $\mathbf{E}\left[\,\overline{Z}\,\right] \leqslant \frac{m|B_u^{R_2}|}{n}$, which gives that

$$\mathbf{Pr}\left[\overline{Z} < \frac{\ell|B_u^{R_2}|}{4}\right] \leqslant \exp\left(-\frac{\mathbf{E}\left[\,\overline{Z}\,\right]^2}{16\cdot\frac{|B_u^{R_2}|}{n}\cdot m + \frac{2m|B_u^{R_2}|}{3n}}\right) \leqslant \exp\left(-\frac{\ell^2|B_u^{R_2}|/4}{17m/n}\right).$$

Using the value of ℓ and m, we have

$$\mathbf{Pr}\left[\overline{Z} < \frac{\ell|B_u^{R_2}|}{4}\right] \leqslant \exp\left(-\frac{\frac{m}{n}|B_u^{R_2}|}{68(28\log\Delta)^2}\right) \leqslant \exp\left(-\frac{CR_2|B_u^{R_2}|}{68(28\log\Delta)^2}\right) \leqslant n^{-\frac{C}{68(28\log\Delta)^2}}.$$

Due to our coupling which gives $Z \geqslant \overline{Z}$ we conclude that with probability at least $1 - n^{-\frac{C}{68(28\log\Delta)^2}}$ there exists a vertex $v \in B_u^{R_2}$ with $W_v^{(m)} \geqslant \frac{\ell}{4}$ and thus $X_v^{(m)} \geqslant \frac{\ell}{4} - R_2$. Then, by smoothness of the load vector [4, Lemma 2.2], we have that with probability at least $1 - n^{-\frac{C}{68(28\log\Delta)^2}}$, every vertex in $B_u^{R_2}$ has load at least $\frac{\ell}{4} - 3R_2 \geqslant \frac{m}{224n\log\Delta}$, where the last step follows for all $C \geqslant 672\log\Delta$. The result follows by taking the union bound over all $u \in V$, which yields that, with probability at least $1 - n^{-\frac{C}{68(28\log\Delta)^2}+1}$, all vertices have load at least $\frac{m}{224n\log\Delta}$. The proof is completed by setting C large enough with respect to α so that $\frac{C}{68(28\log\Delta)^2} - 1 \geqslant \alpha$. ◀

We prove a stronger version of Theorem 1.3, which holds also for non-transitive graphs. For $\gamma \in (0, 1/2]$, let

$$R_2^{(\gamma)} = R_2^{(\gamma)}(G) = \max\big\{r \in \mathbb{N}\colon \text{there exists } S \subseteq V \text{ with } |S| \geqslant n^{\frac{1}{2}+\gamma} \text{ such that } r|B_u^r| < \log n \text{ for all } u \in S\big\}.$$

Note that $R_2^{(\gamma)}$ is non-increasing with γ. Also, when G is vertex transitive, we have $R_2 = R_2^{(\gamma)} + 1$ for all $\gamma > 0$, because in this case, for any given r, the size of B_u^r is the same for all $u \in V$. The theorem below establishes that, for any bounded-degree graph, if there exists a $\gamma \in (0, 1/2]$ for which $R_2^{(\gamma)} = \Theta(R_2)$, then the cover time is $\Theta(R_2)$.

▶ **Theorem 4.2** (General version of Theorem 1.3)**.** *Let G be any graph with bounded degrees. For any $\gamma \in (0, 1/2]$ and $\alpha \geqslant 1$, there exists $C = C(\alpha, \Delta)$ such that*

$$\mathbf{Pr}\left[T_{\mathrm{cov}} < \frac{\gamma R_2^{(\gamma)} n}{8\Delta}\right] \leqslant n^{-\omega(1)} \quad \text{and} \quad \mathbf{Pr}\left[\,T_{\mathrm{cov}} \geqslant CR_2 n\,\right] \leqslant n^{-\alpha}.$$

Proof. The second inequality is established by Proposition 4.1. For the first inequality, let S be a set of $n^{\frac{1}{2}+\gamma}$ vertices u for which $R_2^{(\gamma)} \cdot B_u^{R_2^{(\gamma)}} < \log n$. Let $m = \frac{\gamma R_2^{(\gamma)} n}{8\Delta}$. We consider the Poissonized version of the local search allocation and the 1-choice process. We abuse notation slightly and let $X_v^{(m)}$ and $\overline{X}_v^{(m)}$ denote the load of v for the Poissonized version of the local search allocation and 1-choice process, respectively, when the expected number of

balls allocated in total is m. For any $u \in S$, we will bound the probability that $X_u^{(m)} = 0$. By the second part of Lemma 2.4, we have that

$$\mathbf{Pr}\left[X_u^{(m)} = 0\right] \geqslant \mathbf{Pr}\left[\bigcap\nolimits_{w \in V}\left\{\overline{X}_w^{(m)} \leqslant d_G(u,w)\right\}\right].$$

Recall that N_u^r is the set of vertices at distance r from u and $B_u^\ell = \bigcup_{r=0}^{\ell} N_u^r$. By independence of the Poissonized model, we can write

$$\begin{aligned}
\mathbf{Pr}\left[X_u^{(m)} = 0\right] &\geqslant \mathbf{Pr}\left[\bigcap\nolimits_{w \in B_u^{R_2^{(\gamma)}}}\left\{\overline{X}_w^{(m)} = 0\right\}\right]\mathbf{Pr}\left[\bigcap\nolimits_{i > R_2^{(\gamma)}}\bigcap\nolimits_{w \in N_u^i}\left\{\overline{X}_w^{(m)} \leqslant i\right\}\right] \\
&\geqslant \exp\left(-\frac{m|B_u^{R_2^{(\gamma)}}|}{n}\right)\left(1 - \sum\nolimits_{i > R_2^{(\gamma)}}\sum\nolimits_{w \in N_u^i}\mathbf{Pr}\left[\overline{X}_w^{(m)} > i\right]\right) \\
&\geqslant \exp\left(-\frac{m|B_u^{R_2^{(\gamma)}}|}{n}\right)\left(1 - 2\sum\nolimits_{i > R_2^{(\gamma)}}\sum\nolimits_{w \in N_u^i}\left(\frac{me}{ni}\right)^i\right),
\end{aligned}$$

where the last inequality follows by a Chernoff bound [1, Theorem A.1.15]. Using the simple bound $|N_u^i| \leqslant \Delta^i$ and the fact that $\frac{me\Delta}{ni} \leqslant \frac{1}{2}$ for all $i \geqslant R_2^{(\gamma)}$ (as $\Delta/R_2^{(\gamma)} = o(1)$ since $\Delta = \mathcal{O}(1)$), we have

$$\mathbf{Pr}\left[X_u^{(m)} = 0\right] \geqslant \exp\left(-\frac{m|B_u^{R_2^{(\gamma)}}|}{n}\right)\left(1 - 4\left(\frac{me\Delta}{nR_2^{(\gamma)}}\right)^{R_2^{(\gamma)}}\right) \geqslant n^{-\gamma/8} \cdot \frac{1}{2}.$$

Now let Y be the random variable defined as the number of vertices v satisfying $X_v^{(m)} = 0$. Let K be the random variable for the total number of balls allocated and regard Y as a function of the K independently chosen birthplaces $U_1, U_2, \ldots, U_K$. Then, Y is 1-Lipschitz by [4, Lemma 2.5] for any given K. The calculations above give that

$$\mathbf{E}\left[Y \mid K \leqslant 2em\right] \geqslant \mathbf{E}\left[Y\right] \geqslant \frac{n^{\frac{1}{2}+\frac{7\gamma}{8}}}{2}.$$

Note that $m = \mathcal{O}(n \log n)$ for any G. With this, we apply the method of bounded differences [8, Lemma 1.2] and a standard tail bound to obtain

$$\begin{aligned}
&\mathbf{Pr}\left[X_{\min}^{(n)} = 0\right] \\
&\leqslant \mathbf{Pr}\left[\left|Y - \mathbf{E}\left[Y \mid K \leqslant 2em\right]\right| \geqslant \frac{1}{2}\mathbf{E}\left[Y \mid K \leqslant 2em\right] \middle| K \leqslant 2em\right] + \mathbf{Pr}\left[K > 2em\right] \\
&\leqslant 2\exp\left(-\frac{n^{1+14\gamma/8}}{8(2em)}\right) + 2^{1-2me} = n^{-\omega(1)}.
\end{aligned}$$

This result can then be translated to the non-Poissonized process using Lemma 2.6 and the fact that $m = \frac{\gamma R_2^{(\gamma)} n}{4} = \mathcal{O}(n \log n)$. ◀

We now state a stronger version of Theorem 1.4. The proof is in the full version [5].

▶ **Theorem 4.3** (General version of Theorem 1.4). *Let G be any d-regular graph. Then, for any $\alpha > 1$ there exists $C = C(\alpha) > 0$ such that*

$$\mathbf{Pr}\left[T_{\text{cov}} \geqslant C \cdot \left(n\left(1 + \frac{\log n \cdot \log d}{d}\right)\right)\right] \leqslant n^{-\alpha}.$$

5 Remarks and open questions

Blanket time

In analogy with the cover time for random walks, for each $\delta > 1$, we can define the blanket time as the first time at which the load of each vertex is in the interval $(\frac{1}{\delta} \cdot \frac{m}{n}, \delta \cdot \frac{m}{n})$. It follows from Theorem 1.2 and Proposition 4.1 that, for bounded-degree vertex-transitive graphs, the blanket time is $\Theta(nR_2)$ for all large enough δ.

Extreme graphs

Note that for any connected graph G, we have $R_1(G) \leqslant \sqrt{\frac{\log n}{\log\log n}}$ and $R_2(G) \leqslant \sqrt{\log n}$. Thus, the cycle is the graph with the largest possible maximum load (when $m = n$) and largest possible cover time among all bounded-degree graphs up to constant factors. Also, for any graph G with bounded degrees, we have $R_1(G)$ and $R_2(G)$ are of order $\Omega(\log\log n)$. Thus, bounded-degree expanders are the graphs with the smallest maximum load (when $m = n$) and smallest cover time among all bounded-degree graphs up to constant factors.

Open questions

1. For any vertex-transitive graph (not necessarily of bounded degrees), does it hold that $X^{(n)}_{\max} = \Theta(R_1)$ and $T_{\text{cov}} = \Theta(R_2 n)$ with high probability?
2. For any vertex-transitive graph (not necessarily of bounded degrees) and any $m = \omega(nR_2)$, does it hold that $X^{(m)}_{\max} = \frac{m}{n} + \Theta(R_2)$ with high probability?
3. For any vertex-transitive graph, is the blanket time of order nR_2 for all $\delta > 1$? Also, is the blanket time of the same order as the cover time for all vertex-transitive graphs?
4. Let $G = (V, E)$ and $G' = (V, E')$ be two graphs such that $E \subset E'$. Is the maximum load on G stochastically dominated by the maximum load on G' for any m?

References

1 N. Alon and J.H. Spencer. *The probabilistic method.* John Wiley & Sons, 3rd edition, 2008.

2 Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, 1999.

3 Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: The heavily loaded case. *SIAM J. Comput.*, 35(6):1350–1385, 2006.

4 P. Bogdan, T. Sauerwald, A. Stauffer, and H. Sun. Balls into bins via local search. In *Proc. of the 24th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 16–34, 2013.

5 K. Bringmann, T. Sauerwald, H. Sun, and A. Stauffer. Balls into bins via local search: cover time and maximum load, 2013. Preprint at arXiv:1310.0801.

6 F. Chung and L. Lu. Concentration inequalities and Martingale inequalities: a survey. *Internet Mathematics*, 3:79–127, 2006.

7 Richard M. Karp, Michael Luby, and Friedhelm Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. *Algorithmica*, 16(4/5):517–542, 1996.

8 C. McDiarmid. On the method of bounded differences. *Surveys in Combinatorics*, 141:148–188, 1989.

9 M. Mitzenmacher and E. Upfal. *Probability and Computing: randomized algorithms and probabilistic analysis.* Cambridge University Press, 2005.

10 Martin Raab and Angelika Steger. Balls into bins – a simple and tight analysis. In *2nd Int'l Workshop on Randomization and Computation (RANDOM'98)*, pages 159–170, 1998.

11 Berthold Vöcking. How asymmetry helps load balancing. *J. ACM*, 50(4), 2003.

Meet Your Expectations With Guarantees: Beyond Worst-Case Synthesis in Quantitative Games*

Véronique Bruyère[1], Emmanuel Filiot[2], Mickael Randour[1], and Jean-François Raskin[2]

1 **Computer Science Department, Université de Mons (UMONS), Belgium**
2 **Département d'Informatique, Université Libre de Bruxelles (U.L.B.), Belgium**

Abstract

Classical analysis of two-player quantitative games involves an adversary (modeling the environment of the system) which is purely antagonistic and asks for strict guarantees while Markov decision processes model systems facing a purely randomized environment: the aim is then to optimize the expected payoff, with no guarantee on individual outcomes. We introduce the beyond worst-case synthesis problem, which is to construct strategies that guarantee some quantitative requirement in the worst-case while providing an higher expected value against a particular stochastic model of the environment given as input. We consider both the mean-payoff value problem and the shortest path problem. In both cases, we show how to decide the existence of finite-memory strategies satisfying the problem and how to synthesize one if one exists. We establish algorithms and we study complexity bounds and memory requirements.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases two-player games on graphs, Markov decision processes, quantitative objectives, synthesis, worst-case and expected value, mean-payoff, shortest path

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.199

1 Introduction

Two-player zero-sum quantitative games [14, 28, 3] and Markov decision processes (MDPs) [24, 5] are two popular formalisms for modeling decision making in adversarial and uncertain environments respectively. In the former, two players compete with opposite goals (zero-sum), and we want strategies for player 1 (the system) that ensure a given *minimal performance against all possible strategies* of player 2 (its environment). In the latter, the system plays against a stochastic model of its environment, and we want strategies that ensure a *good expected overall performance*. Those two models are well studied and simple optimal memoryless strategies exist for classical objectives such as mean-payoff [22, 14, 15] or shortest path [1, 12]. But both models have clear weaknesses: strategies that are good for the worst-case may exhibit suboptimal behaviors in probable situations while strategies that are good for the expectation may be terrible in some unlikely but possible situations.

In practice, we want strategies that both ensure (a) some worst-case threshold no matter how the adversary behaves (i.e., against any arbitrary strategy) and (b) a good expectation

* Work partially supported by European project CASSTING (FP7-ICT-601148). Filiot and Randour are respectively F.R.S.-FNRS research associate and research fellow. Raskin is supported by ERC Starting Grant inVEST (279499).



© Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 199–213



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



against the expected behavior of the adversary (given as a stochastic model). We study how to construct such finite-memory strategies. We consider finite memory for player 1 as it can be implemented in practice (as opposed to infinite memory). Player 2 is not restricted in his choice of strategies, but we show that simple strategies suffice. Our problem, the **beyond worst-case synthesis problem**, makes sense for any quantitative measure. We focus on two classical ones: the *mean-payoff*, and the *shortest path*.

Example. Consider the weighted game in Fig. 1 to illustrate the *shortest path* context. Circle states belong to player 1, square states to player 2, integer labels are durations in minutes, and fractions are probabilities that model the expected behavior of player 2. Player 1 wants a strategy to go from "home" to "work" such that "work" is *guaranteed* to be reached within 60 minutes (to avoid missing an important meeting), and player 1 would also like to minimize the expected time to reach "work". The strategy that minimizes the expectation is to take the car (expectation is 33 minutes) but it is excluded as there is a possibility to arrive after 60 minutes (in case of heavy traffic). Bicycle is safe but the expectation of this solution is 45 minutes. We can do better with the following strategy: try to take the train, if the train is delayed three time consecutively, then go back home and take the bicycle. This strategy is safe as it always reaches "work" within 59 minutes and its expectation is $\approx 37,56$ minutes (so better than taking directly the bicycle). Our algorithms are able to decide the existence of (and synthesize) such finite-memory strategies.

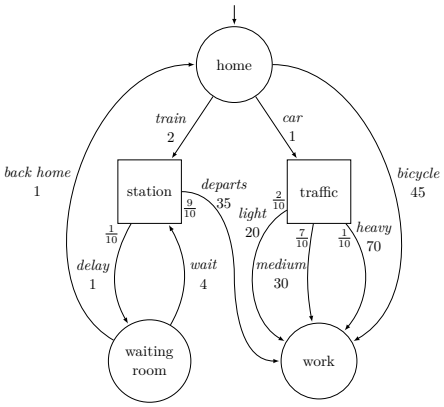


Figure 1 Player 1 wants to minimize its expected time to reach "work", but while ensuring it is less than an hour in all cases.

Contributions. For the mean-payoff, we provide an NP ∩ coNP algorithm (Thm. 7), which would be in P if mean-payoff games were proved to be in P, a long-standing open problem [3, 7]. For the shortest path, we give a pseudo-polynomial time algorithm (Thm. 9), and show that the problem is NP-hard (Thm. 11). For both, synthesized strategies may require up to pseudo-polynomial memory (Thm. 8 and Thm. 10), but accept natural, elegant representations, based on states of the game and simple integer counters. An extended version of this work, including full proofs, can be found in [4].

Related work. Our problems generalize the corresponding problems for two-player zero-sum games and MDPs. In mean-payoff games, optimal memoryless worst-case strategies exist and the best known algorithm is in NP ∩ coNP [14, 28, 3]. For shortest path games, where we consider game graphs with strictly positive weights and try to minimize the cost to target, it can be shown that memoryless strategies also suffice, and the problem is in P. In MDPs, optimal expectation strategies are studied in [24, 15] for both measures: memoryless strategies suffice and they can be computed in P. Our strategies are *strongly risk averse*: they avoid at all cost outcomes below a given threshold (no matter their probability), and inside the set of those *safe* strategies, we maximize expectation. To the best of our knowledge, we are the first to consider such strategies. Other notions of risk have been studied for MDPs: e.g., in [27], the authors want to find policies minimizing the probability (risk) that the total discounted

rewards do not exceed a specified value; in [16], the authors want to achieve a specified value of the long-run limiting average reward at a given probability level (percentile). While those strategies limit risk, they only ensure *low probability* for bad behaviors but not their absence, furthermore, they do not ensure good expectation either. Another body of related work is the study of strategies in MDPs that achieve a trade-off between the expectation and the variance over the outcomes (e.g., [2] for the mean-payoff, [23] for the cumulative reward), giving a statistical measure of the stability of the performance. In our setting, we strengthen this requirement by asking for *strict guarantees on individual outcomes*, while maintaining an appropriate expected payoff.

Future work. Study of other value functions, extension to more general settings (decidable classes of imperfect information games [13], multi-dimension [6, 9], etc), and application to practical cases.

Acknowledgments. We thank G. Latouche and G. Louchard for fruitful discussions about Chernoff bounds in Markov models, and an anonymous reviewer for pointing out interesting related works.

2 Beyond Worst-Case Synthesis

Weighted directed graphs. A *weighted directed graph* is a tuple $\mathcal{G} = (S, E, w)$ where (i) S is the set of vertices, called *states*; (ii) $E \subseteq S \times S$ is the set of directed edges; and (iii) $w\colon E \to \mathbb{Z}$ is the weight function. Given $s \in S$, let $\mathsf{Succ}(s) = \{s' \in S \mid (s, s') \in E\}$ be its set of successors. We assume that for all $s \in S$, $\mathsf{Succ}(s) \neq \emptyset$ (no deadlock). We denote by W the largest absolute weight.

A *play* in $\mathcal{G}$ from an initial state $s_{\mathsf{init}} \in S$ is an infinite sequence of states $\pi = s_0 s_1 s_2 \dots$ such that $s_0 = s_{\mathsf{init}}$ and $(s_i, s_{i+1}) \in E$ for all $i \geq 0$. The *prefix* up to the n-th state of π is the finite sequence $\pi(n) = s_0 s_1 \dots s_n$. We denote its last state by $\mathsf{Last}(\pi(n)) = s_n$. The set of plays of $\mathcal{G}$ is denoted by $\mathsf{Plays}(\mathcal{G})$ and the corresponding set of prefixes is denoted by $\mathsf{Prefs}(\mathcal{G})$. Given a play $\pi \in \mathsf{Plays}(\mathcal{G})$, we denote by $\mathsf{Inf}(\pi) \subseteq S$ the set of states that are visited infinitely often along the play.

Given a function $f\colon \mathsf{Plays}(\mathcal{G}) \to \mathbb{R} \cup \{-\infty, \infty\}$, the *value* of a play π is $f(\pi)$. The *mean-payoff* of a prefix $\rho = s_0 s_1 \dots s_n$ is $\mathsf{MP}(\rho) = \frac{1}{n} \sum_{i=0}^{i=n-1} w((s_i, s_{i+1}))$. For plays, $\mathsf{MP}(\pi) = \liminf_{n\to\infty} \mathsf{MP}(\pi(n))$. Given a graph with strictly positive weights ($w\colon E \to \mathbb{N}_0$) and a target set $T \subseteq S$, the *truncated sum up to* T is $\mathsf{TS}_T\colon \mathsf{Plays}(\mathcal{G}) \to \mathbb{N} \cup \{\infty\}$, $\mathsf{TS}_T(\pi = s_0 s_1 s_2 \dots) = \sum_{i=0}^{n-1} w((s_i, s_{i+1}))$, with n the first index such that $s_n \in T$, and $\mathsf{TS}_T(\pi) = \infty$ if π never reaches any state in T.

Probability distributions. Given a finite set A, a (rational) *probability distribution* on A is a function $p\colon A \to [0, 1] \cap \mathbb{Q}$ such that $\sum_{a\in A} p(a) = 1$. We denote the set of probability distributions on A by $\mathcal{D}(A)$. The *support* of the probability distribution p on A is $\mathsf{Supp}(p) = \{a \in A \mid p(a) > 0\}$.

Two-player games. We consider two-player turn-based games and denote the two *players* by $\mathcal{P}_1$ and $\mathcal{P}_2$. A finite *two-player game* is a tuple $G = (\mathcal{G}, S_1, S_2)$ composed of (i) a finite weighted graph $\mathcal{G} = (S, E, w)$; and (ii) a partition of its states S into S_1 and S_2 that resp. denote the sets of states belonging to $\mathcal{P}_1$ and $\mathcal{P}_2$. A prefix $\pi(n)$ of a play π belongs to $\mathcal{P}_i$, $i \in \{1, 2\}$, if $\mathsf{Last}(\pi(n)) \in S_i$. The set of prefixes that belong to $\mathcal{P}_i$ is denoted by $\mathsf{Prefs}_i(G)$.

We sometimes denote by $|G|$ the size of a game, defined as a polynomial function of $|S|$, $|E|$ and $V = \lceil \log_2 W \rceil$.

Strategies. A *strategy* for $\mathcal{P}_i$, $i \in \{1, 2\}$, is a function $\lambda_i \colon \mathsf{Prefs}_i(G) \to \mathcal{D}(S)$ such that for all $\rho \in \mathsf{Prefs}_i(G)$, we have $\mathsf{Supp}(\lambda_i(\rho)) \subseteq \mathsf{Succ}(\mathsf{Last}(\rho))$. A strategy is *pure* if its support is a singleton for all prefixes. A strategy λ_i for $\mathcal{P}_i$ has *finite memory* if it can be encoded by a stochastic finite state machine with outputs, called *stochastic Moore machine*, $\mathcal{M}(\lambda_i) = (\mathsf{Mem}, \mathsf{m}_0, \alpha_\mathsf{u}, \alpha_\mathsf{n})$, where (i) Mem is a finite set of memory elements, (ii) $\mathsf{m}_0 \in \mathsf{Mem}$ is the initial memory element, (iii) $\alpha_\mathsf{u} \colon \mathsf{Mem} \times S \to \mathsf{Mem}$ is the update function, and (iv) $\alpha_\mathsf{n} \colon \mathsf{Mem} \times S_i \to \mathcal{D}(S)$ is the next-action function. If the game is in $s \in S_i$ and $\mathsf{m} \in \mathsf{Mem}$ is the current memory, then the strategy chooses s', the next state of the game, according to the distribution $\alpha_\mathsf{n}(\mathsf{m}, s)$. When the game leaves a state $s \in S$, the memory is updated to $\alpha_\mathsf{u}(\mathsf{m}, s)$. Pure strategies have deterministic next-action functions. A strategy is *memoryless* if $|\mathsf{Mem}| = 1$, i.e., it only depends on the current state of the game.

We resp. denote by $\Lambda_i(G)$ and $\Lambda_i^F(G)$ the sets of general (i.e., possibly randomized and infinite-memory) and finite-memory strategies for player $\mathcal{P}_i$ on the game G. We do not write G in this notation when the context is clear. A play π is said to be *consistent* with a strategy $\lambda_i \in \Lambda_i$ if for all $n \geq 0$ such that $\mathsf{Last}(\pi(n)) \in S_i$, we have $\mathsf{Last}(\pi(n+1)) \in \mathsf{Supp}(\lambda_i(\pi(n)))$.

Markov decisions processes. A finite *Markov decision process* (MDP) is a tuple $\mathrm{P} = (\mathcal{G}, S_1, S_\Delta, \Delta)$ where (i) $\mathcal{G} = (S, E, w)$ is a finite weighted graph, (ii) S_1 and S_Δ define a partition of the set of states S into states of $\mathcal{P}_1$ and *stochastic states*, and (iii) $\Delta \colon S_\Delta \to \mathcal{D}(S)$ is the transition function that, given a stochastic state $s \in S_\Delta$, defines the probability distribution $\Delta(s)$ over the possible successors of s, such that for all states $s \in S_\Delta$, $\mathsf{Supp}(\Delta(s)) \subseteq \mathsf{Succ}(s)$. In contrast to some other classical definitions of MDPs in the literature, we explicitly allow that, for some states $s \in S_\Delta$, $\mathsf{Supp}(\Delta(s)) \subsetneq \mathsf{Succ}(s)$: some edges of the graph $\mathcal{G}$ are assigned probability zero by the transition function. We define the subset of edges $E_\Delta = \{(s_1, s_2) \in E \mid s_1 \in S_\Delta \ Rightarrow s_2 \in \mathsf{Supp}(\Delta(s_1))\}$, representing all edges that either start in a state of $\mathcal{P}_1$, or are chosen with non-zero probability by the transition function Δ. The notions of prefixes belonging to $\mathcal{P}_1$ and of strategies for $\mathcal{P}_1$ are naturally extended to MDPs.

End-components. We define *end-components* (ECs) of an MDP as subgraphs in which $\mathcal{P}_1$ can ensure to stay despite stochastic states [11]. Let $P = (\mathcal{G}, S_1, S_\Delta, \Delta)$ be an MDP, with $\mathcal{G} = (S, E, w)$ its underlying graph. An EC in P is a set $U \subseteq S$ such that (i) the subgraph $(U, E_\Delta \cap (U \times U))$ is strongly connected, with E_Δ defined as before, i.e., stochastic edges with probability zero are treated as non-existent; and (ii) for all $s \in U \cap S_\Delta$, $\mathsf{Supp}(\Delta(s)) \subseteq U$, i.e., in stochastic states, all outgoing edges either stay in U or belong to $E \setminus E_\Delta$ (the probability of leaving U from a state $s \in S_\Delta$ is zero).

Markov chains. A finite *Markov chain* (MC) is a tuple $M = (\mathcal{G}, \delta)$ where (i) $\mathcal{G} = (S, E, w)$ is a finite weighted graph; and (ii) $\delta \colon S \to \mathcal{D}(S)$ is the transition function that, given $s \in S$, defines the distribution $\delta(s)$, such that for all $s \in S$, $\mathsf{Supp}(\delta(s)) \subseteq \mathsf{Succ}(s)$. In an MC, an *event* is a measurable set of plays $\mathcal{A} \subseteq \mathsf{Plays}(\mathcal{G})$. Every event has a uniquely defined probability [26] (Carathéodory's extension theorem induces a unique probability measure on the Borel σ-algebra over $\mathsf{Plays}(\mathcal{G})$). We denote by $\mathbb{P}^M_{s_\mathsf{init}}(\mathcal{A})$ the probability that a play belongs to $\mathcal{A}$ when the MC M starts in $s_\mathsf{init} \in S$ and is executed for an infinite number of

steps. Given a measurable function $f\colon \mathsf{Plays}(\mathcal{G}) \to \mathbb{R} \cup \{-\infty, \infty\}$, we denote by $expect^{M}_{s_{\mathsf{init}}}(f)$ the *expected value* or *expectation* of f over a play starting in s_{init}.

Outcomes. Let $M = (\mathcal{G}, \delta)$ be a Markov chain, with $\mathcal{G} = (S, E, w)$ its underlying graph. Given an initial state $s_{\mathsf{init}} \in S$, we define the set of its possible *outcomes* as

$$\mathsf{Outs}_M(s_{\mathsf{init}}) = \{\pi = s_0 s_1 s_2 \ldots \in \mathsf{Plays}(\mathcal{G}) \mid s_0 = s_{\mathsf{init}} \wedge \forall\, n \in \mathbb{N},\ s_{n+1} \in \mathsf{Supp}(\delta(s_n))\}\,.$$

Let $G = (\mathcal{G}, S_1, S_2)$ be a two-player game, with $\mathcal{G} = (S, E, w)$ its graph. Given two strategies, $\lambda_1 \in \Lambda_1$ and $\lambda_2 \in \Lambda_2$, and an initial state $s_{\mathsf{init}} \in S$, we extend the notion of outcomes as follows:

$$\mathsf{Outs}_G(s_{\mathsf{init}}, \lambda_1, \lambda_2) = \{\pi = s_0 s_1 s_2 \ldots \in \mathsf{Plays}(\mathcal{G}) \mid s_0 = s_{\mathsf{init}} \wedge \pi \text{ is consistent with } \lambda_1 \text{ and } \lambda_2\}\,.$$

When fixing the strategies, we obtain an MC denoted by $G[\lambda_1, \lambda_2]$. This MC is finite if both λ_1 and λ_2 are finite-memory strategies. The outcomes of G and $G[\lambda_1, \lambda_2]$ are not *sensu stricto* of the same nature as the graph of the MC is obtained through the product of the memory elements of the strategies given as Moore machines and the states of the game. Still, there exists a bijection between outcomes of the MC and their *traces* in the initial game, thanks to the projection operator on S. For the sake of readability, we equivalently refer to outcomes and their traces.

Let $P = (\mathcal{G}, S_1, S_\Delta, \Delta)$ be an MDP, with $\mathcal{G} = (S, E, w)$ its graph. Again, we can fix the strategy λ_1 of $\mathcal{P}_1$ and obtain the MC $P[\lambda_1]$. Its set of outcomes starting in $s_{\mathsf{init}} \in S$ is denoted $\mathsf{Outs}_P(s_{\mathsf{init}}, \lambda_1)$. Finally, back to the two-player game G, if we fix the strategy λ_i of only one player $\mathcal{P}_i$, $i \in \{1, 2\}$, we obtain not an MC, but an MDP for the remaining player $\mathcal{P}_{3-i}$. This MDP is denoted by $G[\lambda_i]$.

Subgraphs and subgames. Given a graph $\mathcal{G} = (S, E, w)$ and a subset $A \subseteq S$, we define the induced subgraph $\mathcal{G} \restriction A = (A, E \cap (A \times A), w)$ naturally. Subgames are defined similarly by considering their induced subgraphs: they are only properly defined if the induced subgraphs contain no deadlock.

Worst-case synthesis. Given a game $G = (\mathcal{G}, S_1, S_2)$, with $\mathcal{G} = (S, E, w)$, an initial state $s_{\mathsf{init}} \in S$, a function $f\colon \mathsf{Plays}(\mathcal{G}) \to \mathbb{R} \cup \{-\infty, \infty\}$, and a threshold $\mu \in \mathbb{Q}$, the *worst-case threshold problem* asks to decide if $\mathcal{P}_1$ has a strategy $\lambda_1 \in \Lambda_1$ such that $\forall\, \lambda_2 \in \Lambda_2, \forall\, \pi \in \mathsf{Outs}_G(s_{\mathsf{init}}, \lambda_1, \lambda_2)$, $f(\pi) \geq \mu$. For the mean-payoff, pure memoryless optimal[1] strategies exist for both players [22, 14]. Hence, deciding the winner is in NP $\cap$ coNP, and it was furthermore shown to be in UP $\cap$ coUP [28, 21, 18]. Whether the problem is in P is a long-standing open problem [3, 7]. For the shortest path (truncated sum value function), it can be shown that the decision problem takes polynomial time, as a winning strategy of $\mathcal{P}_1$ should avoid all cycles (because they yield strictly positive costs), hence usage of attractors and comparison of the worst possible sum of costs with the threshold suffices.

[1] A strategy for $\mathcal{P}_i$, $i \in \{1, 2\}$, is said to be *optimal* if it ensures a threshold higher or equal to the threshold ensured by any other strategy of the same player. The threshold ensured by an optimal strategy is called the *optimal value*.

Expected value synthesis. Given an MDP $P = (\mathcal{G}, S_1, S_\Delta, \Delta)$, with $\mathcal{G} = (S, E, w)$, an initial state $s_{\mathsf{init}} \in S$, a measurable function $f\colon \mathsf{Plays}(\mathcal{G}) \to \mathbb{R} \cup \{-\infty, \infty\}$, and a threshold $\nu \in \mathbb{Q}$, the *expected value threshold problem* asks to decide if $\mathcal{P}_1$ has a strategy $\lambda_1 \in \Lambda_1$ such that $\mathbb{E}_{s_{\mathsf{init}}}^{P[\lambda_1]}(f) \geq \nu$. Optimal expected mean-payoff in MDPs can be achieved by memoryless strategies, and the corresponding decision problem can be solved in polynomial time through linear programming [15]. The truncated sum value function has been studied in the literature under the name of *shortest path problem*: again, memoryless strategies suffice to be optimal and the problem is solvable in polynomial time [1, 12].

Beyond worst-case synthesis. We study the synthesis of finite-memory strategies that ensure, *simultaneously*, a value greater than a threshold μ in the worst-case (i.e., against any strategy of the adversary), and an expected value greater than a threshold ν against a given finite-memory stochastic model of the adversary (e.g., representing commonly observed behavior of the environment).

▶ **Definition 1.** Given a game $G = (\mathcal{G}, S_1, S_2)$, with $\mathcal{G} = (S, E, w)$, an initial state $s_{\mathsf{init}} \in S$, a finite-memory stochastic model $\lambda_2^{\mathsf{stoch}} \in \Lambda_2^F$ of the adversary, represented by a stochastic Moore machine, a measurable value function $f\colon \mathsf{Plays}(\mathcal{G}) \to \mathbb{R} \cup \{-\infty, \infty\}$, and two thresholds $\mu, \nu \in \mathbb{Q}$, the *beyond worst-case (BWC) problem* asks to decide if $\mathcal{P}_1$ has a finite-memory strategy $\lambda_1 \in \Lambda_1^F$ such that

$$\begin{cases} \forall\, \lambda_2 \in \Lambda_2,\ \forall\, \pi \in \mathsf{Outs}_G(s_{\mathsf{init}}, \lambda_1, \lambda_2),\ f(\pi) > \mu & (1) \\ \mathbb{E}_{s_{\mathsf{init}}}^{G[\lambda_1, \lambda_2^{\mathsf{stoch}}]}(f) > \nu & (2) \end{cases}$$

and the BWC synthesis problem asks to synthesize such a strategy if one exists.

We take the convention to ask for values strictly greater than the thresholds to ease the formulation of our results in the following. Indeed, for some thresholds, it is possible to synthesize strategies that ensure ε-close values, for any $\varepsilon > 0$, while it is not feasible to achieve the exact threshold. Notice that we can assume $\nu > \mu$, otherwise the problem reduces to the classical worst-case analysis.

3 Mean-Payoff Value Function

We present algorithm BWC_MP (Alg. 1) for the BWC synthesis problem and we highlight its cornerstones. Results on memory requirements follow. A sample game is presented in Fig. 2.

Inputs and outputs. The algorithm takes as input: a game G^i, a finite-memory stochastic model of the adversary λ_2^i, a worst-case threshold μ^i, an expected value threshold ν^i, and an initial state s_{init}^i. Its output is Yes if and only if there exists a finite-memory strategy of $\mathcal{P}_1$ satisfying the BWC problem. We present how to synthesize such a satisfying strategy in the following.

Preprocessing. The first part of the algorithm (lines 1-7) is the preprocessing of the game G^i and the stochatic model λ_2^i given as inputs in order to apply the second part of the algorithm (lines 8–11) on a modified game G and stochastic model $\lambda_2^{\mathsf{stoch}}$, simpler to manipulate. We ensure that the answer to the BWC problem on the modified game is Yes if and only if it is also Yes on the input game, and that winning strategies of $\mathcal{P}_1$ in G can be transferred to winning strategies in G^i.

Algorithm 1 BWC_MP($G^i, \lambda_2^i, \mu^i, \nu^i, s_{\mathsf{init}}^i$)

Require: $G^i = \left(\mathcal{G}^i, S_1^i, S_2^i\right)$ a game, $\mathcal{G}^i = \left(S^i, E^i, w^i\right)$ its underlying graph, $\lambda_2^i \in \Lambda_2^F(G^i)$ a finite-memory stochastic model of the adversary, $\mathcal{M}(\lambda_2^i) = (\mathsf{Mem}, \mathsf{m}_0, \alpha_{\mathsf{u}}, \alpha_{\mathsf{n}})$ its Moore machine, $\mu^i = \frac{a}{b}, \nu^i \in \mathbb{Q}$, $\mu^i < \nu^i$, resp. the worst-case and the expected value thresholds, and $s_{\mathsf{init}}^i \in S^i$ the initial state

Ensure: The answer is YES if and only if $\mathcal{P}_1$ has a finite-memory strategy $\lambda_1 \in \Lambda_1^F(G^i)$ satisfying the BWC problem from s_{init}^i, for the thresholds pair (μ^i, ν^i) and the mean-payoff value function

{*Preprocessing*}

1: **if** $\mu^i \neq 0$ **then** define $\forall\, e \in E^i$, $w_{\mathsf{new}}^i(e) := b \cdot w^i(e) - a$, and consider thresholds $(0, \nu := b \cdot \nu^i - a)$

2: Compute $S_{WC} := \left\{ s \in S^i \mid \exists\, \lambda_1 \in \Lambda_1(G^i), \forall\, \lambda_2 \in \Lambda_2(G^i), \forall\, \pi \in \mathsf{Outs}_{G^i}(s, \lambda_1, \lambda_2), \mathsf{MP}(\pi) > 0 \right\}$

3: **if** $s_{\mathsf{init}}^i \notin S_{WC}$ **then return** NO **else**

4: Let $G^w := G^i \restriction S_{WC}$ be the subgame induced by worst-case winning states

5: Build $G := G^w \otimes \mathcal{M}(\lambda_2^i) = (\mathcal{G}, S_1, S_2)$, $\mathcal{G} = (S, E, w)$, $S \subseteq (S_{WC} \times \mathsf{Mem})$, the game obtained by product with the Moore machine, and $s_{\mathsf{init}} := (s_{\mathsf{init}}^i, \mathsf{m}_0)$ the corresponding initial state

6: Let $\lambda_2^{\mathsf{stoch}} \in \Lambda_2^M(G)$ be the memoryless transcription of λ_2^i on G

7: Let $P := G[\lambda_2^{\mathsf{stoch}}] = (\mathcal{G}, S_1, S_\Delta = S_2, \Delta = \lambda_2^{\mathsf{stoch}})$ be the MDP obtained from G and $\lambda_2^{\mathsf{stoch}}$

{*Main algorithm*}

8: Compute $\mathcal{U}_{\mathrm{w}}$ the set of maximal winning end-components of P

9: Build $P' = (\mathcal{G}', S_1, S_\Delta, \Delta)$, where $\mathcal{G}' = (S, E, w')$ and w' is defined s.t. $\forall\, e = (s_1, s_2) \in E$, $w'(e) := w(e)$ if $\exists\, U \in \mathcal{U}_{\mathrm{w}}$ s.t. $\{s_1, s_2\} \subseteq U$, or $w'(e) := 0$ otherwise

10: Compute the maximal expected value ν^* from s_{init} in P'

11: **if** $\nu^* > \nu$ **then return** YES **else return** NO

First, we modify the weights of $\mathcal{G}^i$ in order to consider the equivalent BWC problem with thresholds $(0, \nu)$. This classical trick is used to get rid of explicitely considering the worst-case threshold in the following, as it is equal to zero. Second, observe that any strategy that is winning for the BWC problem must also be winning for the classical worst-case problem. Such a strategy cannot allow visits of any state from which $\mathcal{P}_1$ cannot ensure winning against an antagonistic adversary: entering such a state would be losing no matter the prefix. Indeed, mean-payoff is prefix-independent: for all $\rho \in \mathsf{Prefs}(\mathcal{G})$, $\pi \in \mathsf{Plays}(\mathcal{G})$ we have that $\mathsf{MP}(\rho \cdot \pi) = \mathsf{MP}(\pi)$. Hence, we reduce our study to G^w, the subgame induced by worst-case winning states in G^i (lines 2 and 4). Obviously, if from the initial state s_{init}^i, $\mathcal{P}_1$ cannot win the worst-case problem, then the answer to the BWC problem is NO (lines 3). Third, we build the game G which states are defined by the product of the states of G^w and the memory elements of $\mathcal{M}(\lambda_2^i)$ (line 5). Intuitively, we expand the initial game by integrating the memory of the stochastic model of $\mathcal{P}_2$ in the graph. This does not modify the power of the adversary. Fourth, the finite-memory stochastic model λ_2^i on G^i clearly translates to a memoryless stochastic model $\lambda_2^{\mathsf{stoch}}$ on G (line 6). This helps us obtain elegant proofs for the second part of the algorithm.

Analysis of end-components. The second part of the algorithm (lines 8-11) operates on a game G such that from all states, $\mathcal{P}_1$ has a strategy to achieve a strictly positive mean-payoff (recall $\mu = 0$). We consider the MDP $P = G[\lambda_2^{\mathsf{stoch}}]$ and notice that the underlying graphs of G and P are the same thanks to $\lambda_2^{\mathsf{stoch}}$ being memoryless. The next steps rely on the analysis of *end-components* in the MDP, i.e., strongly connected subgraphs in which $\mathcal{P}_1$ can ensure to stay when playing against the stochastic adversary. The motivation to this analysis arises from the following well-known result.

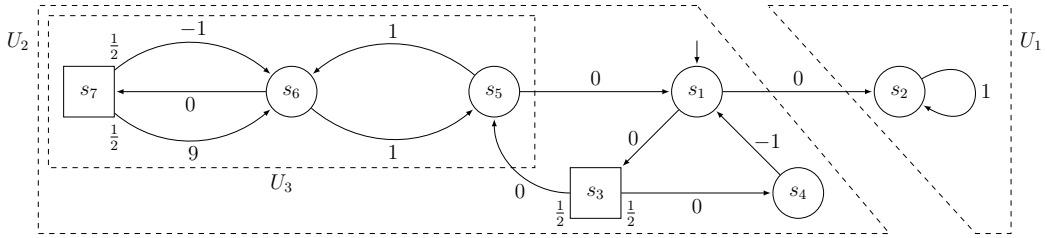


Figure 2 End component U_2 is losing. The set of maximal winning ECs is $\mathcal{U}_{\mathrm{w}} = \{U_1, U_3\}$. The set of winning ECs is $\mathcal{W} = \mathcal{U}_{\mathrm{w}} \cup \{\{s_5, s_6\}, \{s_6, s_7\}\}$.

▶ **Lemma 2** ([10, 11]). *Let $P = (\mathcal{G}, S_1, S_\Delta, \Delta)$ be an MDP, $\mathcal{G} = (S, E, w)$ its underlying graph, $\mathcal{E} \subseteq 2^S$ the set of its ECs, $s_{\mathsf{init}} \in S$ the initial state, and $\lambda_1 \in \Lambda_1(P)$ an arbitrary strategy of $\mathcal{P}_1$. Then,*

$$\mathbb{P}_{s_{\mathsf{init}}}^{P[\lambda_1]}\left(\{\pi \in \mathsf{Outs}_{P[\lambda_1]}(s_{\mathsf{init}}) \mid \mathsf{Inf}(\pi) \in \mathcal{E}\}\right) = 1.$$

Recall that the mean-payoff is prefix-independent, therefore the value of any outcome only depends on the states that are seen infinitely often. Hence, the expected mean-payoff in $P[\lambda_1]$ depends *uniquely* on the value obtained in the ECs. Inside an EC, we can compute the maximal expected value that can be achieved by $\mathcal{P}_1$, and this value is the same in all states of the EC [15].

To satisfy the expected value requirement (eq. (2)), an acceptable strategy has to favor reaching ECs with a sufficient expectation, but under the constraint that it also ensures the worst-case requirement (eq. (1)): some ECs with high expected values may still need to be avoided because they do not permit to guarantee this constraint. This is the cornerstone of the classification of ECs that follows.

Classification of end-components. Let $\mathcal{E} \subseteq 2^S$ be the set of all ECs in P. By definition, only edges in E_Δ, as defined in Sect. 2, are involved to determine which sets of states form an EC in P. For any EC $U \in \mathcal{E}$, there may exist edges from $E \setminus E_\Delta$ starting in U, such that $\mathcal{P}_2$ can force leaving U when using an arbitrary strategy. Still these edges will never be used by the stochastic model $\lambda_2^{\mathsf{stoch}}$. This remark is important to the definition of strategies of $\mathcal{P}_1$ that guarantee the worst-case requirement, as $\mathcal{P}_1$ needs to be able to react to the hypothetic use of such an edge. It is also the case *inside* an EC.

Now, we want to consider the ECs in which $\mathcal{P}_1$ can ensure that the worst-case requirement will be fulfilled (without having to leave the EC): we call them *winning*. The others need to be eventually avoided, hence have zero impact on the expectation of a finite-memory strategy satisfying the BWC problem. So we call the latter *losing*. Formally, let $U \in \mathcal{E}$ be an EC. It is *winning* if, in the subgame $G \restriction U$, from all states, $\mathcal{P}_1$ has a strategy to ensure a strictly positive mean-payoff against any strategy of $\mathcal{P}_2$ *that only chooses edges which are assigned non-zero probability by* $\lambda_2^{\mathsf{stoch}}$, or equivalently, edges in E_Δ. We denote $\mathcal{W} \subseteq \mathcal{E}$ the set of such ECs. Non-winning ECs are *losing*: in those, whatever the strategy of $\mathcal{P}_1$ played against the stochastic model $\lambda_2^{\mathsf{stoch}}$ (or any strategy with the same support), there exists at least one outcome for which the mean-payoff is not strictly positive (even if its probability is zero, its mere existence is not acceptable for the worst-case requirement).

Maximal winning end-components. Based on these definitions, observe that line 8 of algorithm BWC_MP does not actually compute the set $\mathcal{W}$ containing all winning ECs, but

the set $\mathcal{U}_{\mathrm{w}} \subseteq \mathcal{W}$, defined as $\mathcal{U}_{\mathrm{w}} = \{U \in \mathcal{W} \mid \forall U' \in \mathcal{W}, U \subseteq U' \Rightarrow U = U'\}$, i.e., the set of *maximal* winning ECs.

The intuition on *why we can* restrict to this subset is as follows. If an EC $U_1 \in \mathcal{W}$ is included in another EC $U_2 \in \mathcal{W}$, then the maximal expected value achievable in U_2 is at least equal to the one achievable in U_1. Indeed, $\mathcal{P}_1$ can reach U_1 with probability one (by virtue of U_2 being an EC and $U_1 \subseteq U_2$) and stay in it with probability one (by virtue of U_1 being an EC): the expectation is equal to what can be obtained in U_1 thanks to the prefix-independence. Hence it is sufficient to consider maximal winning ECs in our computations.

As for *why we do it*, the complexity gain is critical. The number of winning ECs can be exponential in the size of the input, as $|\mathcal{W}| \leq |\mathcal{E}| \leq 2^{|S|}$. Yet, the number of maximal ones is bounded by $|\mathcal{U}_{\mathrm{w}}| \leq |S|$ as they are disjoint by definition: for any two winning ECs with a non-empty intersection, their union is also an EC, and is still winning because $\mathcal{P}_1$ can essentially stick to the EC of his choice.

▶ **Lemma 3.** *The set $\mathcal{U}_{\mathrm{w}}$ of maximal winning ECs can be computed in NP ∩ coNP.*

Roughly sketched, our recursive subalgorithm computes the maximal EC decomposition of an MDP (in polynomial time [8]), then checks for each EC U in the decomposition (their number is polynomial) if U is winning or not, which requires a call to an NP ∩ coNP oracle solving the worst-case threshold problem on the corresponding subgame. If U is losing, it may still be the case that a sub-EC $U' \subsetneq U$ is winning. We recurse on the MDP reduced to U, where states from which $\mathcal{P}_2$ can win in U have been removed: the stack of calls is at most polynomial.

Ensure reaching winning end-components. We now refine Lemma 2 for *finite-memory* strategies that *satisfy* the BWC problem.

▶ **Lemma 4.** *Let $G = (\mathcal{G}, S_1, S_2)$ be a two-player game, $\lambda_2^{\mathsf{stoch}} \in \Lambda_2^M$ a memoryless stochastic model of $\mathcal{P}_2$, $P = G[\lambda_2^{\mathsf{stoch}}]$ the resulting MDP and $s_{\mathsf{init}} \in S$ the initial state. Let $\lambda_1^f \in \Lambda_1^F$ be a finite-memory strategy of $\mathcal{P}_1$ that satisfies the BWC problem for thresholds $(0, \nu) \in \mathbb{Q}^2$. Then, we have that*

$$\mathbb{P}_{s_{\mathsf{init}}}^{P[\lambda_1^f]}\left(\left\{\pi \in \mathsf{Outs}_{P[\lambda_1^f]}(s_{\mathsf{init}}) \mid \mathsf{Inf}(\pi) \in \mathcal{W}\right\}\right) = 1.$$

Equivalently, the probability that $\mathsf{Inf}(\pi) = U$ for some $U \in \mathcal{E} \setminus \mathcal{W}$ is zero. The equality is crucial. It may be the case, with non-zero probability, that $\mathsf{Inf}(\pi) = U' \subsetneq U$ for some $U' \in \mathcal{W}$ and $U \in \mathcal{E} \setminus \mathcal{W}$ (hence the recursive algorithm to compute $\mathcal{U}_{\mathrm{w}}$). It is clear that $\mathcal{P}_1$ should not visit all the states of a losing EC forever, as then he would not be able to guarantee the worst-case threshold.

Our goal is to build an MDP P', sharing the same graph and ECs as P, such that an optimal strategy for the expectation problem on P' will naturally avoid losing ECs and prescribe which winning ECs are the most interesting to reach for a BWC strategy on the initial game G and MDP P. The expected value obtained in P by any BWC satisfying strategy of $\mathcal{P}_1$ only depends on the weights of edges involved in winning ECs, or equivalently, in maximal winning ECs (as the set of outcomes that are not trapped in them has measure zero). We build P' by modifying the weights of P (line 9): we keep them unchanged in edges that belong to some $U \in \mathcal{U}_{\mathrm{w}}$, and we put them to zero everywhere else, which is lower than the expectation granted by winning ECs (strictly positive by definition).

Reach the highest valued winning end-components. We compute the maximal expected value ν^* that can be achieved by $\mathcal{P}_1$ in the MDP P', from the initial state (line 10). It takes polynomial time and memoryless strategies suffice to achieve the maximal value [15]. Basically, we build a strategy that favors reaching ECs with high associated expectations in P'. We argue that the ECs reached with probability one by this strategy are necessarily winning ECs. Clearly, if a winning EC is reachable instead of a losing one, it will be favored because of the weights definition in P' (expectation is strictly higher in winning ECs). It remains to check if winning ECs are reachable with probability one from any state in S. They are, due to the preprocessing. Indeed, all states are winning for the worst-case requirement. Clearly, from any state in $A = S \setminus \bigcup_{U \in\ ecsSet} U$, $\mathcal{P}_1$ cannot ensure to stay in A (otherwise it would form an EC) and must be able to win the worst-case from reached ECs. Now for any state in $B = \bigcup_{U \in \mathcal{E}} U \setminus \bigcup_{U \in \mathcal{U}_\mathrm{w}} U$, i.e., states in losing ECs and not in any sub-EC winning, $\mathcal{P}_1$ cannot win the worst-case by staying in B, by definition of losing EC. Since $\mathcal{P}_1$ can ensure the worst-case by hypothesis, he must be able to reach $C = \bigcup_{U \in \mathcal{U}_\mathrm{w}} U$ from any state in B, as claimed.

Inside winning end-components. Based on that, winning ECs are reached with probability one. Consider what we can say about such ECs assuming that $E_\Delta = E$, i.e., if all possible edges are mapped to non-zero probabilities. We establish a finite-memory *combined strategy* of $\mathcal{P}_1$ that ensures (i) worst-case satisfaction while yielding (ii) an expected value ε-close to the maximal expectation inside the component. For two well-chosen parameters $K, L \in \mathbb{N}$, it is informally defined as follows: in phase *(a)*, play a memoryless expected value optimal strategy for K steps and memorize $\mathsf{Sum} \in \mathbb{Z}$, the sum of weights along these steps; in phase *(b)*, if $\mathsf{Sum} > 0$, go to *(a)*, otherwise play a memoryless worst-case optimal strategy for L steps, then go to *(a)*. In phases *(a)*, $\mathcal{P}_1$ tries to increase its expectation and approach its optimal one, while in phase *(b)*, he compensates, if needed, losses that occured in phase *(a)*. The two memoryless strategies exist on the subgame induced by the EC: by definition of ECs, based on E_Δ, the stochastic model of $\mathcal{P}_2$ will never be able to force leaving the EC against the combined strategy. A key result of our paper is the existence of values for K and L such that (i) and (ii) are verified, as stated in the next theorem.

▶ **Theorem 5.** *Inside a WEC with $\nu^* \in \mathbb{Q}$ the maximal expectation achievable by $\mathcal{P}_1$, for all $\varepsilon > 0$, there exists a finite-memory strategy of $\mathcal{P}_1$ that satisfies the BWC problem for thresholds* $(0, \nu^* - \varepsilon)$.

We see plays as sequences of periods, each starting with phase *(a)*. First, for any K, we can define $L(K)$ such that any period composed of phases *(a)* + *(b)* ensures a mean-payoff at least $1/(K + L) > 0$. Periods containing only phase *(a)* trivially induce a mean-payoff at least $1/K$. Both rely on the weights being integers. As the length of any period is bounded, the inequality remains strict for the mean-payoff of any play, granting (i). Now, consider parameter K. Clearly, when $K \to \infty$, the expectation over a phase *(a)* tends to the optimal one. Nevertheless, phases *(b)* also contribute to the overall expectation of the combined strategy, and (in general) lower it so that it is strictly less than the optimal for any $K, L \in \mathbb{N}$. Hence to prove (ii), we not only need that the probability of playing phase *(b)* decreases when K increases, but also that it decreases faster than the increase of L, needed to ensure (i), so that overall, the contribution of phases *(b)* tends to zero when $K \to \infty$. This is indeed the case and can be proved using results bounding the probability of observing a mean-payoff significantly (more than some ε) different than the optimal expectation along a phase *(a)* of length $K \in \mathbb{N}$: this probability decreases exponentially when K increases [25, 19] (related to

the notions of Chernoff bounds and Hoeffding's inequality in MCs), while L only needs to be polynomial in K.

Now, consider what happens if $E_\Delta \subsetneq E$. If $\mathcal{P}_2$ uses an arbitrary strategy, he can take edges of probability zero, i.e., in $E \setminus E_\Delta$, either staying in the EC, or leaving it. In both cases, this must be taken into account in order to satisfy eq. (1) as it may involve dangerous weights (recall that zero-probability edges are not considered when an EC is classified as winning or not). Fortunately, if this were to occur, $\mathcal{P}_1$ could switch to a worst-case winning memoryless strategy, which exists in all states thanks to the preprocessing (line 4). This has no impact on the expectation as it occurs with probability zero against $\lambda_2^{\mathsf{stoch}}$. The strategy to follow in winning ECs adds this reaction procedure to the combined strategy: we call it the *witness-and-secure strategy*.

Global strategy synthesis. In summary, losing ECs should be avoided and will be by a strategy that optimizes the expectation on the MDP P'; in winning ECs, $\mathcal{P}_1$ can obtain the expectation of the EC (at some arbitrarily small ε close) *and* ensure the worst-case threshold. We finally compare the value ν^* with the threshold ν (line 11): (i) if $\nu^* > \nu$, there exists a finite-memory strategy satisfying the BWC problem, and (ii) if not, there does not exist such a strategy.

▶ **Lemma 6.** *Algorithm* BWC_MP *is correct and complete.*

To prove (i), we establish a finite-memory strategy in G, called *global strategy*, of $\mathcal{P}_1$ that ensures a strictly positive mean-payoff against any antagonistic adversary, and ensures an expected mean-payoff ε-close to ν^* (hence, strictly greater than ν) against the stochastic adversary modeled by $\lambda_2^{\mathsf{stoch}}$ (i.e., in P). The intuition is as follows. We play the memoryless optimal strategy of the MDP P' for a sufficiently long time, defined by a parameter $N \in \mathbb{N}$, in order to be with probability close to one in a winning EC (the convergence is exponential by results on absorption times in MCs [20]). Then, if inside a winning EC, we switch to the witness-and-secure strategy which ensures both thresholds. If not yet in a winning EC, we switch to a worst-case winning strategy in G, existing by hypothesis. Thus the mean-payoff of plays that do not reach winning ECs is strictly positive. Since in winning ECs we are ε-close to the maximal expected value of the EC, we conclude that it is possible to play the optimal expectation strategy of MDP P' for sufficiently long to obtain an overall expected value which is arbitrarily close to ν^*, and still guarantee the worst-case threshold in all outcomes. To prove (ii), it suffices to understand that only ECs have an impact on the expectation, and that losing ECs cannot be used forever without endangering the worst-case requirement. Given a winning strategy on G, we can build a corresponding winning strategy on G^i by reintegrating the memory elements of the Moore machine in the memory of the strategy of $\mathcal{P}_1$.

Complexity bounds. The input size depends on the sizes of the game and the Moore machine for the stochastic model, and the encodings of weights and thresholds. All computations require (deterministic) polynomial time except for external calls solving the worst-case threshold problem, which is in NP ∩ coNP [28, 21] and not known to be in P. Hence, the overall complexity is in NP ∩ coNP and may collapse to P if the worst-case problem were to be proved in P: the BWC framework for mean-payoff surprisingly provides additional modeling power without negative impact on the complexity class. We establish that the BWC problem is at least as difficult as the worst-case problem thanks to a polynomial time reduction from the latter to the former. Thus, membership to NP ∩ coNP can be seen as optimal regarding our current knowledge of the worst-case problem.

▶ **Theorem 7.** *The beyond worst-case problem for the mean-payoff value function is in NP ∩ coNP and at least as hard as mean-payoff games.*

Memory requirements. The global strategy suffices if satisfaction of the BWC problem is possible. All the involved strategies (global, witness-and-secure, combined) are alternations between pure memoryless strategies, based on parameters N, K and $L \in \mathbb{N}$, which only need to be polynomial in the size of the game and the stochastic model, and in the values, granting the upper bound of Thm. 8. This bound is tight as polynomial memory in the value of weights is needed in general. Consider a family of games, $(G(X))_{X\in\mathbb{N}_0}$, based on the subgame $G \restriction U_3$ in Fig. 2, but with weights $-X$ and $X+5$ instead of -1 and 9 respectively. When choosing $\mu = 0$ and $\nu \in]1, 5/4[$, the BWC problem is satisfiable and it cannot be achieved by the memoryless strategy that always chooses edge (s_6, s_5). It is thus mandatory to choose (s_6, s_7) infinitely often in order to win. Moreover, after some point, everytime this edge is chosen, a satisfying strategy must *eventually* counteract the potential negative weight $-X$ by taking edge (s_1, s_2) for $\lfloor X/2 \rfloor + 1$ times. Hence polynomial memory in W is needed.

▶ **Theorem 8.** *Memory of pseudo-polynomial size may be necessary and is always sufficient to satisfy the BWC problem for the mean-payoff: polynomial in the size of the game and the stochastic model, and polynomial in the weight and threshold values.*

4 Truncated Sum Value Function - Shortest Path Problem

Let us consider a game graph such that $w\colon E \to \mathbb{N}_0$ assigns *strictly positive* integer weights to all edges, and a target set $T \subseteq S$ that $\mathcal{P}_1$ wants to reach with a path of bounded value. In other words, we study the BWC problem for the *shortest path* [1, 12]. More precisely, given an initial state $s_{\mathsf{init}} \in S$, the goal of $\mathcal{P}_1$ is to ensure to reach T with a path of *truncated sum* strictly lower than $\mu \in \mathbb{N}$ against all possible behaviors of $\mathcal{P}_2$ while guaranteeing, at the same time, an expected cost to target strictly lower than $\nu \in \mathbb{Q}$ against the stochastic model of the adversary specified by the stochastic Moore machine $\mathcal{M}(\lambda_2^{\mathsf{stoch}})$. Regarding Def. 1, the inequalities are reversed. Hence we assume $\nu < \mu$.

A pseudo-polynomial time algorithm. First, we construct, from the original game G and the worst-case threshold μ, a new game G_μ such that there is a bijection between the strategies of $\mathcal{P}_1$ in G_μ and the strategies of $\mathcal{P}_1$ in the original game G that are winning for the worst-case requirement: we unfold the original graph $\mathcal{G}$, tracking the current value of the truncated sum *up to the worst-case threshold* μ, and integrating this value in the states of an expanded graph $\mathcal{G}'$. In the corresponding game G', we compute the set of states R from which $\mathcal{P}_1$ can reach the target set with cost lower than μ and we define the subgame $G_\mu = G' \restriction R$ such that any path in the graph of G_μ satisfies the worst-case requirement. Second, from G_μ and the stochastic Moore machine $\mathcal{M}(\lambda_2^{\mathsf{stoch}})$, we construct an MDP in which we search for a *playerOne* strategy that ensures reachability of T with an expected cost strictly lower than ν. If it exists, it is guaranteed that it will also satisfy the worst-case requirement against any strategy of $\mathcal{P}_2$ thanks to the bijection evoked earlier.

▶ **Theorem 9.** *The beyond worst-case problem for the shortest path can be solved in pseudo-polynomial time: polynomial in the size of the underlying game graph, the Moore machine for the stochastic model of the adversary and the encoding of the expected value threshold, and polynomial in the value of the worst-case threshold.*

Memory requirements. The construction of Thm. 9 yields an upper bound that is polynomial in the size of the game and the stochastic model, and in the value of the worst-case threshold. Indeed, the synthesized strategy is memoryless in the MDP P that is obtained by taking the product of the expanded game G_μ, such that $|G_\mu| \leq |G| \cdot (\mu + 1)$, with the Moore machine $\mathcal{M}(\lambda_2^{\mathsf{stoch}})$.

We exhibit a family of games (Fig. 3) for which winning requires memory linear in μ, proving that the pseudo-polynomial bound is tight. Let $\mu \in \{13 + k \cdot 4 \mid k \in \mathbb{N}\}$. From s_1, $\mathcal{P}_1$ can ensure reaching the target set $T = \{s_3\}$ at a guaranteed cost of $\lfloor \frac{\mu}{2} \rfloor$. Yet, in order to *minimize* the expected cost of reaching T, $\mathcal{P}_1$ should try to reach it via state s_2, as the cost will be diminished. Hence, $\mathcal{P}_1$ should play edge (s_1, s_2) repeatedly, up to the point where playing (s_1, s_3) becomes mandatory to preserve the worst-case requirement (i.e., when the running sum of weights becomes equal to $\lfloor \frac{\mu}{2} \rfloor$ as the total cost for the worst outcome will be $2 \cdot \lfloor \frac{\mu}{2} \rfloor < \mu$). To implement this strategy, $\mathcal{P}_1$ has to play (s_1, s_2) exactly $\lfloor \frac{\mu}{4} \rfloor$ times and then switch to (s_1, s_3). This requires memory linear in the value μ. The expected value threshold ν can be chosen sufficiently low so that $\mathcal{P}_1$ is compelled to use this optimal strategy to satisfy the BWC problem.

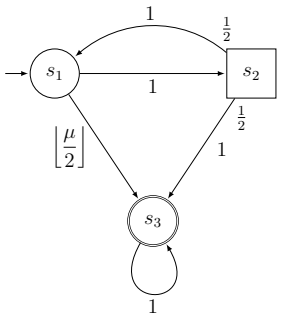


Figure 3 Family of games requiring linear memory in μ.

▶ **Theorem 10.** *Memory of pseudo-polynomial size may be necessary and is always sufficient to satisfy the BWC problem for the shortest path: polynomial in the size of the game and the stochastic model, and polynomial in the worst-case threshold value.*

NP-hardness of the decision problem. We establish that it is very unlikely that a truly-polynomial (i.e., also polynomial in the size of the encoding of the worst-case threshold) time algorithm exists, as the decision problem is NP-hard. Actually, it is likely that the problem is not in NP at all, since we prove a reduction from the K^{th} *largest subset problem* which is known to be NP-hard and commonly thought to be outside NP as natural certificates for the problem are larger than polynomial [17].

The K^{th} largest subset problem is as follows. Given a finite set A, a size function $h\colon A \to \mathbb{N}_0$ assigning strictly positive integer values to elements of A, and two naturals $K, L \in \mathbb{N}$, decide if there exist K distinct subsets $C_i \subseteq A$, $1 \leq i \leq K$, such that $h(C_i) = \sum_{a \in C_i} h(a) \leq L$ for all K subsets. The reduction is as follows. We build a game composed of two gadgets. The *random subset selection gadget* stochastically generates paths representing subsets of A, with the property that all subsets are equiprobable. The *choice gadget* follows. In it, $\mathcal{P}_1$ decides either to go to a state s_e, which leads to lower expectations but may be dangerous for the worst-case requirement, or to go to a state s_{wc}, always safe with regard to the worst-case but inducing an higher expected cost. The crux of the proof is to define values of the thresholds and the weights such that an optimal (i.e., minimizing the expectation while guaranteeing a given worst-case threshold) strategy for $\mathcal{P}_1$ consists in choosing s_e only when the generated subset $C \subseteq A$ satisfies $h(C) \leq L$, as asked by the K^{th} largest subset problem; and such that this strategy satisfies the BWC problem if and only if there exist K distinct subsets that verify this bound, i.e., if and only if the answer to the K^{th} largest subset problem is Yes.

▶ **Theorem 11.** *The beyond worst-case problem for the shortest path is NP-hard.*

References

1 D.P. Bertsekas and J.N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16:580–595, 1991.

2 T. Brázdil, K. Chatterjee, V. Forejt, and A. Kucera. Trading performance for stability in Markov decision processes. In *Proc. of LICS*, pages 331–340. IEEE Computer Society, 2013.

3 L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J.-F. Raskin. Faster algorithms for mean-payoff games. *Formal Methods in System Design*, 38(2):97–118, 2011.

4 V. Bruyère, E. Filiot, M. Randour, and J.-F. Raskin. Meet your expectations with guarantees: beyond worst-case synthesis in quantitative games. *CoRR*, abs/1309.5439, 2013. `http://arxiv.org/abs/1309.5439`.

5 K. Chatterjee and L. Doyen. Games and Markov decision processes with mean-payoff parity and energy parity objectives. In *Proc. of MEMICS*, LNCS. Springer, 2011.

6 K. Chatterjee, L. Doyen, T.A. Henzinger, and J.-F. Raskin. Generalized mean-payoff and energy games. In *Proc. of FSTTCS*, LIPIcs 8, pages 505–516. Schloss Dagstuhl - LZI, 2010.

7 K. Chatterjee, L. Doyen, M. Randour, and J.-F. Raskin. Looking at mean-payoff and total-payoff through windows. In *Proc. of ATVA*, LNCS 8172, pages 118–132. Springer, 2013.

8 K. Chatterjee and M. Henzinger. An $\mathcal{O}(n^2)$ time algorithm for alternating Büchi games. In *Proc. of SODA*, pages 1386–1399. SIAM, 2012.

9 K. Chatterjee, M. Randour, and J.-F. Raskin. Strategy synthesis for multi-dimensional quantitative objectives. In *Proc. of CONCUR*, LNCS 7454, pages 115–131. Springer, 2012.

10 C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.

11 L. de Alfaro. *Formal verification of probabilistic systems*. PhD thesis, Stanford University, 1997.

12 L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In *Proc. of CONCUR*, LNCS 1664, pages 66–81. Springer, 1999.

13 A. Degorre, L. Doyen, R. Gentilini, J.-F. Raskin, and S. Torunczyk. Energy and mean-payoff games with imperfect information. In *Proc. of CSL*, LNCS 6247, pages 260–274. Springer, 2010.

14 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *Int. Journal of Game Theory*, 8(2):109–113, 1979.

15 J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer, 1997.

16 J.A. Filar, D. Krass, and K.W. Ross. Percentile performance criteria for limiting average Markov decision processes. *Transactions on Automatic Control*, pages 2–10, 1995.

17 M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the Theory of NP-Completeness*. Freeman New York, 1979.

18 T. Gawlitza and H. Seidl. Games through nested fixpoints. In *Proc. of CAV*, LNCS 5643, pages 291–305. Springer, 2009.

19 P.W. Glynn and D. Ormoneit. Hoeffding's inequality for uniformly ergodic Markov chains. *Statistics & Probability Letters*, 56(2):143–146, 2002.

20 C.M. Grinstead and J.L. Snell. *Introduction to probability*. American Mathematical Society, 1997.

21 M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998.

22 T.M. Liggett and S.A. Lippman. Stochastic games with perfect information and time average payoff. *Siam Review*, 11(4):604–607, 1969.

23 S. Mannor and J.N. Tsitsiklis. Mean-variance optimization in Markov decision processes. In *Proc. of ICML*, pages 177–184. Omnipress, 2011.

24 M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

25 M. Tracol. Fast convergence to state-action frequency polytopes for MDPs. *Oper. Res. Lett.*, 37(2):123–126, 2009.

26 M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. of FOCS*, pages 327–338. IEEE Computer Society, 1985.

27 C. Wu and Y. Lin. Minimizing risk models in Markov decision processes with policies depending on target values. *Journal of Mathematical Analysis and Applications*, 231(1):47–67, 1999.

28 U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.

Chordal Editing is Fixed-Parameter Tractable*

Yixin Cao† **and Dániel Marx**

Institute for Computer Science & Control, Hungarian Academy of Sciences
yixin@sztaki.hu, dmarx@cs.bme.hu

Abstract

Graph modification problems are typically asked as follows: is there a set of k operations that transforms a given graph to have a certain property. The most commonly considered operations include vertex deletion, edge deletion, and edge addition; for the same property, one can define significantly different versions by allowing different operations. We study a very general graph modification problem which allows all three types of operations: given a graph G and integers k_1, k_2, and k_3, the CHORDAL EDITING problem asks if G can be transformed into a chordal graph by at most k_1 vertex deletions, k_2 edge deletions, and k_3 edge additions. Clearly, this problem generalizes both CHORDAL VERTEX/EDGE DELETION and CHORDAL COMPLETION (also known as MINIMUM FILL-IN). Our main result is an algorithm for CHORDAL EDITING in time $2^{O(k \log k)} \cdot n^{O(1)}$, where $k := k_1 + k_2 + k_3$; therefore, the problem is fixed-parameter tractable parameterized by the total number of allowed operations. Our algorithm is both more efficient and conceptually simpler than the previously known algorithm for the special case CHORDAL DELETION.

1998 ACM Subject Classification G.2.2 Graph algorithms

Keywords and phrases chordal graph, parameterized computation, graph modification problems, chordal deletion, chordal completion, clique tree decomposition, holes, simplicial vertex sets

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.214

1 Introduction

A graph is chordal if it contains no hole, that is, an induced cycle of at least four vertices. After more than half century of intensive investigation, the properties and the recognition of chordal graphs are well understood. Their natural structure earns them wide applications, some of which might not seem to be related to graphs at first sight. During the study of Gaussian elimination on sparse positive definite matrices, Rose [15, 16] formulated the CHORDAL COMPLETION problem, which asks for the existence of a set of at most k edges whose insertion makes a graph chordal, and showed that it is equivalent to MINIMUM FILL-IN. Balas and Yu [1] proposed a heuristics algorithm for the maximum clique problem by first finding a maximum (spanning) chordal subgraph. This is equivalent to the CHORDAL EDGE DELETION problem, which asks for the existence of a set of at most k edges whose deletion makes a graph chordal. Dearing et al. [5] observed that a maximum chordal subgraph can also be used to find maximum independent set and sparse matrix completion. This observation turns out to be archetypal: many NP-hard problems (coloring, maximum clique, etc.) are known to be solvable in polynomial time when restricted to chordal graphs, and hence admit a similar heuristics algorithm. Cai [3] considered the parameterized complexity of coloring

* Research supported by the European Research Council (ERC) grant 280152 and the Hungarian Scientific Research Fund (OTKA) grant NK105645.
† Work partially done as a Ph.D. student at Texas A&M University, when he was supported by US NSF under the grants CCF-0830455 and CCF-0917288.



© Yixin Cao and Dániel Marx;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 214–225

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


problems on graphs close to certain graph classes. In particular, he asked as an open question on the graphs that can be made chordal by the deletion of k vertices or edges, of which the edge version was resolved by Marx [11] affirmatively. It should be noted that such a coloring algorithm needs first the set of k vertices or edges. For chordal graphs, to find them is equivalent to solving the CHORDAL VERTEX/EDGE DELETION problem. Though with slightly different purpose, the inspiration behind [1, 5] and [3] are exactly the same.

All the three problems, unfortunately but understandably, are NP-hard [18, 13, 10]. Therefore, early work of Cai [2] and Kaplan et al. [8] focused on their parameterized complexity. They proved that that the CHORDAL COMPLETION problem can be solved in time $4^k \cdot n^{O(1)}$, implying that it is fixed-parameter tractable (FPT). Marx [12] showed that the complementary deletion problems, both edge and vertex versions, are also FPT. Recently, Fomin and Villanger [7] gave an algorithm for CHORDAL COMPLETION with running time $k^{O(\sqrt{k})} \cdot n^{O(1)}$, that is, with subexponential dependence on k.

The three operations can be combined, and then the question becomes: can a graph be made chordal by deleting at most k_1 vertices and k_2 edges and adding at most k_3 edges. This leads us to the CHORDAL EDITING problem, which generalizes all three aforementioned problems in a natural way. Note that chordal graphs are hereditary, hence it does not make sense to add new vertices. The budgets for different operations are not transferable, as otherwise it degenerates to CHORDAL VERTEX DELETION. Our main result establishes the fixed-parameter tractability of CHORDAL EDITING parameterized by $k := k_1 + k_2 + k_3$.

▶ **Theorem 1.1 (Main result).** *There is a $2^{O(k \log k)} \cdot n^{O(1)}$ time algorithm for deciding, given an n-vertex graph G, whether there are a set V_- of at most k_1 vertices, a set E_- of at most k_2 edges, and a set E_+ of at most k_3 non-edges, such that the deletion of V_- and E_- and the addition of E_+ make G a chordal graph.*

As a corollary, we get a new FPT algorithm for the special case CHORDAL DELETION; our algorithm is far simpler and faster than the algorithm of [12].

Related work. Observing that a large hole cannot be fixed by the insertion of a small number of edges, it is easy to devise a bounded search tree algorithm for the CHORDAL COMPLETION problem [8, 2]. No such simple argument works for the deletion versions: the removal of a single vertex/edge suffices to break a hole of an arbitrary length. The way Marx [12] showed that this problem is FPT is to (1) prove that if the graph contains a large clique, then we can identify an irrelevant vertex whose deletion does not change the problem; and (2) observe that if the graph has no large cliques, then it has bounded treewidth, so the problem can be solved by standard techniques, such as the application of Courcelle's Theorem. In contrast, our algorithm uses simple reductions and structural properties, which reveal a better understanding of the CHORDAL VERTEX DELETION problem, and easily extend to the more general CHORDAL EDITING problem.

We remark that there were formulations that consider both edge operations, e.g., the CLUSTER EDITING problem [4], as well as the many problems studied by Natanzon et al. [13]. Their objective is to minimize the total number of edge operations, i.e., $k_2 + k_3$ in our notation, which is slightly different from them. As a matter of fact, our problem formulation is more general: if we can solve the version where the edge additions and edge deletions are bounded separately, then we can try every combination of k_2 and k_3 where $k_2 + k_3$ satisfies the given bound.

Our techniques. As a standard opening step, we use the *iterative compression* method introduced by Reed et al. [14] and concentrate on the compression problem, where we are

equipped with a hole cover M. The subgraph $G - M$ is chordal and hence admits a clique tree decomposition. First, we break every short hole by simple branching. The main technical idea appears in the way we break long holes. We use the clique decomposition to show that the shortest hole H can be decomposed into a bounded number of segments, where the internal vertices of each segment, as well as the part of the graph "close" to them behave in a well-structured and simple way with respect to their interaction with M. To break H, we have to break some of the segments, and the properties of the segments allow us to show that we need to consider only a bounded number of canonical separators breaking these segements. Therefore, we can branch on chosing one of these canonical separators and break the hole using it, resulting in an FPT algorithm.

Notation. All graphs discussed in this paper shall always be undirected and simple. The length $|H|$ of a hole H is defined to be the number of edges in it; note that $|H| = |V(H)|$. If a pair of vertices is adjacent, we say $u \sim v$. By $v \sim X$ we mean v is adjacent to at least one vertex of the set X. Two vertex sets X and Y are completely connected if $x \sim y$ for each pair of $x \in X$ and $y \in Y$. A vertex is *simplicial* if $N(v)$ induce a clique. The notation $N_U(v)$ stands for the neighbors of v in the set U, i.e., $N_U(v) = N(v) \cap U$, regardless of whether $v \in U$ or not. We use $N_H(v)$ as a shorthand for $N_{V(H)}(v)$.

A set S of vertices *separates* x and y, and is called an *(x, y)-separator* if there is no (x, y)-path in the subgraph $G - S$; it is *minimal* if no proper subset of S separates x and y. A graph is chordal if and only if every minimal separator in it induces a clique [6].

Let $\mathcal{T}$ be a tree whose nodes, called *bags*, correspond to the maximal cliques of a graph G. With the customary abuse of notation, the same symbol K is used for a bag in $\mathcal{T}$ and its corresponding maximal clique of G. Let $\mathcal{T}(x)$ denote the subgraph of $\mathcal{T}$ induced by all bags containing x. The tree $\mathcal{T}$ is a *clique tree* of G if for any vertex $x \in V(G)$, the subgraph $\mathcal{T}(x)$ is connected. It is known that the intersection of any pair of adjacent bags K_i and K_j of $\mathcal{T}$ makes a minimal separator; in particular, it is a separator for any pair of vertices $x \in K_i \setminus K_j$ and $y \in K_j \setminus K_i$. A vertex is simplicial if and only if it belongs to exactly one maximal clique; thus, any non-simplicial vertex appears in some minimal separator(s) [9].

2 Outline of the algorithm

A subset $V_- \subseteq V(G)$ is called a *hole cover* of G if its deletion makes G chordal. We say that (V_-, E_-, E_+), where $V_- \subseteq V(G)$ and $E_- \subseteq E(G)$ and $E_+ \subseteq V(G)^2 \setminus E(G)$, is a *chordal editing set* of G if the deletion of V_- and E_- and the addition of E_+, applied successively, make G chordal. Its *size* is defined to be the 3-tuple $(|V_-|, |E_-|, |E_+|)$, and we say that it is *smaller* than (k_1, k_2, k_3) if all of $|V_-| \leq k_1$ and $|E_-| \leq k_2$ and $|E_+| \leq k_3$ hold true and at least one inequality is strict. Note that since chordal graphs are hereditary, it does not make sense to add new vertices. The main problem studied in the paper is formally defined as follows.

CHORDAL EDITING $(G, (k_1, k_2, k_3))$	
Input:	A graph G and three nonnegative integers k_1, k_2, and k_3.
Task:	Either construct a chordal editing set (V_-, E_-, E_+) of G that has size at most (k_1, k_2, k_3), or report that no such a set exists.

One might be tempted to define the editing problem by imposing a combined quota on the total number of operations, i.e., a single parameter $k = k_1 + k_2 + k_3$, instead of three separate parameters. However, this formulation is computationally equivalent to CHORDAL DELETION in a trivial sense, as vertex deletions are clearly preferable to both edge operations.

0. **return** if G is chordal or one of k_1, k_2, and k_3 becomes negative;
1. find a shortest hole H;
2. **if** H is shorter than $k+4$ **then** guess a way to fix it; **goto** 0.
3. **else** decompose H into $O(k^3)$ segments;
guess a segment and break it;
4. **goto** 0.

Figure 1 Outline of our algorithm for CHORDAL EDITING COMPRESSION.

We use the technique of *iterative compression*: we define and solve a compression version of the problem first and argue that this implies the fixed-parameter tractability of the original problem. In the compression problem a hole cover M of bounded size is given in the input, making the problem somewhat easier: as $G - M$ is chordal, we have useful structural information about the graph. Note that the definition below has a slightly technical (but standard) additional condition, i.e., we are not allowed to delete a vertex in M.

CHORDAL EDITING COMPRESSION $(G, M, (k_1, k_2, k_3))$
Input: A graph G, three nonnegative integers k_1, k_2, and k_3, and a hole cover M of G whose size is at most $k_1 + k_2 + k_3 + 1$.
Task: Either construct a chordal editing set (V_-, E_-, E_+) of G such that its size is at most (k_1, k_2, k_3) and V_- is disjoint from M, or report that no such a set exists.

The set M is called the *modulator* of this instance. We use $k := k_1 + k_2 + k_3$ to denote the total numbers of operations.

We sketch how the technique of iterative compression can be applied to use an algorithm for CHORDAL EDITING COMPRESSION to solve CHORDAL EDITING.

Let $v_1, v_2, \ldots, v_n$ be an arbitrary ordering of $V(G)$, and let G^i be the graph induced by $\{v_1, \ldots, v_i\}$. We try to find a chordal editing set of size (k_1, k_2, k_3) for each G^i. Assume that we have obtained a solution (V_-^i, E_-^i, E_+^i) for G^i, then we can make a hole cover X^i of G^i by taking V_-^i, and an arbitrary endvertex from each edge in $E_-^i \cup E_+^i$. Clearly, $X^i \cup \{v_{i+1}\}$ is a hole cover of G^{i+1}. By guessing the (possibly empty) set X_-^i of vertices of a hypothetical solution that is in $X^i \cup \{v_{i+1}\}$ and deleting them from G^{i+1}, we make an instance of CHORDAL EDITING COMPRESSION where the graph is $G^{i+1} - X_-^i$, the modulator is $M^{i+1} := X^i \cup \{v_{i+1}\} \setminus X_-^i$, and the parameters are $(k_1 - |X_-^i|, k_2, k_3)$. Then the compression algorithm for CHORDAL EDITING COMPRESSION can be used to find a chordal editing set disjoint from M^{i+1} for $G^{i+1} - X_-^i$. If the answer is "NO," then we can conclude that the original instance is also "NO." Otherwise the obtained solution, together with X_-^i, gives the solution $(V_-^{i+1}, E_-^{i+1}, E_+^{i+1})$ for G^{i+1}. We proceed to G^{i+2}, until we reach G^n which is G. Hence the original problem is solved with at most n calls of the algorithm for CHORDAL EDITING COMPRESSION.

The main part of this paper will be focused on an algorithm for CHORDAL EDITING COMPRESSION. Its outline is described in Figure 1. We will endeavor to prove

▶ **Theorem 2.1.** CHORDAL EDITING COMPRESSION *is solvable in time* $2^{O(k \log k)} \cdot n^{O(1)}$.

Steps 1 and 2 are straightforward: we can find a shortest hole H in polynomial time, and if $|H| \leq k + 3$, then there are only $O(k^2)$ ways to fix it. To fix a hole of length $|H| \geq k + 4 > k_3 + 3$, we need to delete at least one vertex or edge from it. As we shall see in Section 3, such a hole can be divided into a bounded number of "segments" and the deletions have to "break" at least one of the segments (i.e., delete one vertex or edge from it). In our

case, breaking a segment means a strange mixed form of separation: we have to separate two vertices by removing both edges and vertices. We study this notion of mixed separation on chordal graphs in Section 4. Finally, we show in Section 5 that there is a bounded number of canonical ways of breaking a segment and we may branch on choosing one segment and one of the canonical ways of breaking it. This completes the proofs of Theorem 2.1 and 1.1.

3 Segments

We shall define a hierarchy of vertex sets V_0, V_1, and V_2. Each set is a subset of the preceding one, and all of them induce chordal subgraphs. Let A denote the set of common neighbors of the shortest hole H found in step 1, and define $A_M = A \cap M$ and $A_0 = A \setminus M$. We can assume that A induces a clique: if two vertices $x, y \in A$ are nonadjacent, then together with the two nonadjacent vertices v_1 and v_3 of H, they form a 4-hole (xv_1yv_3x). The following observation follows from the fact that H is the shortest hole of G.

▶ **Proposition 3.1.** A vertex not in A is adjacent to at most three vertices of H and these vertices have to be consecutive in H.

The first set is defined by $V_0 = V(G) \setminus (M \cup A)$. Note that $\{M, V_0, A_0\}$ partitions $V(G)$, and H is disjoint from A_0. Since $|H| \geq k + 4 > |M|$, the hole H intersects both M and V_0. Every component of $H - M$ is an induced path of G_0, and there are at most $|M|$ such paths. Observing $|M| = O(k)$, to decompose H into $O(k^3)$ segments as claimed, it suffices to divide each of these paths into $O(k^2)$ parts. Let P denote such a path $(v_1v_2 \dots v_p)$. To avoid triviality, we may assume $p > 3$; as a result and by Proposition 3.1, the distance between v_1 and v_p in G_0 is at least 3. A further consequence is $v_1 \not\sim v_p$.

Let G_0 denote the chordal subgraph $G[V_0]$, and let $\mathcal{T}$ be a fixed clique tree for G_0. We take the unique path of bags $\mathcal{P} = (K_1, \dots, K_q)$ that connects the disjoint subtrees $\mathcal{T}(v_1)$ and $\mathcal{T}(v_p)$ in $\mathcal{T}$, where $K_1 \in \mathcal{T}(v_1)$ and $K_q \in \mathcal{T}(v_p)$. The condition $p > 3$ implies that $q > 2$. The removal of K_1 and K_q will separate $\mathcal{T}$ into a set of subtrees, one of which contains all K_ℓ with $1 < \ell < q$; let $\mathcal{T}_1$ denote this nonempty subtree. The second set, V_1, is defined to be the union of all bags in $\mathcal{T}_1$ and $\{v_1, v_p\}$. By definition and observing that V_1 fully contains P, it induces a connected subgraph.

We then focus on bags in $\mathcal{P}$ and their union. (One may have judiciously observed that vertices in bags of $\mathcal{P}$ induce an interval graph.) From the definition of clique tree, we can infer that v_1 and v_p appear only in K_1 and K_q respectively, while every internal vertex of P appears in more than one bags of $\mathcal{P}$. For every i with $1 \leq i \leq p$, we denote by $\texttt{first}(i)$ (resp., $\texttt{last}(i)$) the smallest (resp., largest) index ℓ such that $1 \leq \ell \leq q$ and $v_i \in K_\ell$, e.g., $\texttt{first}(1) = \texttt{last}(1) = \texttt{first}(2) = 1$ and $\texttt{last}(p-1) = \texttt{first}(p) = \texttt{last}(p) = q$. As P is an induced path, for each i with $1 < i < p$, we have

$$\texttt{first}(i) \leq \texttt{last}(i-1) < \texttt{first}(i+1) \leq \texttt{last}(i). \tag{1}$$

For $1 \leq \ell < q$, we define $S_\ell := K_\ell \cap K_{\ell+1}$. For any pair of nonadjacent vertices v_i, v_j in P, (i.e., $1 \leq i < i+1 < j \leq p$,) all minimal (v_i, v_j)-separators are then $\{S_\ell \mid \texttt{last}(i) \leq \ell < \texttt{first}(j)\}$.

The third set, V_2, is defined to be the union of vertices in all induced (v_1, v_p)-paths in G_0. Since a vertex x is an internal vertex of an induced (v_1, v_p)-path of G_0 if and only if it is in some minimal (v_1, v_p)-separator of G_0, we have (noting $q > 2$)

▶ **Proposition 3.2.** A vertex is in $V_2 \setminus \{v_1, v_p\}$ if and only if it appears in more than one bags of $\mathcal{P}$. Moreover, $V_2 \setminus \{v_1, v_p\} \subseteq \bigcup_{1<\ell<q} K_\ell$.

The definition of V_0 and G_0 depend upon the hole H, while the definition of V_1 and V_2 depend upon both the hole H and the path P. In this paper, we are always concerned with a particular path of a particular hole, which will be specified before the usage of V_1 and V_2.

The set $V_0 \setminus V_1$ is easily understood, and we now consider $V_1 \setminus V_2$. Given a pair of nonadjacent vertices $x, y \in V_2$, we say that x lies to the *left* (resp., *right*) of y if the bags of $\mathcal{P}$ containing x have smaller (resp., greater) indices than those containing y. If an induced path of $G[V_2]$ consists of three or more vertices, then its endvertices are nonadjacent and have a left-right relation. This relation can be extended to all pairs of consecutive (and adjacent) vertices x, y in this path, the one with smaller distance to the left endvertex of the path is said *to the left of the other*. It is easy to verify that these two definitions are compatible.

▶ **Lemma 3.3.** *For any component C of the subgraph induced by $V_1 \setminus V_2$, the set $N_{V_0}(C)$ induces a clique and there exists ℓ such that $1 < \ell < q$ and $N_{V_0}(C) \subseteq K_\ell$.*

Proof. Consider a vertex $x \in C$, which is different from v_1 and v_p. Since $x \in V_1$, it appears in some bag of $\mathcal{T}_1$. Recall that the only bag of $\mathcal{T}_1$ that is adjacent to K_1 is K_2. Thus if $x \in K_1$, then it has to be in K_2 as well, which is impossible as $x \notin V_2$ (Proposition 3.2). Therefore, $x \notin K_1$; for the same reason, $x \notin K_q$. As a result, $N_{V_0}(x) \subseteq V_1$, and then $N_{V_0}(C) \subseteq V_2$.

It now suffices to show that $N_{V_0}(C)$ induces a clique. Suppose that, for contradiction, there is a pair of nonadjacent vertices $x, y \in N_{V_0}(C)$. We can find an induced (v_1, v_p)-path P' through x and y; without loss of generality, let x lie to the left of y, i.e., $P' = (v_1 \cdots x \cdots y \cdots v_p)$. Let x' and y' be the first and last vertices in P' that are adjacent to C, and $(x'P''y')$ be an induced path with all internal vertices from C. Note that x' either is x or lies to the left of x in P' and y' either is y or lies to the right of y, which imply $x' \not\sim y'$. Thus $(v_1 \cdots x'P''y' \cdots v_p)$ is an induced (v_1, v_p)-path through C, which is impossible. This completes the proof. ◀

Such a component C is called a *branch* of P, and we say that it is *near to* $v_i \in P$ if there is an ℓ with $\texttt{first}(i) \leq \ell \leq \texttt{last}(i)$ satisfying the condition of Lemma 3.3. Since a component C is near to $v_i \in P$ if and only if $N_{V_0}(C) \subseteq N[v_i]$, and applying Proposition 3.1 on any vertex in $N_{V_0}(C)$, we conclude that a branch is near to at most three vertices of P. If a hole passes through C, then C has to be adjacent to M: by Lemma 3.3, $N_{V_0}(C)$ is a clique, thus a hole cannot enter and leave C both via $N_{V_0}(C)$. The converse is not necessarily true: some branch that is adjacent to M might still be disjoint from all holes, e.g., if $N(C)$ is a clique. This observation inspires us to generalize the definition of simplicial vertices to sets of vertices.

▶ **Definition 3.4.** A set X of vertices is called *simplicial in a graph G* if $N[X]$ induces a chordal subgraph of G and $N(X)$ induces a clique of G.

It is easy to verify that a simplicial set of vertices is disjoint from all holes. This suggests that simplicial sets are irrelevant to CHORDAL EDITING problem and we may never want to add/delete edges incident to a vertex in a simplicial set. However, this is not true in general, and we may need to add/delete such edges if $N(X)$ was modified. As characterized by the following lemma, this is the only reason for touching X in the solution: set X will only concern us after $N(X)$ has been changed. We say that a chordal editing set (V_-, E_-, E_+) *edits* a set $X \subset V(G)$ of vertices if either V_- contains a vertex of X or $E_- \cup E_+$ contains an edge with at least one endpoint in X. We use a classic result of Dirac [6] stating that the graph obtained by identifying two cliques of the same size from two chordal graphs is also chordal.

▶ **Lemma 3.5.** *A minimal chordal editing set edits a simplicial set U only if it removes at least one edge induced by $N(U)$.*

Proof. Let (V_-, E_-, E_+) be a minimal editing set of G such that E_- does not contain any edge induced by $N(U)$. We restrict the editing set to the subgraph $G-U$, i.e., we consider the set $(V_- \setminus U, E_- \setminus (U \times V(G)), E_+ \setminus (U \times V(G)))$, and let G' be the graph obtained by applying it to G. Clearly $G' - U = G - U$ is chordal, where $N(U) \setminus V_-$ induces a clique. Also chordal is the subgraph of G' induced by $N[U] \setminus V_-$. Both of them contain the clique $N(U) \setminus V_-$. Since G' can be obtained from them by identifying $N(U) \setminus V_-$, it is also chordal. Then by the minimality of (V_-, E_-, E_+), it must be the same as $(V_- \setminus U, E_- \setminus (U \times V(G)), E_+ \setminus (U \times V(G)))$, and this proves this lemma. ◀

Now we are ready to define segments of P, which are delimited by some special vertices called junctions. By definition, a branch is simplicial in G_0, but unnecessarily in G. We say that a vertex $w \notin K$ is *adjacent* to a bag K if w is adjacent to at least one vertex in K.

▶ **Definition 3.6** (Segment)**.** A vertex $v \in P$ is called a *junction (of P)* if *(1)* some bag K that contains v is adjacent to $M \setminus A_M$; *(2)* some branch near to v is adjacent to $M \setminus A_M$; *(3)* some branch near to v is not simplicial in G; or *(4)* $N_{V_2}(v)$ is not completely connected to A. A sub-path $(v_s \cdots v_t)$ of P is called a *segment*, denoted by $[v_s, v_t]$, if v_s and v_t are the only junctions in it.

We point out that the four types are not exclusive, and one junction might be in more than one types. For a junction v of type (1) or (2), we say that the vertex in $M \setminus A_M$ used in its definition *witnesses* it.

▶ Remark. Informally speaking, for a junction v of type (1) or (2), there is a connection from v to $M \setminus A_M$ that is *local* to v in some sense; for a junction v of type (3) or (4), there is a hole near to v, and its disposal might interfere with that of H. If another hole H' intersects a segment $[v_s, v_t]$, then H' has to go through the whole segment, or more specifically, it necessarily enters and exits the segment via $N[v_s]$ and $N[v_t]$, respectively.

The definition of junction and segment extends to all paths of $H - M$. In polynomial time, we can construct V_0 for H and V_1, V_2 for each path P of $H - M$, from which all junctions of H can be identified. For each path of $H - M$, the endvertices are adjacent to $M \setminus A_M$, hence junctions. As a result, every vertex in $V(H) \setminus M$ is contained in some segment, and in each path of $H - M$, the number of segments is the number of junctions minus one.

We are now ready for the main result of this section that gives a cubic bound on the number of segments of H. It should be noted the constants—both the exponent and the coefficient—in the following statement are not tight, and the current values simplify the argument significantly. Recall that a vertex not in A sees at most three vertices in H, and they have to be consecutive.

▶ **Theorem 3.7.** *If H contains more than $|M| \cdot (12k^2 + 92k + 82)$ segments, then we can either find a vertex that has to be in V_-, or return "NO."*

Proof. We show that H contains at most $|M| \cdot (12k^2 + 92k + 82)$ junctions. Recall that there are at most $|M|$ paths in $H - M$. To obtain a contradiction, we suppose that some path P of $H - M$ contains $12k^2 + 92k + 82$ junctions. Let us first attend to junctions of type (1) in P.

▶ **Claim 1.** Each $w \in M \setminus A_M$ witness at most 15 junctions of type (1).

Proof. Suppose, for contradiction, that 15 vertices in H appears in some bag adjacent to w; let X be this set of vertices. Assume first that X is consecutive. At most 3 of them are adjacent to w, and they are consecutive in H. Thus, we can always pick 6 consecutive vertices from X that are disjoint from $N_H(w)$; let them be $\{v_i, \ldots, v_{i+5}\}$. By definition, there are two vertices $u_1, u_2 \in V_0 \cap N(w)$ such that $u_1 \sim v_i$ and $u_2 \sim v_{i+5}$. It is easy to verify that $u_2 \not\sim v_{i+2}$ and $u_1 \not\sim v_{i+3}$ and $u_1 \not\sim u_2$. Therefore, we can find an induced (u_1, u_2)-path with all interval vertices from $\{v_i, \ldots, v_{i+5}\}$. The length of this path is at least 3, and hence it makes a hole with w of length at most 9. Assume now that X is not consecutive in P, then we can pick a pair of nonadjacent vertices v_i, v_j from X such that the $v_\ell \notin X$ for every $i < \ell < j$. There are two vertices $u_1, u_2 \in V_0 \cap N(w)$ such that $u_1 \sim v_i$ and $u_2 \sim v_j$. It is easy to verify that $(wu_1v_i \cdots v_ju_2w)$ is a hole. By assumption that $|X| \geq 15$, we have $j - i \leq |H| - 13$. In either case, we end with a hole strictly shorter than H. The contradictions prove this claim. ⌟

▶ **Claim 2.** If some vertex $w \in M \setminus A_M$ witnesses $5k + 80$ junctions of the first two types in P, then we can return "NO."

Proof. Let X be this set of junctions, we order them according to their indices in P and group each consecutive five from the beginning. We omit groups that contain junctions of type (1) witnessed by w, and in each remaining group, we pair the second and last vertices in it. According to Claim 1, we end with at least $k + 1$ pairs, which we denote by (v_{ℓ_1}, v_{r_1}), $\cdots$, $(v_{\ell_{k+1}}, v_{r_{k+1}})$, $\cdots$.

For each pair (v_{ℓ_j}, v_{r_j}), where $1 \leq j \leq k + 1$, we construct a hole H_j as follows. By definition, there is a branch C_{ℓ_j} (resp., C_{r_j}) whose neighborhood in H is a proper subset of $\{v_{\ell_j-1}, v_{\ell_j}, v_{\ell_j+1}\}$ (resp., $\{v_{r_j-1}, v_{r_j}, v_{r_j+1}\}$). By the selection of the pair v_{ℓ_j} and v_{r_j} (two vertices of X have been skipped in between), they are nonadjacent, and $r_j - \ell_j > 2$. Therefore, C_{ℓ_j} and C_{r_j} are distinct and necessarily nonadjacent. Since C_{ℓ_j} induces a connected subgraph and is adjacent to both w and $\{v_{\ell_j-1}, v_{\ell_j}, v_{\ell_j+1}\}$, we can find an induced (w, v_{ℓ_j+1})-path P_{ℓ_j} with all internal vertices from $C_{\ell_j} \cup \{v_{\ell_j-1}, v_{\ell_j}\}$. Likewise, we can obtain an induced (w, v_{r_j-1})-path P_{r_j} with all internal vertices from $C_{r_j-1} \cup \{v_{r_j}, v_{r_j+1}\}$. These two paths P_{ℓ_j} and P_{r_j}, together with $(v_{\ell_j+1} \ldots v_{r_j-1})$, make the hole H_j: we have $\ell_j + 1 < r_j - 1$; for each $\ell_j + 1 \leq s \leq r_j - 1$, $v_s \not\sim w$; and for each $\ell_j + 1 < s < r_j - 1$, $v_s \not\sim C_{\ell_j}, C_{r_j}$. This hole goes through w. This way we can construct $k + 1$ holes, and it can be easily verified that they intersect only in w. Since we are not allowed to delete w, we cannot fix all these holes by at most k operations. Thus we can return "NO." ⌟

If Claim 2 applies, then we are already done; otherwise, there are at most $|M| \cdot (5k + 80)$ junctions of the first two types. We proceed by considering the set B of junctions that are only of type (3) or (4) but not of the first two types. Its number is at least

$$(12k^2 + 92k + 82) - (5k + 80) \cdot |M| \geq 7k^2 + 7k + 1.$$

We order B according to their indices in P, and let b_i denote the index of the ith vertex of B in P. For each $0 \leq i \leq k(k + 1)$, we use the $(7i + 3)$th vertex of B to construct a hole H_i. Then we argue that this collection of holes either allows us to identify a vertex that has to be in the solution, or conclude infeasibility.

The first case is when there is a pair of nonadjacent vertices $x \in N_{V_2}(v_{b_{7i+3}})$ and $y \in A$. In this case we can assume that x is adjacent to neither $v_{b_{7i+1}}$ nor $v_{b_{7i+5}}$; otherwise $(xv_{b_{7i+1}}yv_{b_{7i+3}}x)$ or $(xv_{b_{7i+3}}yv_{b_{7i+5}}x)$ is a 4-hole, which contradicts the fact that H is the shortest. In other words, x only appears in some bag between $K_{\texttt{last}(b_{7i+1})}$ and $K_{\texttt{first}(b_{7i+5})}$; on the other hand, by definition of V_2, it appears in at least two of these bags. There is thus

an induced $(v_{b_{7i+1}}, v_{b_{7i+5}})$-path P_i via x in $G[V_2]$. Starting from x, we traverse P_i to the left until the first vertex x_1 that is adjacent to y; the existence of such a vertex is ensured by the fact that $y \sim v_{b_{7i+1}}$. Similarly, we find the first neighbor x_2 of y in P_i to the right of x. Then the sub-path of P_i between x_1 and x_2, together with y, gives the hole H_i. By construction, no vertex of $H_i - y$ is adjacent to $v_{b_{7i}}$ or $v_{b_{7i+6}}$.

In the other case, some branch C_i near to $v_{b_{7i+3}}$ is not simplicial in G. By definition, either the subgraph induced by $N(C_i)$ is not a clique, or the subgraph induced by $N[C_i]$ is not chordal. Since $v_{b_{7i+3}}$ does not satisfy the conditions of type (1) and (2), $N(C_i) \cap M \subseteq A_M$, i.e., $N(C_i) \setminus V_0 \subseteq A$. On the other hand, according to Lemma 3.3, $N(C_i) \cap V_0$ induces a clique. Therefore, there must be a pair of nonadjacent vertices $x \in N(C_i) \cap V_0$ and $y \in A_M$. As C_i is near to $v_{b_{7i+3}}$, it must hold that $x \in N(v_{b_{7i+3}})$; this has already been discussed in the previous case. Suppose now that $N(C_i)$ induces a clique and there is a hole H_i in $N[C_i]$. We have seen that $N[C_i] \cap M = A_M$, thus this hole H_i intersects A_M; let w be a vertex in $V(H_i) \cap A_M$. If H_i is disjoint from A_0, then no vertex in $H_i \setminus M$ can be adjacent to $v_{b_{7i}}$ or $v_{b_{7i+5}}$. Otherwise, it contains some vertex $u \in A_0$; noting that A induces a clique, $H_i \cap A = \{u, w\}$. Moreover, $N(C_i) \cap V_2$ is in the neighborhood of $v_{b_{7i+3}}$ and therefore $N(C_i) \cap V_2$ and $N(C_j) \cap V_2$ are disjoint for $i \neq j$: the existence of a vertex $x \in V_2$ adjacent to both C_i and C_j would contradict Proposition 3.1 (noting that the distance of $v_{b_{7i+3}}$ and $v_{b_{7j+3}}$ is greater than 2 on the hole H).

In sum, we have a set $\mathcal{H}$ of at least $k(k+1)+1$ distinct holes such that (1) each hole in $\mathcal{H}$ contains at most one vertex of A_0, and (2) the intersection of any pair of them is in A. Recall that each hole has length at least $k+4$, hence cannot be fixed by edge additions only. If there is a $u \in A_0$ contained in at least $k+1$ holes of $\mathcal{H}$, then we have to put u into V_-; otherwise we have to delete distinct elements (edges or vertices) to break different holes, which is impossible. Now assume that no such a vertex u exists, then there must be $k+1$ holes that intersect only in M, which allow us to return "NO." ◀

4 Mixed separators in chordal graphs

Given a pair of nonadjacent vertices x, y of a graph, we say that a pair of vertex set V_S and edge set E_S is a *mixed (x, y)-separator* if the deletion of V_S and E_S leaves x and y in two different components; its size is defined to be $(|V_S|, |E_S|)$. A mixed (x, y)-separator is *inclusive-wise minimal* if there exists no other mixed (x, y)-separator (V'_S, E'_S) such that $V'_S \subseteq V_S$ and $E'_S \subseteq E_S$ and at least one containment is proper.

▶ **Lemma 4.1.** *Let x and y be a pair of nonadjacent vertices in a chordal graph F. For any pair of nonnegative integers (a, b), we can find a mixed (x, y)-separator of size at most (a, b) or asserts its nonexistence in time $3^{a+b+1} \cdot |V(F)|^{O(1)}$.*

Another interpretation of this lemma is

▶ **Corollary 4.2.** *Let x and y be a pair of nonadjacent vertices in a chordal graph F. For any nonnegative integer $a \leq k_1$, in time $3^{k_1+k_2+1} \cdot |V(F)|^{O(1)}$ we can find the minimum number b such that $b \leq k_2$ and there is a mixed (x, y)-separator of size (a, b) or assert that there is no mixed (x, y)-separator of size (a, k_2).*

5 Proof of Theorem 2.1

We are now ready to put everything together and finish the analysis of the algorithm. We say that a chordal editing set is minimum if there exists no chordal editing set with a smaller size. Note that a segment is contained in a unique path of $H - M$, which determines V_1 and V_2.

Proof of Theorem 2.1. Let (V^*_-, E^*_-, E^*_+) be a minimum chordal editing set of G of size no more than (k_1, k_2, k_3). We start from a closer look at how it breaks H; by Theorem 3.7, we may assume that H contains $O(k^3)$ segments. There are three options for breaking H. In the first case, V^*_- contains some junction, or E^*_- contains some edge of H that is in $M \times V_0$. In this case, we can branch on including one of these vertices or edges into the solution; there are $O(k^3)$ of them. Otherwise, we need to delete an internal vertex or edge from some segment. Let $d = 2k + 4$. In the second case, there is either (1) some i with $s < i \leq s + d$ such that $v_i \in V^*_-$ or $v_{i-1}v_i \in E^*_-$; or (2) some j with $t - d \leq j < t$ such that $v_j \in V^*_-$ or $v_j v_{j+1} \in E^*_-$. In particular, if the segment to be broken satisfies $t - s \leq 2d$, then we must be in this case. If one of the two aforementioned cases is correct, then we can identify one vertex or edge of the solution by branching. In total, there are $O(k^4)$ branches we need to try.

Henceforth, we assume that none of these two cases holds. We still have to delete at least one vertex or edge from H; this vertex or edge must belong to some segment $[v_s, v_t]$ with $t - s > 2d$. For such a segment, we consider V_1 and V_2 corresponding to it. For any pair of indices i, j with $s \leq i < i + 3 \leq j \leq t$, we use $U_{[i,j]}$ to denote the union of the set of bags in the nonempty subtree of $\mathcal{T} - \{K_{\mathtt{last}(i)}, K_{\mathtt{first}(j)}\}$ that contains $\{K_{\mathtt{last}(i)+1}, \ldots, K_{\mathtt{first}(j)-1}\}$ as well as $\{v_i, v_j\}$. Let $G_{[i,j]}$ be the subgraph induced by $U_{[i,j]}$.

▶ **Claim 3.** There must be some segment $[v_s, v_t]$ such that vertices v_{s+d} and v_{t-d} are disconnected in $G_{[s,t]} - V^*_- - E^*_-$.

Proof. We prove by contradiction. For a segment $[v_s, v_t]$ with $t - s \leq 2d$, the path $(v_s \cdots v_t)$ remains intact in $G - V^*_- - E^*_-$. Thus it suffices to consider segments $[v_s, v_t]$ with $t - s > 2d$. Let $s' = s + d$ and $t' = t - d$. For such a segment, we can find an induced (v_s, v_t)-path $P_{[s,t]}$ in $G_{[s,t]} - V^*_- - E^*_-$, which is also an (unnecessarily induced) path of G. This path has to visit every bag K_ℓ with $\mathtt{last}(s) \leq \ell \leq \mathtt{first}(t)$. In other words, in the original graph G, the path $P_{[s,t]}$ intersects every $N[v_i]$ with $s < i \leq s'$. Since we delete at most $k_2 \leq k$ edges each of which is adjacent to a single vertex in the sub-path $(v_s \cdots v_{s'})$, and $(d - k_2) \geq k + 4$, there must be a vertex $v_{s''}$ with $s'' \geq s + k + 4$ that is not incident to any edge in E^*_-. This vertex is either in or adjacent to $P_{[s,t]}$ in $G_{[s,t]} - V^*_- - E^*_-$. Likewise, we can find a vertex $v_{t''}$ with $t'' \leq t - k - 4$ that is in or adjacent to $P_{[s,t]}$ in $G_{[s,t]} - V^*_- - E^*_-$. We now change the path into $(v_s \cdots v_{s''} P' v_{t''} \cdots v_t)$, where P' is an induced $(v_{s''}, v_{t''})$-path with all internal vertices from $P_{[s,t]}$.

Let s''' with $s \leq s''' \leq s''$ be the smallest index such that $v_{s'''}$ is adjacent to P'. We argue that $s''' \geq s'' - 2$. Otherwise, the neighbor x of $v_{s'''}$ in P' (noting that it is not in A) is to the left of $v_{s''}$. Any path from $v_{s'''}$ to v_t in G_0 has to visit $N[v_{s''}]$. Since no edge incident to $v_{s''}$ is deleted, the path P' has a chord, which is impossible. Similarly, let t''' with $t'' \leq t''' \leq t$ be the greatest such that $v_{t'''}$ is adjacent to P', and we have $t''' \leq t'' + 2$. We can take an induced $(v_{s'''}, v_{t'''})$-path of $G - V^*_- - E^*_-$ with all internal vertices from P', and extend it by including $(v_s \cdots v_{s'''})$ and $(v_{t'''} \cdots v_t)$ to make a chordless (v_s, v_t)-path $P'_{[s,t]}$ in $G - V^*_- - E^*_-$. The length of this path is at least $2(k + 4 - 2) \geq 2k_3 + 4$.

Therefore, for each segment $[v_s, v_t]$ of H, we have obtained an induced (v_s, v_t)-path $P'_{[s,t]}$ in $G - V^*_- - E^*_-$. Concatenating all these paths, as well as edges of H in $M \times V(G)$, we get a cycle C. To verify that C is a hole, it suffices to verify that the internal vertices of $P'_{[s,t]}$ is disjoint and nonadjacent to other parts of C. On the one hand, no internal vertex of $P'_{[s,t]}$ is adjacent to $M \setminus A_M$ by definition (C is disjoint from A). On the other hand, all internal vertices of $P'_{[s,t]}$ appear in the subtree that contains $K_{\mathtt{last}(s+4)}$ in $\mathcal{T} - \{K_{\mathtt{last}(s+3)}, K_{\mathtt{first}(t-3)}\}$, while no vertex in the (v_t, v_s)-path in C does. This verifies that C is a hole of $G - V^*_- - E^*_-$. Since the length of C is longer than $2k_3 + 4$, there must be a hole after the addition of E^*_+, which contains at most k_3 edges. This contradiction proves the claim. ◀

In other words, (V_-^*, E_-^*) contains some inclusive-wise minimal mixed $(\{v_s, \ldots, v_{s'}\}, \{v_{t'}, \ldots, v_t\})$-separator (V_S^*, E_S^*) in $G_{[s,t]}$. The resulting graph obtained by deleting (V_S^*, E_S^*) from $G_{[s,t]}$ is characterized by the following claim.

▶ **Claim 4.** Let (V_S, E_S) be an inclusive-wise minimal mixed $(v_{s'}, v_{t'})$-separator in $G[U_{[s',t']}]$. For any pair of indices s'', t'' with $s \leq s'' \leq s' < t' \leq t'' \leq t$, both $X \setminus \{v_{s''}\}$ and $Y \setminus \{v_{t''}\}$ are simplicial in $G' = G - V_S - E_S$, where X and Y be the components of $G_{[s'',t'']} - V_S - E_S$ containing $v_{s''}$ and $v_{t''}$, respectively.

Proof. It is easy to verify that $N_{G'}(X \setminus \{v_{s''}\}) \subseteq (K_{\texttt{last}(s'')} \cap V_2) \cup A$. The set $K_{\texttt{last}(s'')} \cap V_2$ is completely connected to A; otherwise $s''+1$ is a junction, which is impossible. Let $X' = N_{G'}[X \setminus \{v_{s''}\}]$; a vertex in X' is either in V_2, some branch, or A. We now verify that X' induces a chordal subgraph of G', which means that $X \setminus \{v_{s''}\}$ is simplicial in G'. Since (V_S, E_S) is inclusive-wise minimal, no edge in E_S is induced by X or Y. As a result, for every branch C near to some vertex v_i with $s < i < t$, $C \cap X'$ is simplicial. On the other hand, by definition of segments, $V_2 \cap X'$ is completely connected to A. Therefore, $G'[X']$ is chordal. A symmetric argument applies to $Y \setminus \{v_{t''}\}$. ⌟

We consider the subgraph obtained from G by deleting (V_S^*, E_S^*), i.e., $G' = G - V_S^* - E_S^*$. Note that $(V_-^* \setminus V_S^*, E_-^* \setminus E_S^*, E_+^*)$ is a minimum chordal editing set of G'.

▶ **Claim 5.** For any mixed $(\{v_s, \ldots, v_{s'}\}, \{v_{t'}, \ldots, v_t\})$-separator (V_S^*, E_S^*) of size at most $(|V_S^*|, |E_S^*|)$ in $G_{[s,t]}$, substituting (V_S, E_S) for (V_S^*, E_S^*) in (V_-^*, E_-^*, E_+^*) gives another minimum editing set to G.

Proof. We first argue the existence of some vertex $v_{s''}$ with $s \leq s'' \leq s'$ such that E_- contains no edge induced by $K_{\texttt{last}(s'')}$. For each s'' with $s \leq s'' \leq s'$, since $\texttt{last}(s'') \geq \texttt{first}(s''+1)$ and every vertex in them is adjacent to at most 3 vertices of H (Proposition 3.1), bags $K_{\texttt{last}(s'')}$ and $K_{\texttt{last}(s''+2)}$ are disjoint. In particular, an edge cannot be induced by both $K_{\texttt{last}(s'')}$ and $K_{\texttt{last}(s''+2)}$. Suppose that E_- contains an edge induced by $K_{\texttt{last}(s'')}$ for each s'' with $s \leq s'' < s'$, then we must have $|E_-| > (s'-s)/2 \geq k_2$, which is impossible. Likewise, we have some vertex $v_{t''}$ with $t' \leq t'' \leq t$ such that E_- contains no edge induced by $K_{\texttt{last}(t'')}$. By Claim 4, it follows that every vertex of $U_{[s'',t'']}$ is in a simplicial set of $G - V_S^* - E_S^*$. Since $(V_-^* \setminus V_S^*, E_-^* \setminus E_S^*, E_+^*)$ is a minimum chordal editing set to $G - V_S^* - E_S^*$, we have by Lemma 3.5 that $(V_-^* \setminus V_S^*, E_-^* \setminus E_S^*, E_+^*)$ does not edit any vertex of $U_{[s'',t'']}$.

Suppose that there is a hole C in the graph obtained by applying $((V_-^* \setminus V_S^*) \cup V_S, (E_-^* \setminus E_S^*) \cup E_S, E_+^*)$ to G. By construction, C contains a vertex of $U_{[s',t']} \subseteq U_{[s'',t'']}$. However, by Claim 4, every vertex of $U_{[s'',t'']}$ is in some simplicial set of $G - V_S - E_S$ and, as $(V_-^* \setminus V_S^*, E_-^* \setminus E_S^*, E_+^*)$ does not edit $U_{[s'',t'']}$, every such vertex is in a simplical set after applying $((V_-^* \setminus V_S^*) \cup V_S, (E_-^* \setminus E_S^*) \cup E_S, E_+^*)$ to G. Thus no vertex of $U_{[s'',t'']}$ is on a hole, a contradiction. ⌟

For any segment $[v_s, v_t]$, we can use Corollary 4.2 to find all possible sizes of minimum mixed $(\{v_s, \ldots, v_{s'}\}, \{v_{t'}, \ldots, v_t\})$-separator. There are at most k_1 of them. By Claim 5, one of them can be used to compose a minimum chordal editing set. In each iteration, we branch into $O(k^4)$ instances to break a hole, and in each branch decreases k by at least 1. The runtime is thus $O(k)^{4k} \cdot n^{O(1)} = 2^{O(k \log k)} \cdot n^{O(1)}$. This completes the proof. ◀

Acknowledgement. We thank Sylvain Guillemot for a careful reading of an early version of this paper.

References

1. E. Balas and C. S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM J. Comput.*, 15(4):1054–1068, 1986.
2. L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Proc. Letters*, 58(4):171–176, 1996.
3. L. Cai. Parameterized complexity of vertex colouring. *Discrete Appl. Math.*, 127(3):415–429, 2003.
4. Y. Cao and J. Chen. Cluster editing: kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012.
5. P. Dearing, D. Shier, and D. Warner. Maximal chordal subgraphs. *Discrete Appl. Math.*, 20(3):181–190, 1988.
6. G. A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg*, 25(1):71–76, 1961.
7. F. V. Fomin and Y. Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM J. Comput.*, 42(6):2197–2216, 2013. A preliminary version appeared in SODA 2012.
8. H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.*, 28(5):1906–1922, 1999. A preliminary version appeared in FOCS 1994.
9. J. G. Lewis, B. W. Peyton, and A. Pothen. A fast algorithm for reordering sparse matrices for parallel factorization. *SIAM J. Sci. Comput.*, 10(6):1146–1173, Nov. 1989.
10. J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. System Sci.*, 20(2):219–230, 1980.
11. D. Marx. Parameterized coloring problems on chordal graphs. *Theor. Comp. Sci.*, 351(3):407–424, 2006.
12. D. Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 57(4):747–768, 2010.
13. A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Discrete Appl. Math.*, 113(1):109–128, 2001. A preliminary version appeared in LNCS vol. 1665, WG 1999.
14. B. A. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Oper. Res. Letters*, 32(4):299–301, 2004.
15. D. J. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, 32(3):597–609, 1970.
16. D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Reed, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, New York, 1973.
17. J. Xue. Edge-maximal triangulated subgraphs and heuristics for the maximum clique problem. *Networks*, 24(2):109–120, 1994.
18. M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Discrete Methods*, 2(1):77–79, 1981.

Online Dynamic Power Management with Hard Real-Time Guarantees*

Jian-Jia Chen[1], Mong-Jen Kao[2], D. T. Lee[2,3], Ignaz Rutter[1], and Dorothea Wagner[1]

1 **Faculty for Informatics, Karlsruhe Institute of Technology (KIT), Germany**
j.chen@kit.edu, rutter@kit.edu, dorothea.wagner@kit.edu
2 **Institute for Information Science, Academia Sinica, Taipei, Taiwan**
mong@iis.sinica.edu.tw
3 **Department of Computer Science and Information Engineering, National Chung-Hsing University, Tai-Chung, Taiwan**
dtlee@ieee.org

Abstract

We consider the problem of online dynamic power management that provides hard real-time guarantees for multi-processor systems. In this problem, a set of jobs, each associated with an arrival time, a deadline, and an execution time, arrives to the system in an online fashion. The objective is to compute a non-migrative preemptive schedule of the jobs and a sequence of power on/off operations of the processors so as to minimize the total energy consumption while ensuring that all the deadlines of the jobs are met. We assume that we can use as many processors as necessary. In this paper we examine the complexity of this problem and provide online strategies that lead to practical energy-efficient solutions for real-time multi-processor systems.

First, we consider the case for which we know in advance that the set of jobs can be scheduled feasibly on a single processor. We show that, even in this case, the competitive factor of any online algorithm is at least 2.06. On the other hand, we give a 4-competitive online algorithm that uses at most two processors. For jobs with unit execution times, the competitive factor of this algorithm improves to 3.59.

Second, we relax our assumption by considering as input multiple streams of jobs, each of which can be scheduled feasibly on a single processor. We present a trade-off between the energy-efficiency of the schedule and the number of processors to be used. More specifically, for k given job streams and h processors with $h > k$, we give a scheduling strategy such that the energy usage is at most $4 \cdot \lceil \frac{k}{h-k} \rceil$ times that used by any schedule which schedules each of the k streams on a separate processor. Finally, we drop the assumptions on the input set of jobs. We show that the competitive factor of any online algorithm is at least 2.28, even for the case of unit job execution times for which we further derive an $O(1)$-competitive algorithm.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Energy-Efficient Scheduling, Online Dynamic Power Management

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.226

1 Introduction

Reducing power consumption and improving energy efficiency have become important design requirements in computing systems. For mobile devices, effective power management

* This work was supported in part by National Science Council (NSC), Taiwan, under Grants NSC101-2221-E-005-026-MY2 and NSC101-2221-E-005-019-MY2.



© Jian-Jia Chen, Mong-Jen Kao, D. T. Lee, Iganz Rutter, and Dorothea Wagner;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 226–238



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



can considerably extend the standby period and prolong battery lifetime. For large-scale computing clusters, appropriately powering down the idling processing units can considerably reduce the electricity bill.

In order to increase the energy efficiency, two different mechanisms have been introduced to reduce the energy consumption. *(1) Power-down Mechanism:* When a processor is idling, it can be put into a low-power state, e.g., sleep or power-off. While the processor consumes less power in these states, a fixed amount of energy is required to switch the system back to work. In the literature, the problem of deciding the sequence of state transitions is referred to as *dynamic power management (DPM). (2) Dynamic Speed Scaling:* The concept of dynamic speed scaling refers to the flexibility provided by a processor to adjust its processing speed dynamically. The rate of energy consumption is typically described by a convex function of the processing speed. This feature is also referred to as *dynamic voltage frequency scaling (DVFS)*, following its practical implementation scheme.

The majority of previous work regarding energy-efficient scheduling focuses mainly on uni-processor systems. For systems that support power-down mechanism, Baptiste [5] considered hard real-time jobs, i.e., deadline misses of jobs are not allowed, with unit execution times and proposed the first polynomial-time algorithm that computes an optimal strategy for turning on and powering off a processor. In a follow-up paper, Baptiste et al. [6] further extended the result to jobs with arbitrary execution times and reduced the time complexity. When considering workload-conserving scheduling, i.e., the system is not allowed to enter low-power states when the ready queue is not empty, Augustine et al. [4] considered systems with multiple low-power states and provided online algorithms.

Dynamic speed scaling was introduced to allow computing systems to reach a balance between high performance and low power consumption dynamically. Hence, scheduling algorithms that assume dynamic speed scaling, e.g., Yao et al. [30], usually execute jobs as slowly as possible while ensuring that timing constraints are met. When the energy required to keep the processor active is not negligible, however, executing jobs too slowly may result in more energy consumption. For most realistic power-consumption functions, there exist a *critical speed*, which is the most energy-efficient for job execution [12, 22].

Irani et al. [22] initiated the study of combining both mechanisms. For offline energy-minimization, they presented a 2-approximation. For the online version, they introduced a *greedy procrastinating principle*, which enables online algorithms that have certain properties and that are designed for speed scaling without power-down mechanism to additionally support the power-down mechanism. The idea behind this principle is to postpone job execution as much as possible in order to bundle workload for batch execution. The usage of job procrastination with dynamic speed scaling for periodic tasks has later been explored extensively in a series of studies [11, 12, 26].

The combination of the power-down mechanism with dynamic speed scaling suggests the philosophy of *racing-to-idle*: Execute jobs at higher speeds and gain longer quality sleeping intervals. Albers and Antoniadis [1] showed that the problem of minimizing the energy consumption for speed scaling with a sleep state is NP-hard and provided a $\frac{4}{3}$-approximation.

All of the aforementioned work mainly focuses on uni-processor systems. By contrast, for multi-processor systems, relatively fewer results are known. Demaine et al. [17] considered unit jobs and presented a polynomial-time algorithm based on dynamic programming for power-down mechanism. Approximations for several variations were also presented. In a follow-up paper, Demaine and Zadimoghaddam [18] presented logarithmic approximations for general formulations of scheduling problems with submodular objective functions, including energy consumption. Albers et al. [2] considered dynamic speed scaling with job migration

and presented polynomial-time algorithms based on maximum flow problems.

As scheduling to meet deadline constraints is a long-standing difficult problem [14–16, 20], additional augmentation on the hardware level, e.g., speed of the processors or number of the machines, has been considered to provide practical solutions. See, for example, [3, 8, 10, 28]. In practice, machine augmentation follows the trends in multi-core systems, while speed augmentation has been shown its limits as overclocking is difficult to achieve due to the dramatic increase of power consumption and thermal dissipation.

Our Focus and Contribution. In this paper, we examine the problem of online dynamic power management that provides hard real-time guarantees, i.e., each job must finish its execution before its deadline, for multi-processor systems. We assume a system equipped with multiple processors that are identical and that operate independently from each other, and we can use as many processors as necessary. Job executions can be preempted but can not be migrated, i.e., the execution of a job must be done on the same processor. The objective is to compute a schedule of the jobs and a sequence of switch on/off operations of the processors so as to minimize the total energy consumption. For this problem model we give an elaborate study that leads to practical energy-efficient solutions for real-time multi-processor systems.

First, we consider the case for which we know in advance that the set of jobs can be scheduled feasibly on one processor. We show that the competitive factor of any online algorithm is at least 2.06, even for this restricted case. Then we propose the idea of *energy-efficient anchors*, which are defined for each of the jobs, to indicate a proper moment for which the online scheduler should no longer postpone the execution of the jobs. We show that this idea leads to a 4-competitive online algorithm which uses at most two processors. For jobs with unit execution times, we show that the competitive factor improves to 3.59.

Second, we relax the conditions of our assumption by considering as input multiple streams of jobs, each of which can be scheduled feasibly on one processor. We present a simple strategy, as a byproduct of our first algorithm, to allow a trade-off between the number of processors we have and the energy-efficiency of the resulting schedule. More specifically, for k given job streams and h processors with $h > k$, we give a scheduling strategy such that the energy usage is at most $4 \cdot \left\lceil \frac{k}{h-k} \right\rceil$ times that used by any schedule which schedules each of the k streams on a separate processor.

The above algorithms lead to practical energy-efficient solutions in real-time systems for which partitioned scheduling with recurrent real-time task model is adopted [7, 9]. The recurrent task models, such as the *sporadic real-time task* model [27] or the *arrival curve* model in Real-Time Calculus (RTC) [29], describe an infinite sequence of job arrivals, generated by the tasks. Under such a model, the worst-case characteristics of job arrivals are specified. For example, a sporadic real-time task defines the minimum inter-arrival time of any two consecutive jobs. With the partitioned scheduling scheme, it is required that all the jobs generated by a recurrent task be executed on a single processor. In many cases, however, the real-time system needs to provide hard deadline guarantees and verifying the system could be very costly, and the goal is to make the schedule more efficient by using more processors without going through the costly verification steps. Therefore, even though the partitioned scheduling scheme is more restricted in the sense that the jobs are partitioned in advance, it has been widely adopted in practical real-time systems [7, 9] as it incurs no additional overhead for ensuring the feasibility of the resulting online schedule. Our algorithms provide an online energy-efficient solution with a reasonable trade-off for this situation.

Finally we drop the assumptions on the schedulability as well as the number of job

streams and consider general set of jobs with unit execution times.

We show that the competitive factor of any online algorithm is at least 2.28. For the positive side, we present a $O(1)$-competitive algorithm, which combines ideas from different results and is interesting in its own right.

2 Notations and Problem Definition

In this section, we formally define the scheduling problem we consider and introduce notations that will be used throughout this paper. Each job, say, j, is associated with three non-negative integral parameters, namely, the arrival time a_j, the execution time c_j, and the deadline d_j. The arrival time of a job is the time it arrives to the system. The execution time is the amount of CPU time it requires to finish its execution, and the deadline is the latest moment at which the job must be completed. We assume that c_j and d_j are known at the moment when j arrives to the system.

For notational brevity, we use a triple $j = (a_j, d_j, c_j)$ to denote the corresponding parameters for any job j. We say a job j is a unit job if $c_j = 1$ and we write $j = (a_j, d_j)$. In addition, a job j is said to be *urgent* if $c_j = d_j - a_j$.

We make the following assumptions on the processors. When a processor is *off*, it cannot execute any job and consumes no power. Switching on, or, alternatively, turning on, a processor requires E_w units of energy. When a processor is on, it can execute jobs at a fixed speed. We say that a processor is in the *busy* state if it is executing a job. If a processor is on but not executing any job, then it is said to be in the *standby* state. The energy consumed by a processor per unit of time, i.e., the power consumption, is ψ_b when it is busy and ψ_σ when it is in standby, respectively. We assume that $\psi_\sigma \leq \psi_b$. Initially all processors are off.

The *break-even time*, denoted by $\mathcal{B}$, is defined to be E_w/ψ_σ. Intuitively, this corresponds to the amount of time a processor has to stay in standby in order to have the same energy consumption for switching on a processor. The break-even time is an important concept that has been widely used for ski-rental-related problems [25] and dynamic power management algorithms in the literature, e.g., [21–23].

Below we define the concept of job scheduling. Let $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ be the set of processors and $\mathcal{J}$ be a set of jobs.

▶ **Definition 1** (A Schedule for a Set of Jobs). A schedule $\mathbf{S}$ for $\mathcal{J}$ on $\mathcal{M}$ is a set of pairs $(\mathcal{I}_1, \text{job}_1), (\mathcal{I}_2, \text{job}_2), \ldots, (\mathcal{I}_m, \text{job}_m)$, where for each $1 \leq i \leq m$,

- $\mathcal{I}_i$ denotes the set of time intervals during which processor M_i is on, and
- $\text{job}_i \colon \mathbb{R}^+ \longrightarrow \mathcal{J} \cup \{\emptyset\}$ is a function of time t that indicates the job to be executed on processor M_i at time t. If M_i is not executing any job or is off at time t, then $\text{job}_i(t) = \emptyset$.

The schedule $\mathbf{S}$ is said to be *feasible* for $\mathcal{J}$ on $\mathcal{M}$ if for each job $j \in \mathcal{J}$, there exists a processor $M_i \in \mathcal{M}$ such that

$$\sum_{\mathrm{I} \in \mathcal{I}_i} \int_{\mathrm{I} \cap [a_j, d_j]} \delta(\text{job}_i(t), j) \cdot \mathrm{d}t = c_j,$$

where $\delta(x, y)$ is defined to be 1 if $x = y$ and 0 otherwise. In other words, the schedule $\mathbf{S}$ is feasible if for each job j, there exists a processor M_i such that the amount of time M_i is executing j during the time interval $[a_j, d_j]$ is c_j. We remark that, it is implicitly implied in the definition that a feasible schedule is also a preemptive schedule and the jobs can only be executed without migration.

The number of processors the schedule $\mathbf{S}$ uses is defined to be $|\{i\colon 1 \le i \le m, \mathcal{I}_i \neq \emptyset\}|$, i.e., the number of processors that have been switched on at least once in $\mathbf{S}$. The energy consumption of the schedule $\mathbf{S}$, denoted $\mathrm{E}(\mathbf{S})$, is defined as

$$\mathrm{E}(\mathbf{S}) = \sum_{1 \le i \le m} \left(E_w \cdot |\mathcal{I}_i| + \sum_{I \in \mathcal{I}_i} |I| \cdot \psi_\sigma \right) + \sum_{j \in \mathcal{J}} c_j \cdot (\psi_b - \psi_\sigma),$$

where $|\mathcal{I}_i|$ denotes cardinality of $\mathcal{I}_i$, i.e., the number of time intervals it contains, and $|I|$ denotes the length of the time interval I.

▶ **Definition 2** (DPM Job Scheduling)**.** Given a set $\mathcal{J}$ of jobs and a set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ of processors, the *DPM Job Scheduling Problem* is to find a feasible schedule $\mathbf{S}$ for $\mathcal{J}$ on $\mathcal{M}$ such that $\mathrm{E}(\mathbf{S})$ is minimized.

In this paper, we consider the online version of the DPM job scheduling problem in which the jobs arrive to the system dynamically in an online manner, i.e., at any time t, the algorithm only sees the jobs whose arrival times are less than or equal to t, and the scheduling decisions have to be made without prior knowledge on future job arrivals. To be more precise, let $\mathcal{J}$ be the input job set and $\mathcal{J}(t) = \{j\colon j \in \mathcal{J}, a_j \le t\}$ be the subset of $\mathcal{J}$ that contains the jobs whose arrival times are no greater than t. At any time t, the algorithm sees the job set $\mathcal{J}(t)$ and is able to decide the assignment of the jobs to the processors as well as the state transitions of the processors, i.e., turning on or switching off, at that time. The objective is to compute a feasible schedule for $\mathcal{J}$ such that the energy consumed is minimized.

An online algorithm Π is said to be η-competitive for the online DPM job scheduling problem if for any job set $\mathcal{J}$, we have $\mathrm{E}(\Pi(\mathcal{J})) \le \eta \cdot \mathrm{E}(Opt(\mathcal{J})) + x$, where $\Pi(\mathcal{J})$ is the schedule computed by algorithm Π, $Opt(\mathcal{J})$ is an optimal schedule for $\mathcal{J}$, i.e., the one that results in the minimum energy consumption, and x is a constant.

3 Preliminary Results

In this section, we present preliminary results that are related to our assumptions for the problem models we addressed. We begin with the characterization of the job sets that are packable on one processor. For any job set $\mathcal{J}$ and any $0 \le \ell < r$, let $\Upsilon(\mathcal{J}, \ell, r) = \{j : j \in \mathcal{J},\ \ell \le a_j,\ d_j \le r\}$ be the set of jobs that arrive and have to be done within the time interval $[\ell, r]$. Let $\Upsilon^{\#}(\mathcal{J}, \ell, r) = \sum_{j \in \Upsilon(\mathcal{J}, \ell, r)} c_j$ denote the total amount of workload in $\Upsilon(\mathcal{J}, \ell, r)$.

Chetto et al. [13] studied the schedulability of any given set of jobs using the *earliest-deadline-first (EDF)* principle, which always selects the job with the earliest deadline for execution at any moment, and proved the following lemma.

▶ **Lemma 3** (Chetto et al. [13])**.** *For any set $\mathcal{J}$ of jobs, $\mathcal{J}$ can be scheduled on one processor using the EDF principle if and only if the following condition holds:*

$$\textit{For any time interval } [\ell, r], \quad \Upsilon^{\#}(\mathcal{J}, \ell, r) \le r - \ell. \tag{1}$$

It is well-known that, for any job set $\mathcal{J}$, if there exists a feasible schedule that uses only one processor for $\mathcal{J}$, then the EDF principle is also guaranteed to produce a feasible schedule [19]. Hence, Condition (1) gives a necessary and sufficient condition for any set of jobs to be able to be packable on a processor.

However, even when the set of jobs is known in advance to be packable on a processor, we still need multiple processors in order to achieve energy-efficiency in an online setting. This is illustrated by the following lemma.

▶ **Lemma 4.** *Let* Π *be an online algorithm that produces feasible schedules using only one processor for any job set that can be packed feasibly on one processor. For any* $\alpha > 0$*, there exists a job set* $\mathcal{J}_\alpha$ *that can be packed feasibly on one processor such that the competitive factor of* Π *on* $\mathcal{J}_\alpha$ *is at least* α*.*

Hence it is essential to use additional processors in order to give an energy-efficient scheduling scheme for the online DPM job scheduling problem. This dilemma is further extended in §4.1 to obtain a lower bound on the competitive factor of any online algorithm.

Below we introduce notions related to the number of processors required by a set of *unit jobs*. For any job set $\mathcal{J}$ and any $0 \le \ell < r$, let

$$\rho(\mathcal{J}, \ell, r) = \frac{\Upsilon^{\#}(\mathcal{J}, \ell, r)}{r - \ell}$$

denote the density of workload of $\mathcal{J}$ with respect to the time interval $[\ell, r]$. Let $\hat{\varrho}(\mathcal{J}) = \max_{0 \le \ell < r} \rho(\mathcal{J}, \ell, r)$ denote the maximum density of $\mathcal{J}$. Let $\mathrm{P}^{\#}(\mathcal{J})$ denote the minimum number of processors required by any feasible schedule for $\mathcal{J}$. The following lemma gives an alternative characterization of $\mathrm{P}^{\#}(\mathcal{J})$ when $\mathcal{J}$ is composed merely by unit jobs.

▶ **Lemma 5** ([24]). *For any set* $\mathcal{J}$ *of unit jobs, we have* $\mathrm{P}^{\#}(\mathcal{J}) = \lceil \hat{\varrho}(\mathcal{J}) \rceil$*.*

However, for jobs with arbitrary execution times, packing the jobs using a minimum number of processors is a long-standing difficult problem even for the offline case [14–16, 20], and for the online version only very special cases were studied [16, 24].

4 Online Scheduling for Job Sets Packable on One Processor

In this section, we consider the case for which we know in advance that the input set of jobs can be scheduled feasibly on one processor, i.e., Condition (1) from Lemma 3 holds for the input set of jobs. First we prove a lower bound of 2.06 on the competitive factor of any online algorithm by designing an online adversary $\mathcal{A}$ that observes the behavior of the scheduling algorithm and that determines the forthcoming job sequence. In §4.2 we present an online strategy that gives a 4-competitive schedule using at most two processors. In §4.3 we show that this strategy leads to a 3.59-competitive schedule when the jobs have unit execution times.

4.1 Lower Bound on the Competitive Factor

Let Π be an online scheduling algorithm for this problem. We set $\psi_b = \psi_\sigma = \psi = 1$ and $E_w = k$, where k is an integer chosen to be sufficiently large. Hence the break-even time $\mathcal{B}$ is also k. We define a *monitor* operation of the adversary $\mathcal{A}$ as follows.

▶ **Definition 6.** When $\mathcal{A}$ **monitors** Π during time interval $[t_0, t_1]$, it checks if Π keeps at least one processor on between time t_0 and t_1. If Π turns off all the processors at some point t between t_0 and t_1, then $\mathcal{A}$ releases an urgent unit job $(t + 1, t + 2)$, forcing Π to turn on at least one processor to process it. If Π keeps at least one processor on during the monitored period, then $\mathcal{A}$ does nothing.

Let x, η, and χ, where $0 \leq x \leq \frac{2}{5}$, be three non-negative parameters to be chosen. The online adversary works as follows. At time 0, $\mathcal{A}$ releases a unit job $(0, \mathcal{B})$ and observes the behavior of Π. Let t be the moment at which Π schedules this job to execute. Since Π produces a feasible schedule, we know that $0 \leq t \leq \mathcal{B} - 1$. We have the following two cases.

Case(1): If $0 \leq t \leq \left(\frac{1}{2} - x\right)\mathcal{B}$, then $\mathcal{A}$ monitors Π from time t to $\frac{3}{2}\mathcal{B}$.
Case(2): If j is not executed till $\left(\frac{1}{2} - x\right)\mathcal{B}$, the adversary $\mathcal{A}$ releases $\left(\frac{1}{2} + x\right)\mathcal{B} - 1$ unit jobs with deadline $\mathcal{B}$ at time $\left(\frac{1}{2} - x\right)\mathcal{B} + 1$. As a result, the online algorithm is forced to switch on at least two processors to execute these jobs. The adversary continues to monitor Π until time $\left(\frac{3}{2} + \eta\right)\mathcal{B}$. If no urgent unit job has been released till time $\left(\frac{3}{2} + \eta\right)\mathcal{B}$, the adversary $\mathcal{A}$ terminates. Otherwise, $\mathcal{A}$ monitors Π for another $\chi\mathcal{B}$ units of time until $\left(\frac{3}{2} + \eta + \chi\right)\mathcal{B}$.

Let $\mathcal{E}(\Pi)$ and $\mathcal{E}(\mathcal{O})$ denote the energy consumed by algorithm Π and an offline optimal schedule on the input sequence generated by $\mathcal{A}$, respectively. By deriving a lower bound on $\mathcal{E}(\Pi)$ and an upper bound on $\mathcal{E}(\mathcal{O})$, we obtain the following theorem.

▶ **Theorem 7.** *The competitive factor of any online algorithm for the online DPM job scheduling problem is at least* 2.06*, even for the case of unit jobs that are known in advance to be packable on one processor.*

4.2 4-Competitive Online Scheduling

In order to design an online algorithm that produces an energy-efficient schedule, we have to deal with two questions. (1) To what extent should we bundle the execution of the jobs in order to achieve energy-efficiency? (2) Provided that the job execution may be delayed, how do we guarantee the feasibility of the resulting schedule?

For the former question, we introduce the concept of *energy-efficient anchors* for the jobs to determine the appropriate timing to begin their execution. For the latter question, the feasibility is guaranteed by suitably partitioning the job set such that the jobs that are delayed still satisfy Condition (1) from Lemma 3. Below we describe our algorithm in more detail. Let $\mathcal{J}$ be the input set of jobs, and recall that $\mathcal{J}(t)$ is the subset of jobs whose arrival times are smaller than or equal to t.

For any $t, t^\dagger$ with $0 \leq t \leq t^\dagger$, let $\mathbf{Q}(t)$ be the subset of $\mathcal{J}(t)$ that contains the jobs that have not yet finished their execution up to time t, and let $\mathbf{Q}(t, t^\dagger)$ be the subset of $\mathbf{Q}(t)$ containing those jobs whose deadlines are smaller than or equal to $t^\dagger$. Note that, by definition, we have $\mathbf{Q}(t, t^\dagger) \subseteq \mathbf{Q}(t) \subseteq \mathcal{J}(t) \subseteq \mathcal{J}$. For notational brevity, let $c'_j(t)$ denote the remaining execution time of job j at time t, and let $W(t) = \sum_{j \in \mathbf{Q}(t)} c'_j(t)$ and $W(t, t^\dagger) = \sum_{j \in \mathbf{Q}(t, t^\dagger)} c'_j(t)$ denote the total remaining execution time of the jobs in $\mathbf{Q}(t)$ and $\mathbf{Q}(t, t^\dagger)$, respectively. Furthermore, we divide $\mathbf{Q}(t)$ into two subsets according to the arrival times of the jobs. For any t, t^* with $0 \leq t^* \leq t$, let $\mathbf{Q}^{t^*}_{proc}(t)$ be the subset of $\mathbf{Q}(t)$ containing the jobs whose arrival times are less than t^*, and let $\mathbf{Q}^{t^*}_{forth}(t) = \mathbf{Q}(t) \backslash \mathbf{Q}^{t^*}_{proc}(t)$.

Let λ, $0 \leq \lambda \leq 1$, be a constant to be determined later. For each job $j \in \mathcal{J}$, we define a parameter h_j to be $\max\{a_j, d_j - \lambda\mathcal{B}\}$. The value h_j is referred to as the *energy-efficient anchor* for job j.

Let M_1 and M_2 denote the two processors which our algorithm $\mathcal{S}$ will manage. We say that the system is *busy*, if at least one processor is executing a job. The system is said to be *off* if both processors are turned off. Otherwise, the system is said to be in *standby*. During the process of job scheduling, our algorithm $\mathcal{S}$ maintains an *urgency flag*, which is initialized to be *false*. The description of the algorithm $\mathcal{S}$ is provided in Table 1.

Table 1 A description of the online scheduling algorithm $\mathcal{S}$.

At any time t, $\mathcal{S}$ proceeds as follows.

(A) Conditions for turning on the processors:	(B) Handling the job scheduling:	(C) Conditions for turning off the processors:
1. **If** the system is *off* and there exists some $j \in \mathbf{Q}(t)$ such that $h_j \leq t$, **then** turn on processor M_1. 2. **If** the *urgency flag* is *false* and there exists some $t^\dagger$ with $t^\dagger > t$ such that $W(t, t^\dagger) > t^\dagger - t$, **then** - turn on M_1 if it is *off*, - turn on M_2, set t^* to be t, and set the *urgency flag* to be *true*.	1. **If** the *urgency flag* is *true*, **then** use the EDF principle to schedule jobs from $\mathbf{Q}^{t^*}_{proc}(t)$ on M_1 and jobs from $\mathbf{Q}^{t^*}_{forth}(t)$ on M_2. 2. **If** the *urgency flag* is *false* and the system is not *off*, **then** use the EDF principle to schedule jobs from $\mathbf{Q}(t)$ on the processor that is *on*.	1. **If** the *urgency flag* is *true* and $\mathbf{Q}^{t^*}_{proc}(t)$ empty, **then** turn off M_1 and set the *urgency flag* to be *false*. 2. **If** the *urgency flag* is *false*, the system is *standby*, and $t - t_1 \geq \mathcal{B}$, where t_1 is the time processor M_1 was turned on, **then** turn off all processors.

Note that, M_1 and M_2 can both be on only when the *urgency flag* is *true*. To prove the claimed competitive factor, we analyze the relative positions between the time intervals during which an optimal offline schedule keeps the system off and our online algorithm is executing jobs. Then we charge the energy consumed by our schedule to that consumed by the optimal offline schedule to obtain the claimed bound.

▶ **Theorem 8.** *By setting $\lambda = 1$, the algorithm $\mathcal{S}$ computes a 4-competitive schedule that uses at most two processors for any set of jobs satisfying Condition (1) for the online DPM job scheduling problem.*

4.3 3.59-Competitive Scheduling for Unit Jobs

When the jobs have unit execution times, we show that we can benefit even more from a properly chosen parameter $\lambda = 4 - \sqrt{10}$. The major difference is that, when the system is in urgency while $\mathbf{Q}^{t^*}_{forth}(t)$ is empty, i.e, processor M_2 is in standby, we use a global earliest-deadline-first scheduling by executing two jobs on M_1 and M_2 instead of keeping one processor in standby, which in turn improves resource utilization. Let $\mathcal{S}^\dagger$ denote the modified algorithm. We have the following theorem.

▶ **Theorem 9.** *By setting $\lambda = 4 - \sqrt{10}$, the algorithm $\mathcal{S}^\dagger$ computes a 3.59-competitive schedule that uses at most two processors for any set of unit jobs satisfying Condition (1) for the online DPM job scheduling problem.*

5 Online Multi-Processor Scheduling

In this section we present results derived for online dynamic power management in multi-processor systems. In §5.1 we generalize the algorithm $\mathcal{S}$ presented in §4.2 for a given set of job streams, each of which delivers a set of jobs that can be scheduled feasibly on one processor. This allows a trade-off between the number of processors we have and the energy-efficiency of the resulting schedule.

Then we consider online dynamic power management for general sets of unit jobs. In §5.2 we prove a lower bound of 2.28 on the competitive factor of any online algorithm. Finally in §5.3 we present an online algorithm that gives a $O(1)$-competitive schedule.

5.1 Trading the Energy-Efficiency with the Number of Processors

In §4.2 we have shown how a stream of jobs satisfying Condition (1) can be scheduled by the algorithm $\mathcal{S}$ to obtain a 4-competitive schedule which uses at most two processors. In this section, we show that, by suitably delaying and bundling the workload, we can generalize the algorithm $\mathcal{S}$ for a given set of job streams, each of which delivers a job set satisfying Condition (1), to allow a trade-off between the number of processors we have and the energy-efficiency of the resulting schedule.

Let $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_k$ be the given set of job streams, where $\mathcal{J}_i$ satisfies Condition (1) for all $1 \leq i \leq k$. Below we present a strategy that leads to a schedule that uses at most h processors for any $h > k$ and whose energy usage is at most $4 \cdot \left\lceil \frac{k}{h-k} \right\rceil$ times that used by any schedule which schedules each of the k streams on a separate processor.

First, if $h \geq 2k$, then we apply the algorithm $\mathcal{S}$ on every pair of the streams, i.e., on $\mathcal{J}_{2i}$ and $\mathcal{J}_{2i+1}$, for all $1 \leq i \leq \frac{k}{2}$, and we get a schedule with a factor of 4. For the case $k < h < 2k$, we divide $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_k$ into $h-k$ subsets such that each subset gets at most $\lceil k/(h-k) \rceil$ streams. The h processors are allocated in the following way. Each stream of jobs, i.e., $\mathcal{J}_i$ for all $1 \leq i \leq k$, gets one processor, and each of the remaining $h-k$ processors are allocated to each of the $h-k$ subsets such that no two processors are allocated to the same subset. Therefore, the problem is reduced to the remaining case, $h = k+1$.

Below we describe how the case $h = k+1$ is handled. Let $M_0, M_1, \ldots, M_k$ denote the $k+1$ processors to be managed, and let $\mathbf{Q}_i$, $1 \leq i \leq k$, be the corresponding ready queue for processor M_i. We use the parameter $\lambda = 1$ to set the energy-efficient anchor for each job that arrives. Recall that $c'_j(t)$ is the remaining execution time of job j at time t. For any $t \geq 0$ and any $t^\dagger$ with $t^\dagger \geq t$, we use $W_0(t, t^\dagger) = \sum_{j \in \mathbf{Q}_0, d_j \leq t^\dagger} c'_j(t)$ to denote the total remaining execution time of the jobs in $\mathbf{Q}_0$ whose deadlines are less than or equal to $t^\dagger$.

The algorithm works as follows. When a job $j \in \mathcal{J}_i$, $1 \leq i \leq k$, arrives to the system, we check whether or not processor M_i is on. If M_i is on, then we add j to $\mathbf{Q}_i$. Otherwise, we further check whether $W_0(t, t^\dagger) + c_j \leq t^\dagger - t$ holds for all $t^\dagger \geq d_j$. If it does, then j is added to $\mathbf{Q}_0$. Otherwise, we add j to $\mathbf{Q}_i$ and turn on the processors M_i and M_0 (if M_0 is off). At any time t, we have the following further conditions to consider.

(a) Conditions for turning on M_0: If the energy-efficient anchor of some job in $\mathbf{Q}_0$ is met or if $\exists\ t^\dagger \geq t$ such that $W_0(t, t^\dagger) \geq t^\dagger - t$, then we turn on M_0 if it is off.

(b) Job scheduling: For each $0 \leq i \leq k$ such that M_i is on, we use the EDF principle to schedule the jobs of $\mathbf{Q}_i$ on M_i.

(c) Conditions for turning off the processors: If $\mathbf{Q}_0$ becomes empty and M_0 has been turned on for at least $\mathcal{B}$ amount of time, then we turn off M_0 immediately. For $1 \leq i \leq k$, if M_i is on, $\mathbf{Q}_i$ becomes empty, and M_0 is off, then we turn off M_i.

Let $\mathcal{S}_{multi}$ denote the algorithm. We have the following theorem.

▶ **Theorem 10.** *Given a set of k job streams, each of which can be scheduled feasibly on one processor, algorithm $\mathcal{S}_{multi}$ computes a schedule that uses at most h processors, for any $h > k$, such that the energy usage is at most $4 \cdot \left\lceil \frac{k}{h-k} \right\rceil$ times that used in any schedule which schedules each of the k streams on a separate processor, for the online DPM job scheduling problem.*

5.2 Lower Bound on the Competitive Factor

In this section we prove a lower bound of 2.28 on the competitive factor of any online algorithm for the online DPM scheduling problem with unit jobs. Let Π be an online

scheduling algorithm for this problem. More specifically, we present an online adversary $\mathcal{A}$, which observes the behavior of Π and which decides the set of forthcoming jobs such that the competitive ratio of Π on the input sequence generated by $\mathcal{A}$ is at least 2.28.

We set $\psi_b = \psi_\sigma = \psi = 1$ and $E_w = k$, where $k \gg 1$ is an integer chosen to be sufficiently large. Hence the break-even time, $\mathcal{B}$, is also k. The adversary $\mathcal{A}$ works in two stages. In the first stage, $\mathcal{A}$ uses the gadget designed in [24] for the machine-minimizing job scheduling problem to force Π to use more processors than necessary. As a result, at the time when the first stage ends, the number of processors Π uses is at least 2.09 times that required by any optimal schedule for $\mathcal{J}^*$ in terms of number of processors. In the second stage, $\mathcal{A}$ monitors the number of active processors Π keeps and makes sure that Π does not turn off the processors too fast. Below we describe the second stage in more detail.

Stage II. Let η be a non-negative real number to be decided later. We define the real-valued function $f_\eta(t)\colon [0,1] \to \mathbb{R}$ to be $f_\eta(t) = \eta + e^t \cdot (3.09 - 2\eta)$, where e is the base of natural logarithm, i.e., the Euler's number.

The adversary $\mathcal{A}$ works as follows. At each time t with $q\alpha^2 \le t < q\alpha^2 + \mathcal{B}$, the adversary checks if the number of processors algorithm Π keeps in the state of on is at least $\hat{\varrho}(\mathcal{J}^*) \cdot f_\eta\left(\frac{t-q\alpha^2}{\mathcal{B}}\right)$. If it is, then $\mathcal{A}$ does nothing. Otherwise, $\mathcal{A}$ punishes the aggressive behavior of Π by releasing $\lceil \hat{\varrho}(\mathcal{J}^*) \rceil$ urgent jobs with deadline $t+1$ and terminates.

▶ **Lemma 11.** *If the algorithm Π gets punished by $\mathcal{A}$, then the competitive factor of the resulting schedule is at least η.*

Lemma 11 gives a bound when the algorithm Π turns off the processors in an aggressive way. On the other hand, if Π does not behave aggressively, then the resulting competitive factor will decrease as η increases. By setting η to be 2.28, we get the following theorem.

▶ **Theorem 12.** *The competitive factor of any online algorithm for the online DPM job scheduling problem is at least* 2.28*, even for unit jobs.*

5.3 O(1)-Competitive Online Scheduling for Unit Jobs

In this section we consider the case for which the jobs have unit execution times. The lower bound result provided in §5.2 gives a rough idea on the difficulty of this problem, which includes the following. (a) First, how many processors should be used when we have no prior knowledge on future job arrivals? (b) Second, how can we turn the standby processors off so that we do not suffer much when we have to turn them on again later?

We incorporate the results of [24] as a partial solution to question (a) mentioned above and give a $O(1)$-competitive online algorithm, denoted $\mathcal{S}^\dagger_{multi}$, for this problem. The algorithm works as follows. At each moment, $\mathcal{S}^\dagger_{multi}$ computes the density of the workload that has arrived to the system "recently" and makes its scheduling decisions accordingly. If the density is *low*, then $\mathcal{S}^\dagger_{multi}$ adopts the strategy presented in §4.2 and §4.3 to bundle the execution of the jobs on two processors. Otherwise, $\mathcal{S}^\dagger_{multi}$ uses the approach suggested in [24] to estimate the number of processors required by future job arrivals for multi-processor scheduling. For question (b), we let each processor stay on for an additional amount of time before it is turned off.

The approach we use combines ideas from different results. Although the idea is conceivable, bounding the energy efficiency is tricky and requires further non-trivial observations on the connections between online schedules and optimal schedules.

Below we describe the algorithm $\mathcal{S}^{\dagger}_{multi}$ in more detail. Let $\mathcal{J}$ denote the input set of jobs and $\mathbf{Q}$ denote the ready queue which contains the set of jobs that arrive and that are not yet scheduled. The algorithm $\mathcal{S}^{\dagger}_{multi}$ maintains a variable t^*, initialized to be -1, to denote the last time when $\mathbf{Q}$ becomes empty. Let $\mathcal{J}^* = \mathcal{J} \backslash \mathcal{J}(t^*)$ be the set of jobs that arrive after time t^*. In addition, the algorithm $\mathcal{S}^{\dagger}_{multi}$ maintains another variable t^*_h to denote the first time for which the workload density becomes greater than or equal to 1 since time t^*, i.e., t^*_h is the smallest integer such that $t^*_h > t^*$ and $\hat{\varrho}(\mathcal{J}^*(t^*_h)) \geq 1$. For notational brevity, t^*_h is set to be ∞ if there is no such moment.

At each time t, the algorithm $\mathcal{S}^{\dagger}_{multi}$ computes the workload density $\hat{\varrho}(\mathcal{J}^*(t))$ and updates the value of t^*_h if necessary. Depending on the value of t^*_h, the jobs that arrive are handled differently. If $t^*_h > t$, then the jobs that have just arrived are added to the *lightly-loaded ready queue* $\mathbf{Q}_\ell$. Otherwise, they are added to the *heavily-loaded ready queue* $\mathbf{Q}_h$.

For the jobs that are added to $\mathbf{Q}_\ell$, we use the algorithm $\mathcal{S}^{\dagger}$ proposed in §4.3 to schedule them. To help describe the algorithm, in the following we use M_1 and M_2 to denote the two specific processors that are used by $\mathcal{S}^{\dagger}$. In addition, we use $\#_i$, where $i \geq 1$, to denote the remaining processors that will be used to handle the jobs that are added to $\mathbf{Q}_h$.

Handling the *heavily-loaded ready queue* $\mathbf{Q}_h$. Let γ_2 be a constant chosen to be 5.2. If $t^*_h > t$, then $\mathbf{Q}_h(t)$ is empty and there is nothing to process. If $t^*_h \leq t$, then the algorithm $\mathcal{S}^{\dagger}_{multi}$ makes sure that at least $\lceil \gamma_2 \cdot \hat{\varrho}(\mathcal{J}^*(t)) \rceil$ processors, excluding M_1 and M_2, have been turned on for job execution. Let $\#(t)$ be the number of processors that are on, excluding M_1 and M_2, and let $\chi = \min\{\#(t), |\mathbf{Q}_h(t)|\}$. We remark that, as $\hat{\varrho}(\mathcal{J}^*(t))$ changes over time, it is possible that $\#(t) > \lceil \gamma_2 \cdot \hat{\varrho}(\mathcal{J}^*(t)) \rceil$.

The algorithm $\mathcal{S}^{\dagger}_{multi}$ fetches χ jobs with earliest deadlines from $\mathbf{Q}_h(t)$ and assigns them for execution on $\#_1, \#_2, \ldots, \#_\chi$. If $|\mathbf{Q}_h(t)| < \#(t)$, then $\mathcal{S}^{\dagger}_{multi}$ continues to fetch jobs from $\mathbf{Q}_\ell(t)$, if there exists any, using the first-fit principle, i.e., the processor with smaller index has higher priority for job execution, such that either all of the $\#(t)$ processors are occupied or $\mathbf{Q}_\ell(t)$ becomes empty.

Turning off the processors. After the scheduling decisions on the ready queues, i.e., $\mathbf{Q}_\ell$ and $\mathbf{Q}_h$, are made, the algorithm $\mathcal{S}^{\dagger}_{multi}$ checks the following conditions. For all i with $1 \leq i \leq \#(t)$, if processor $\#_i$ has stayed in standby for $\mathcal{B}$ amount of time since turned on, then $\mathcal{S}^{\dagger}_{multi}$ switches processor $\#_i$ off immediately.

At any time t, if the ready queue $\mathbf{Q}$ becomes empty after the scheduling decisions on $\mathbf{Q}_\ell$ and $\mathbf{Q}_h$ are made, then t^* is set to t and t^*_h is set to be ∞. We conclude the result with the following theorem.

▶ **Theorem 13.** *The algorithm $\mathcal{S}^{\dagger}_{multi}$ computes a $(\gamma_1 + 52 \cdot \gamma_2 + 1)$-competitive schedule for the online DPM job scheduling problem with unit jobs, where $\gamma_1 = 3.59$ is the competitive factor of $\mathcal{S}^{\dagger}$ and $\gamma_2 = 5.2$ is a constant.*

References

1 Susanne Albers and Antonios Antoniadis. Race to idle: new algorithms for speed scaling with a sleep state. In *SODA*, pages 1266–1285, 2012.

2 Susanne Albers, Antonios Antoniadis, and Gero Greiner. On multi-processor speed scaling with migration: extended abstract. In *SPAA*, pages 279–288, 2011.

3 S. Anand, Naveen Garg, and Nicole Megow. Meeting deadlines: How much speed suffices? In *ICALP (1)*, volume 6755 of *LNCS*, pages 232–243. Springer, 2011.

4 John Augustine, Sandy Irani, and Chaitanya Swamy. Optimal power-down strategies. In *FOCS*, pages 530–539, 2004.

5 P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: A polynomial time algorithm for offline dynamic power management. In *SODA*, pages 364–367, 2006.

6 Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial time algorithms for minimum energy scheduling. In *ESA*, pages 136–150, 2007.

7 Sanjoy K. Baruah and Nathan Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *RTSS*, pages 321–329, 2005.

8 V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. A constant-approximate feasibility test for multiprocessor real-time scheduling. In *ESA*, pages 210–221, 2008.

9 Jian-Jia Chen and S. Chakraborty. Resource augmentation bounds for approximate demand bound functions. In *RTSS*, pages 272–281, 2011.

10 Jian-Jia Chen and S. Chakraborty. Partitioned packing and scheduling for sporadic real-time tasks in identical multiprocessor systems. In *ECRTS*, pages 24–33, 2012.

11 Jian-Jia Chen and Tei-Wei Kuo. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. In *LCTES*, 2006.

12 Jian-Jia Chen and Tei-Wei Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *ICCAD*, 2007.

13 Houssine Chetto and Maryline Silly-Chetto. Scheduling periodic and sporadic tasks in a real-time system. *Inf. Process. Lett.*, 30(4):177–184, 1989.

14 Julia Chuzhoy and Paolo Codenotti. Resource minimization job scheduling. In *APPROX'09/RANDOM'09*, pages 70–83, Berlin, Heidelberg, 2009. Springer-Verlag.

15 Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. Machine minimization for scheduling jobs with interval constraints. In *FOCS'04*, 2004.

16 M. Cieliebak, T. Erlebach, F. Hennecke, B. Weber, and P. Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In *IFIP*. Springer, 2004.

17 E. Demaine, M. Ghodsi, M. Hajiaghayi, A. Sayedi-Roshkhar, and M. Zadimoghaddam. Scheduling to minimize gaps and power consumption. In *SPAA*, pages 46–54, 2007.

18 Erik D. Demaine and Morteza Zadimoghaddam. Scheduling to minimize power consumption using submodular functions. In *SPAA*, pages 21–29, 2010.

19 Michael L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.

20 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co, 1979.

21 Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C. Buttazzo. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Real-Time Systems*, 47(2):163–193, 2011.

22 Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. In *SODA*, pages 37–46, 2003.

23 Sandy Irani, Sandeep K. Shukla, and Rajesh K. Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embedded Comput. Syst.*, 2(3):325–346, 2003.

24 M.-J. Kao, J.-J. Chen, I. Rutter, and D. Wagner. Competitive design and analysis for machine-minimizing job scheduling problem. In *ISAAC*, pages 75–84, 2012.

25 A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, 1994.

26 Yann-Hang Lee, Krishna P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *ECRTS*, pages 105–112, 2003.

27 A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

28 Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *STOC*, pages 140–149, 1997.

29 L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. *ISCAS*, 4:101–104, 2000.

30 F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.

Depth-4 Lower Bounds, Determinantal Complexity: A Unified Approach

Suryajith Chillara and Partha Mukhopadhyay

Chennai Mathematical Institute, Siruseri, India
{suryajith, partham}@cmi.ac.in

Abstract

Tavenas has recently proved that any $n^{O(1)}$-variate and degree n polynomial in VP can be computed by a depth-4 $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuit of size $2^{O(\sqrt{n}\log n)}$ [14]. So, to prove VP $\neq$ VNP it is sufficient to show that an explicit polynomial in VNP of degree n requires $2^{\omega(\sqrt{n}\log n)}$ size depth-4 circuits. Soon after Tavenas' result, for two different explicit polynomials, depth-4 circuit size lower bounds of $2^{\Omega(\sqrt{n}\log n)}$ have been proved (see [7] and [4]). In particular, using combinatorial design Kayal et al. [7] construct an explicit polynomial in VNP that requires depth-4 circuits of size $2^{\Omega(\sqrt{n}\log n)}$ and Fournier et al. [4] show that the iterated matrix multiplication polynomial (which is in VP) also requires $2^{\Omega(\sqrt{n}\log n)}$ size depth-4 circuits.

In this paper, we identify a simple combinatorial property such that any polynomial f that satisfies this property would achieve a similar depth-4 circuit size lower bound. In particular, it does not matter whether f is in VP or in VNP. As a result, we get a simple unified lower bound analysis for the above mentioned polynomials.

Another goal of this paper is to compare our current knowledge of the depth-4 circuit size lower bounds and the determinantal complexity lower bounds. Currently the best known determinantal complexity lower bound is $\Omega(n^2)$ for Permanent of a $n \times n$ matrix (which is a n^2-variate and degree n polynomial) [3]. We prove that the determinantal complexity of the iterated matrix multiplication polynomial is $\Omega(dn)$ where d is the number of matrices and n is the dimension of the matrices. So for $d = n$, we get that the iterated matrix multiplication polynomial achieves the current best known lower bounds in both fronts: depth-4 circuit size and determinantal complexity. Our result also settles the determinantal complexity of the iterated matrix multiplication polynomial to $\Theta(dn)$.

To the best of our knowledge, a $\Theta(n)$ bound for the determinantal complexity for the iterated matrix multiplication polynomial was known only for any constant $d > 1$ [6].

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Arithmetic Circuits, Determinantal Complexity, Depth-4 Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.239

1 Introduction

One of the main challenges in algebraic complexity theory is to separate VP from VNP. This problem is well known as Valiant's hypothesis [15]. This is an algebraic analog of the problem P vs NP. Recall that a multivariate polynomial family $\{f_n(\mathrm{X}) \in \mathbb{F}[x_1, x_2, \ldots, x_n] : n \geq 1\}$ is in the class VP if f_n has degree of at most $\mathrm{poly}(n)$ and can be computed by an arithmetic circuit of size $\mathrm{poly}(n)$. It is in VNP if it can be expressed as

$$f_n(\mathrm{X}) = \sum_{\mathrm{Y} \in \{0,1\}^m} g_{n+m}(\mathrm{X}, \mathrm{Y})$$


© Suryajith Chillara and Partha Mukhopadyay;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 239–250

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


where $m = |\mathrm{Y}| = \mathrm{poly}(n)$ and g_{n+m} is a polynomial in VP. Permanent polynomial characterizes the class VNP over the fields of all characteristics except 2 and the determinant polynomial characterizes the class VP with respect to the quasi-polynomial projections.

▶ **Definition 1.** The determinantal complexity of a polynomial f, over n variables, is the minimum m such that there are affine linear functions $A_{k,\ell}$, $1 \leq k, \ell \leq m$ defined over the same set of variables and $f = \det((A_{k,\ell})_{1 \leq k,\ell \leq m})$. It is denoted by $\mathrm{dc}(f)$.

To resolve Valiant's hypothesis, proving $\mathrm{dc}(\mathrm{perm}_n) = n^{\omega(\log n)}$ is sufficient. Von zur Gathen [16] proved $\mathrm{dc}(\mathrm{perm}_n) \geq \sqrt{\frac{8}{7}}n$. Later Cai [2], Babai and Seress [17], and Meshulam [10] independently improved the lower bound to $\sqrt{2}n$. In 2004, Mignon and Ressayre [11] came up with a new idea of using second order derivatives and proved that $\mathrm{dc}(\mathrm{perm}_n) \geq \frac{n^2}{2}$ over the fields of characteristic zero. Subsequently, Cai et al. [3] extended the result of Mignon and Ressayre to all fields of characteristic $\neq 2$.

For any polynomial f, Valiant [15] proved that $\mathrm{dc}(f) \leq 2(F(f)+1)$ where $F(f)$ is the arithmetic formula complexity of f. Later, Nisan [12] proved that $\mathrm{dc}(f) = O(B(f))$ where $B(f)$ is the arithmetic branching program complexity of f.

Another possible way to prove Valiant's hypothesis is to prove that the permanent polynomial can not be computed by any polynomial size arithmetic circuit. In 2008, Agrawal and Vinay proved that any arithmetic circuit of sub-exponential size can be depth reduced to a depth-4 circuit maintaining a nontrivial upper bound on the size [1]. Subsequently, Koiran [8] and Tavenas [14] have come up with improved depth reductions (in terms of parameters). In particular, Tavenas proved that any $n^{O(1)}$-variate polynomial of degree n in VP can also be computed by a $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$-circuit of top fan-in $2^{O(\sqrt{n}\log n)}$.

In a recent breakthrough, Gupta et al. [5] proved a $2^{\Omega(\sqrt{n})}$ lower bound for the size of the depth-4 circuits computing the determinant or the permanent polynomial using the method of shifted partial derivatives. Subsequently, Kayal et al. [7] improved the situation by proving a $2^{\Omega(\sqrt{n}\log n)}$ depth-4 $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$-circuit size lower bound for an explicit polynomial in VNP.

More precisely, in [7] the following family of polynomials constructed from the combinatorial design of Nisan-Wigderson [13] was considered:

$$\mathrm{NW}_{n,\epsilon}(\mathrm{X}) = \sum_{a(z) \in \mathbb{F}[z]} x_{1a(1)} x_{2a(2)} \cdots x_{na(n)} \,.$$

where $a(z)$ runs over all univariate polynomials of degree $< k = \epsilon\sqrt{n}$ where $0 < \epsilon < 1$ is a suitably fixed parameter, and $\mathbb{F}$ is a finite field of size n. Here we consider the natural identification of $\mathbb{F}$ with the set $\{1, 2, \ldots, n\}$. Since the number of monomials in $\mathrm{NW}_{n,\epsilon}(\mathrm{X})$ is $n^{O(\sqrt{n})}$, the result from [7] gives a tight bound of $2^{\Theta(\sqrt{n}\log n)}$ for the depth-4 circuit complexity of $\mathrm{NW}_{n,\epsilon}(\mathrm{X})$. From the explicitness of the polynomial, it is clear that the polynomial family $\mathrm{NW}_{n,\epsilon}(\mathrm{X})$ is in VNP for any $0 < \epsilon < 1$.

Although the combined implication of [7] and [14] looks very exciting from the perspective of lower bounds, a recent result by Fournier et al. [4] shows that such a lower bound is also obtained by the iterated matrix multiplication polynomial which is in VP. Similar to the works of Gupta et al. [5] and Kayal et al. [7], Fournier et al. also used the method of shifted partial derivatives as their main technical tool. The iterated matrix multiplication polynomial of d generic $n \times n$ matrices $\mathrm{X}^{(1)}, \mathrm{X}^{(2)}, \ldots, \mathrm{X}^{(d)}$ is the $(1,1)$th entry of the product of the matrices. More formally, let $\mathrm{X}^{(1)}, \mathrm{X}^{(2)}, \ldots, \mathrm{X}^{(d)}$ be d generic $n \times n$ matrices with disjoint set of variables and $x_{ij}^{(k)}$ be the variable in $\mathrm{X}^{(k)}$ indexed by $(i,j) \in [n] \times [n]$. Then

the iterated matrix multiplication polynomial (denoted by $\text{IMM}_{n,d}$) is defined as follows:

$$\text{IMM}_{n,d}(\text{X}) = \sum_{i_1,i_2,\ldots,i_{d-1}\in[n]} x^{(1)}_{1i_1} x^{(2)}_{i_1i_2} \ldots x^{(d-1)}_{i_{(d-2)}i_{(d-1)}} x^{(d)}_{i_{(d-1)}1}\,.$$

Notice that $\text{IMM}_{n,d}(\text{X})$ is a $n^2(d-2)+2n$-variate polynomial of degree d. To see that $\text{IMM}_{n,d}(\text{X}) \in \text{VP}$, it is sufficient to observe that it can be computed by a polynomial-size algebraic branching program. For the sake of completeness, we recall the definition of the algebraic branching programs.

▶ **Definition 2.** An algebraic branching program (ABP), over the set of variables X and field $\mathbb{F}$ is a layered (i.e. the edges are only between two consecutive layers) directed acyclic graph G with two special vertices s and t. The weight of an edge is a linear form in $\mathbb{F}[\text{X}]$. The weight of a path is the product of the weights of its edges. The polynomial computed by G is the sum of the weights of all the paths from s to t in G.

To prove $\text{IMM}_{n,d}(\text{X}) \in \text{VP}$, one just needs to observe that for all $1 \leq i \leq d$ the matrix $\text{X}^{(i)}$ can be identified with the adjacency matrix of the subgraph between the layers i and $i+1$. Hence, the result from [4] is also tight and shows the optimality of the depth reduction of Tavenas [14]. Recent work of Kumar and Saraf [9] shows that the depth reduction as shown by [14] is optimal even for the homogenous formulas. This strengthens the result of [4] who proved the optimality of depth reduction for the circuits.

One of the main motivations of our study comes from this tantalizing fact that two seemingly different polynomials $\text{NW}_{n,\epsilon}(\text{X}) \in \text{VNP}$ and $\text{IMM}_{n,d}(\text{X}) \in \text{VP}$ behave very similarly as far as the $2^{\Omega(\sqrt{n}\log n)}$-size lower bound for depth-4 circuits are concerned. In this paper, we seek a conceptual reason for this behaviour. We identify a simple combinatorial property such that any polynomial that satisfies it would require $2^{\Omega(\sqrt{n}\log n)}$-size depth-4 arithmetic circuits. We call it *Leading Monomial Distance Property*. In particular, it does not matter whether the polynomial is easy (i.e. in VP) or hard (i.e. the polynomial is in VNP but not known to be in VP). As a result of this abstraction we present a simple *unified* analysis of the depth-4 circuit size lower bounds for $\text{NW}_{n,\epsilon}(\text{X})$ and $\text{IMM}_{n,d}(\text{X})$.

To define the Leading Monomial Distance Property, we first define the notion of distance between two monomials.

▶ **Definition 3.** Let m_1, m_2 be two monomials over a set of variables. Let S_1 and S_2 be the (multi)-sets of variables corresponding to the monomials m_1 and m_2 respectively. The distance $\text{dist}(m_1, m_2)$ between the monomials m_1 and m_2 is the $\min\{|S_1| - |S_1 \cap S_2|, |S_2| - |S_1 \cap S_2|\}$ where the cardinalities are the order of the (multi)-sets.

For example, let $m_1 = x_1^2x_2x_3^2x_4$ and $m_2 = x_1x_2^2x_3x_5x_6$. Then $S_1 = \{x_1, x_1, x_2, x_3, x_3, x_4\}$, $S_2 = \{x_1, x_2, x_2, x_3, x_5, x_6\}$, $|S_1| = 6$, $|S_2| = 6$ and $\text{dist}(m_1, m_2) = 3$.

We say that a $n^{O(1)}$-variate and n-degree polynomial has the Leading Monomial Distance Property, if the leading monomials of a *large subset* ($\approx n^{\sqrt{n}}$) of its span of the derivatives (of order $\approx \sqrt{n}$) have *good pair-wise distance*. Leading monomials are defined by defining a suitable order on the set of variables. We denote the leading monomial of a polynomial $f(\text{X})$ by $\mathsf{LM}(f)$. More formally, we prove the following theorem in Section 4.

▶ **Theorem 4.** *Let $f(\text{X})$ be a $n^{O(1)}$-variate polynomial of degree n. Let there be $s \geq n^{\delta k}$ (δ is any constant > 0) different polynomials in $\langle \partial^{=k}(f)\rangle$ for $k = \epsilon\sqrt{n}$ such that any two of their leading monomials have pair-wise distance of at least $\Delta \geq \frac{n}{c}$ for any constant $c > 1$, and $0 < \epsilon < \frac{1}{40c}$. Then any depth-4 $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuit that computes $f(\text{X})$ must be of size $e^{\Omega_{\delta,c}(\sqrt{n}\ln n)}$.*

In fact, from the proof it will be clear that the theorem remains valid for any constant ϵ arbitrarily close to $\frac{1}{4c}$. For technical simplicity, we prefer to the state the above theorem in its current form.

Another motivation of this work is to find a connection between our current knowledge of the determinantal complexity lower bounds and the depth-4 circuit size lower bounds. The best known determinantal complexity lower bound for a $n^{O(1)}$-variate and n degree (Permanent) polynomial is $\Omega(n^2)$. Here we ask the following question: can we give an example of an explicit $n^{O(1)}$-variate degree n polynomial in VNP for which the determinantal complexity is $\Omega(n^2)$ and the depth-4 complexity is $2^{\Omega(\sqrt{n}\log n)}$? We settle this problem by showing a $\Omega(n^2)$ lower bound for $\mathrm{dc}(\mathrm{IMM}_{n,n}(\mathrm{X}))$ which is a $O(n^3)$-variate and n-degree polynomial. In particular, we prove the following theorem.

▶ **Theorem 5.** *For any integers n and $d > 1$, the determinantal complexity of the iterated matrix multiplication polynomial $\mathrm{IMM}_{n,d}$ is $\Omega(dn)$.*

Since $\mathrm{IMM}_{n,d}(\mathrm{X})$ has an algebraic branching program of size $O(dn)$ [12], from the above theorem it follows that $\mathrm{dc}(\mathrm{IMM}_{n,d}(\mathrm{X})) = \Theta(dn)$. This improves upon the earlier bound of $\Theta(n)$ for the determinantal complexity of the iterated matrix multiplication polynomial for any constant $d > 1$ [6]. Similar to the approach of [3] and [11], we also use the the rank of Hessian matrix as our main technical tool.

2 Organization

In Section 3, we state a few results from [5], [7], and [14]. In Section 4, we do a unified analysis of the depth-4 lower bound results of [7] and [4]. We prove the determinantal complexity lower bound of $\mathrm{IMM}_{n,d}(\mathrm{X})$ in Section 5. We state a few open problems in Section 6.

3 Preliminaries

The following beautiful lemma (from [5]) is the key to the asymptotic estimates required for the lower bound analyses.

▶ **Lemma 6** (Lemma 6, [5]). *Let $a(n), f(n), g(n) : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be the integer valued functions such that $(f+g) = o(a)$. Then,*

$$\ln \frac{(a+f)!}{(a-g)!} = (f+g)\ln a \pm O\left(\frac{(f+g)^2}{a}\right).$$

In this paper, whenever we apply this lemma, $(f+g)^2$ will be $o(a)$. So, we will not worry about the error term (which will be asymptotically zero) generated by this estimate.

The $\Sigma\Pi^{[D]}\Sigma\Pi^{[t]}$ circuits are depth-4 arithmetic circuits with alternating layers of addition and multiplication gates where the fan-in of the multiplication gates in the bottom layer is bounded by a parameter t and the fan-in of the multiplication gates in the layer adjacent to the output gate is bounded by the parameter D. These circuits compute polynomials of the form $C = \sum_{i=1}^{s} \prod_{j=1}^{D_i} Q_{ij}(\mathrm{X})$ where the degree of the polynomial Q_{ij} is bounded by t for all i and j.

Building on the results of [1] and [8], Tavenas [14] proved the following theorem.

▶ **Theorem 7** (Theorem 4, [14]). *Let f be an N-variate polynomial computed by a circuit of size s and of degree d. Then f is computed by a $\Sigma\Pi^{[D]}\Sigma\Pi^{[t]}$ circuit C of size $2^{O(\sqrt{d\log(ds)\log N})}$. Furthermore, if f is homogenous, it will also the case for C.*

Following Tavenas' proof, one can choose $D = 15\sqrt{d}$ and $t = \sqrt{d}$. As a consequence, we infer that any $n^{O(1)}$-variate polynomial of degree n in VP can be computed by a $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuit of size $2^{O(\sqrt{n}\log n)}$.

For a monomial $\mathbf{x}^{\mathbf{i}} = x_1^{i_1}x_2^{i_2}\dots x_n^{i_n}$, let $\partial^{\mathbf{i}}f$ be the partial derivative of f with respect to the monomial $\mathbf{x}^{\mathbf{i}}$. The degree of the monomial is denoted by $|\mathbf{i}|$ where $|\mathbf{i}| = (i_1 + i_2 + \cdots + i_n)$. We recall the following definition of shifted partial derivatives from [5].

▶ **Definition 8.** Let $f(\mathrm{X}) \in \mathbb{F}[\mathrm{X}]$ be a multivariate polynomial. The span of the ℓ-shifted k-th order derivatives of f, denoted by $\langle\partial^{=k}f\rangle_{\leq\ell}$, is defined as

$$\langle\partial^{=k}f\rangle_{\leq\ell} = \mathbb{F}\text{-span}\{\mathbf{x}^{\mathbf{i}}\cdot(\partial^{\mathbf{j}}f) : \mathbf{i},\mathbf{j}\in\mathbb{Z}^n_{\geq 0} \text{ with } |\mathbf{i}|\leq\ell \text{ and } |\mathbf{j}| = k\}\,.$$

We denote by $\dim(\langle\partial^{=k}f\rangle_{\leq\ell})$ the dimension of the vector space $\langle\partial^{=k}f\rangle_{\leq\ell}$.

Let $\succ$ be any admissible monomial ordering. The *leading monomial* of a polynomial $f(\mathrm{X}) \in \mathbb{F}[\mathrm{X}]$, denoted by $\mathsf{LM}(f)$ is the largest monomial $\mathbf{x}^{\mathbf{i}} \in f(\mathrm{X})$ under the order $\succ$. The next lemma follows directly from Proposition 11 and Corollary 12 of [5].

▶ **Lemma 9.** *For any multivariate polynomial $f(\mathrm{X}) \in \mathbb{F}[\mathrm{X}]$,*

$$\dim(\langle\partial^{=k}f\rangle_{\leq\ell}) \geq \#\{\mathbf{x}^{\mathbf{i}}\cdot\mathsf{LM}(g) : \mathbf{i},\mathbf{j}\in\mathbb{Z}^n_{\geq 0} \textit{ with } |\mathbf{i}|\leq\ell, |\mathbf{j}| = k, \textit{ and } g\in\mathbb{F}\textit{-span}\{\partial^{\mathbf{j}}f\}\}\,.$$

In [7], the following upper bound on the dimension of the shifted partial derivative space for polynomials computed by $\Sigma\Pi^{[D]}\Sigma\Pi^{[t]}$ circuits was shown. This bound was implicit in the work of Gupta et al. [5].

▶ **Lemma 10** (Lemma 4, [7]). *If $C = \sum_{i=1}^{s'} Q_{i1}Q_{i2}\dots Q_{iD}$ where each $Q_{ij} \in \mathbb{F}[\mathrm{X}_N]$ is a polynomial of degree bounded by t. Then for any $k \leq D$,*

$$\dim(\langle\partial^{=k}(C)\rangle_{\leq\ell}) \leq s'\binom{D}{k}\binom{N+\ell+k(t-1)}{N}\,.$$

4 Unified analysis of depth-4 lower bounds

In this section, we first prove a simple combinatorial lemma which we believe is the crux of the best known depth-4 lower bound results. In fact, the lower bounds on the size of $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuits computing the polynomials $\mathrm{NW}_{n,\epsilon}(\mathrm{X})$ and $\mathrm{IMM}_{n,n}(\mathrm{X})$ follow easily from this lemma by suitable setting of the parameters.

▶ **Lemma 11.** *Let $m_1, m_2, \dots, m_s$ be the monomials over N variables s.t. $\mathrm{dist}(m_i, m_j) \geq \Delta$ for all $i \neq j$. Let M be the set of monomials of the form $m_i m'$ where $1 \leq i \leq s$ and m' is a monomial of length at most ℓ over the same set of N variables. Then, the cardinality of M is at least $\left(sB - s^2\binom{N+\ell-\Delta}{N}\right)$ where $B = \binom{N+\ell}{N}$.*

Proof. Let B_i be the set of all monomials $m_i m'$ where m' is a monomial of length at most ℓ. It is easy to see that $|B_i| = \binom{N+\ell}{N}$. We would like to estimate $|\cup_i B_i|$. Using the principle of inclusion and exclusion, we get $|\cup_{i=1}^{s} B_i| \geq \sum_{i\in[s]}|B_i| - \sum_{i,j\in[s],i\neq j}|B_i\cap B_j|$.

Now we estimate the upper bound for $|B_i \cap B_j|$ such that $i \neq j$. Consider the monomials M_i and M_j in B_i and B_j respectively. For M_i and M_j to match, M_i should contain at least Δ variables from m_j and similarly M_j should contain at least Δ variables from m_i. The rest of the at most $(\ell - \Delta)$ degree monomials should be identical in M_i and M_j. The number of such monomials over N variables is at most $\binom{N+\ell-\Delta}{N}$. Thus, $|B_i \cap B_j| \leq \binom{N+\ell-\Delta}{N}$.

Then the total number of monomials of the form $m_i m'$ for all $i \in [s]$ where m' is a monomial of length at most ℓ is lower bounded as follows:

$$|\cup_{i=1}^{s} B_i| \geq sB - s^2 \binom{N+\ell-\Delta}{N} = sB\left(1 - \frac{s}{B}\binom{N+\ell-\Delta}{N}\right).$$

◀

We use the above lemma to prove the main theorem of this section (restated from Section 1).

▶ **Theorem 12.** *Let $f(\mathrm{X})$ be a $n^{O(1)}$-variate polynomial of degree n. Let there be at least $n^{\delta k}$ (δ is any constant > 0) different polynomials in $\langle \partial^{=k}(f)\rangle$ for $k = \epsilon\sqrt{n}$ such that any two of their leading monomials have a distance of at least $\Delta \geq \frac{n}{c}$ for any constant $c > 1$, and $0 < \epsilon < \frac{1}{40c}$. Then any depth-4 $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuit that computes $f(\mathrm{X})$ must be of size $e^{\Omega_{\delta,c}(\sqrt{n}\ln n)}$.*

Proof. Consider a set of $s = n^{\delta k}$ polynomials $f_1, f_2, \ldots, f_s \in \langle \partial^{=k}(f)\rangle$ such that $\mathrm{dist}(\mathsf{LM}(f_i), \mathsf{LM}(f_j)) \geq n/c$ for all $i \neq j$. We denote by m_i, the leading monomial $\mathsf{LM}(f_i)$.

We now invoke Lemma 11 with the parameters $s = n^{\delta k}, \Delta = n/c$. Let N be the number of variables in f. From Lemma 11, we know that $|\cup_{i=1}^{s} B_i| \geq sB\left(1 - \frac{s}{B}\binom{N+\ell-\Delta}{N}\right)$. To get a good lower bound for $|\cup_{i=1}^{s} B_i|$, we need to upper bound $\frac{s}{B}\binom{N+\ell-\Delta}{N}$. Let us bound it by an inverse polynomial in n by suitably choosing ℓ. We set $\frac{s\binom{N+\ell-d}{N}}{\binom{N+\ell}{N}} \leq \frac{1}{p(n)}$ where $p(n)$ is a polynomial in n.

After simplification, we get $s\frac{(N+\ell-\Delta)!}{(N+\ell)!}\frac{\ell!}{(\ell-\Delta)!} \leq \frac{1}{p(n)}$. Using Lemma 6 we tightly estimate the subsequent computations. In particular, we always choose the parameter ℓ such that $\Delta^2 = o(N+\ell)$. This also shows that the error term given by Lemma 6 is always asymptotically zero and we need not worry about it.

We now apply Lemma 6 to derive $s\left(\frac{\ell}{N+\ell}\right)^{\Delta} \leq \frac{1}{p(n)}$ or equivalently $s\left(\frac{1}{1+\frac{N}{\ell}}\right)^{\Delta} \leq \frac{1}{p(n)}$. We use the inequality $1 + x > e^{x/2}$ for $0 < x < 1$ to lower bound $\left(1 + \frac{N}{\ell}\right)^{\Delta}$ by $e^{\frac{N\Delta}{2l}}$. Thus, it is enough to choose ℓ in a way that $s \cdot p(n) \leq e^{\frac{N\Delta}{2\ell}}$ or equivalently $\ell \leq \frac{N\Delta}{2\ln(s\cdot p(n))}$. By fixing $p(n) = n^2$ and substituting for the parameters k and Δ , we get $\ell \leq \frac{N\sqrt{n}}{4c\delta\epsilon\ln n}$. From Lemma 9, we get that the dimension of $\langle \partial^{=k} f\rangle_{\leq \ell} \geq (1 - \frac{1}{n^2})\ s\binom{N+\ell}{N}$.

Combining this with Lemma 10, we get $s' \geq \frac{\left(1-\frac{1}{n^2}\right)s\binom{N+l}{N}}{\binom{D}{k}\binom{N+l+k(t-1)}{N}}$. Suppose we choose ℓ such that $(kt-k)^2 = o(\ell)$. Then, by applying Lemma 6 we can easily show the following:

$$s' \geq \frac{s\left(1-\frac{1}{n^2}\right)}{\binom{D}{k}(1+\frac{N}{l})^{(kt-k)}} \geq \frac{n^{\delta k}\left(1-\frac{1}{n^2}\right)}{\binom{D}{k}e^{\frac{N}{\ell}kt}}.$$

Since $D = O(\sqrt{n})$ and $k = \epsilon\sqrt{n}$, we can estimate $\binom{D}{k}$ to be $e^{O_\epsilon(\sqrt{n})}$ by Shannon's entropy estimate for binomial coefficients. To get the required lower bound it is sufficient to choose ℓ such that $\frac{Nkt}{\ell} < (0.1)\delta k\ln n$. Since $t \leq \sqrt{n}$, it is enough to choose $\ell > \frac{10N\sqrt{n}}{\delta\ln n}$. By comparing the lower and upper bounds of ℓ, we can fix ϵ such that $\epsilon < \frac{1}{40c}$. Since ϵ depends only on c, we can infer that $s' = e^{\Omega_{\delta,c}(\sqrt{n}\ln n)}$. ◀

The above proof clearly goes through even if we set $\frac{Nkt}{\ell} < \mu\delta k\ln n$ for any $0 < \mu < 1$, and choose $\epsilon < \frac{\mu}{4c}$. But for simplicity, we prefer to state Theorem 12 in its current form.

In the next section, we show that the lower bounds on the size of $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuits computing $\mathrm{NW}_{n,\epsilon}(\mathrm{X})$ and $\mathrm{IMM}_{n,n}(\mathrm{X})$ can be obtained by simply applying Theorem 12.

Moreover, it shows that the lower bound arguments of $\text{IMM}_{n,n}(\text{X})$ are essentially same as the lower bound arguments of $\text{NW}_{n,\epsilon}(\text{X})$.

4.1 Lower bounds on the size of depth-4 circuits computing $\text{NW}_{n,\epsilon}(\text{X})$ and $\text{IMM}_{n,n}(\text{X})$

Now we derive the depth-4 circuit size lower bound for $\text{NW}_{n,\epsilon}(\text{X})$ polynomial by a simple application of Theorem 12.

▶ **Corollary 13.** *For $0 < \epsilon < 1/80$, any depth-4 $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuit computing the polynomial $\text{NW}_{n,\epsilon}(\text{X})$ must be of size $2^{\Omega(\sqrt{n}\log n)}$.*

Proof. Recall that $\text{NW}_{n,\epsilon}(\text{X}) = \sum_{a(z)\in\mathbb{F}[z]} x_{1a(1)}x_{2a(2)}\dots x_{na(n)}$ where $\mathbb{F}$ is a finite field of size n and $a(z)$ is a univariate polynomial of degree $\leq k-1$ where $k = \epsilon\sqrt{n}$. Notice that any two monomials can intersect in at most $k-1$ variables.

We differentiate the polynomial $\text{NW}_{n,\epsilon}(\text{X})$ with respect to the first $k = \epsilon\sqrt{n}$ variables of each monomial. After differentiation, we get n^k monomials of length $(n-k)$ each. Since they are constructed from the image of univariate polynomials of degree at most $(k-1)$, the distance Δ between any two monomials $\geq n-2k > n/2$. So to get the required lower bound we invoke Theorem 12 with $\delta = 1$ and $c = 2$. ◀

Next we derive the lower bound on the size of the depth-4 circuit computing $\text{IMM}_{n,n}(\text{X})$.

▶ **Corollary 14.** *Any depth-4 $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuit computing the $\text{IMM}_{n,n}(\text{X})$ polynomial must be of size $2^{\Omega(\sqrt{n}\log n)}$.*

Proof. Recall that $\text{IMM}_{n,n}(\text{X}) = \sum_{i_1,i_2,\dots,i_{n-1}\in[n]} x^{(1)}_{1i_1}x^{(2)}_{i_1i_2}\dots x^{(n-1)}_{i_{(n-2)}i_{(n-1)}}x^{(n)}_{i_{(n-1)}1}$. It is a polynomial over $(n-2)n^2+2n$ variables. We fix the following lexicographic ordering on the variables of the set of matrices $\{\text{X}^{(1)},\text{X}^{(2)},\dots,\text{X}^{(n)}\}$ as follows: $\text{X}^{(1)} \succ \text{X}^{(2)} \succ \text{X}^{(3)} \succ \dots \succ \text{X}^{(n)}$ and in any $\text{X}^{(i)}$ the ordering is $x^{(i)}_{11} \succ x^{(i)}_{12} \succ \dots \succ x^{(i)}_{1n} \succ \dots \succ x^{(i)}_{n1} \dots \succ x^{(i)}_{nn}$.

Choose a prime p such that $\frac{n}{2} \leq p \leq n$. Consider the set of univariate polynomials $a(z) \in \mathbb{F}_p[z]$ of degree at most $(k-1)$ for $k = \epsilon\sqrt{n}$ where ϵ is a small constant to be fixed later in the analysis.

Consider a set of $2k$ of the matrices $\text{X}^{(2)},\text{X}^{(3+\frac{n}{4k})},\dots,\text{X}^{(2k+1+\frac{(2k-1)n}{4k})}$ such that they are $n/4k$ distance apart. Clearly $2k+1+\frac{(2k-1)n}{4k} < n$. For each univariate polynomial a of degree at most $(k-1)$, define a set $S_a = \{x^{(2)}_{1,a(1)}, x^{(3+\frac{n}{4k})}_{2,a(2)},\dots,x^{(2k+1+\frac{(2k-1)n}{4k})}_{2k,a(2k)}\}$. Number of such sets is at least $\left(\frac{n}{2}\right)^k$ and $|S_a \cap S_b| < k$ for $a \neq b$. Now we consider a polynomial $f(\text{X})$ which is a restriction of the polynomial $\text{IMM}_{n,n}(\text{X})$. By restriction, we simply mean that a few variables of $\text{IMM}_{n,n}(\text{X})$ are fixed to some elements from the field and the rest of the variables are left untouched. We define the restriction as follows:

$$x^{(q)}_{ij} = 0 \text{ if } r + \frac{(r-2)n}{4k} < q < (r+1) + \frac{(r-1)n}{4k} - 1 \text{ for } 2 \leq r \leq 2k \text{ and } i \neq j.$$

The rest of the variables are left untouched.

Next we differentiate the polynomial $f(\text{X})$ with respect to the sets of variables S_a indexed by the polynomials $a(z) \in \mathbb{F}[z]$. Consider the leading monomial of the derivatives with respect to the sets S_a for all $a(z) \in \mathbb{F}[z]$. Since $|S_a \cap S_b| < k$, it is straightforward to observe that the distance between any two leading monomials is at least $k \cdot \frac{n}{4k} = \frac{n}{4}$. The intuitive justification is that whenever there is a difference in S_a and S_b, that difference can be stretched to a distance $\frac{n}{4k}$ because of the restriction that eliminates the non diagonal entries.

Now we prove the lower bound for the polynomial $f(\mathrm{X})$ by applying Theorem 12. Notice that $f(\mathrm{X})$ is a $n^{O(1)}$-variate polynomial of degree n such that there are at least $(n/2)^k > n^{\frac{1}{4}(2k)}$ different polynomials in $\langle \partial^{=2k}(f) \rangle$ such that any two of their leading monomials have distance $\Delta \geq n/4$. So we set the parameters $\delta = 1/4$ and $c = 4$ in Theorem 12. A simple calculation shows that the parameter ϵ can be fixed to something $< 1/320$.

Since $f(\mathrm{X})$ is a restriction of $\mathrm{IMM}_{n,n}(\mathrm{X})$, any lower bound for $f(\mathrm{X})$ is a lower bound for $\mathrm{IMM}_{n,n}(\mathrm{X})$ too. Otherwise, if $\mathrm{IMM}_{n,n}(\mathrm{X})$ has a $2^{o(\sqrt{n}\log n)}$ sized $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuit, then we get a $2^{o(\sqrt{n}\log n)}$ sized $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuit for $f(\mathrm{X})$ by substituting for the variables according to the restriction. ◀

5 Determinantal complexity of $\mathrm{IMM}_{n,d}(\mathrm{X})$

We start by recalling a few facts from [3]. Let $A_{k,\ell}(\mathrm{X})$, $1 \leq k, \ell \leq m$ be the affine linear functions over $\mathbb{F}[\mathrm{X}]$ such that the following is true:

$$\mathrm{IMM}_{n,d}(\mathrm{X}) = \det((A_{k,\ell}(\mathrm{X}))_{1\leq k,\ell\leq m}) .$$

Consider a point $\mathrm{X}_0 \in \mathbb{F}^{n^2 d}$ such that $\mathrm{IMM}_{n,d}(\mathrm{X}_0) = 0$. The affine linear functions $A_{k,\ell}(\mathrm{X})$ can be expressed as $L_{k,\ell}(\mathrm{X} - \mathrm{X}_0) + y_{k,\ell}$ where $L_{k,\ell}$ is a linear form and $y_{k,\ell}$ is a constant from the field. Thus, $(A_{k,\ell}(\mathrm{X}))_{1\leq k,\ell\leq m} = (L_{k,\ell}(\mathrm{X} - \mathrm{X}_0))_{1\leq k,\ell\leq m} + \mathrm{Y}_0$. If $\mathrm{IMM}_{n,d}(\mathrm{X}_0) = 0$ then $\det(\mathrm{Y}_0) = 0$. Let C and D be two non-singular matrices such that $\mathrm{CY}_0\mathrm{D}$ is a diagonal matrix:

$$\mathrm{CY}_0\mathrm{D} = \begin{pmatrix} 0 & 0 \\ 0 & \mathrm{I}_s \end{pmatrix} .$$

Since $\det(\mathrm{Y}_0) = 0$, $s < m$. From the previous works [17], [2], [11], and [3], it is enough to assume that $s = m - 1$. Since the first row and the first column of $\mathrm{CY}_0\mathrm{D}$ are zero, we may multiply $\mathrm{CY}_0\mathrm{D}$ by $\mathrm{diag}(\det(\mathrm{C})^{-1}, 1, \ldots, 1)$ and $\mathrm{diag}(\det(\mathrm{D})^{-1}, 1, \ldots, 1)$ on the left and the right side. Without loss of generality, we may assume that $\det(\mathrm{C}) = \det(\mathrm{D}) = 1$. By multiplying with C and D on the left and the right and suitably renaming $(L_{k,\ell}(\mathrm{X} - \mathrm{X}_0))_{1\leq k,\ell\leq m}$ and Y_0 we get

$$\mathrm{IMM}_{n,d}(\mathrm{X}) = \det((L_{k,\ell}(\mathrm{X} - \mathrm{X}_0)_{1\leq k,\ell\leq m} + \mathrm{Y}_0))$$

where $\mathrm{Y}_0 = \mathrm{diag}(0, 1, \ldots, 1)$.

We use $\mathrm{H}_{\mathrm{IMM}_{n,d}}(\mathrm{X})$ to denote the Hessian matrix of the iterated matrix multiplication and is defined as follows:

$$\mathrm{H}_{\mathrm{IMM}_{n,d}}(\mathrm{X}) = (H_{s;ij,t;k\ell}(\mathrm{X}))_{1\leq i,j\leq n, 1\leq s,t\leq d}$$

$$H_{s;ij,t;k\ell}(\mathrm{X}) = \frac{\partial^2 \mathrm{IMM}_{n,d}(\mathrm{X})}{\partial x_{ij}^{(s)} \partial x_{k\ell}^{(t)}}$$

where $x_{ij}^{(s)}$ and $x_{k\ell}^{(t)}$ denote the (i,j)th and (k,ℓ)th entries of the variable sets $\mathrm{X}^{(s)}$ and $\mathrm{X}^{(t)}$ respectively.

By taking second order derivatives and evaluating the Hessian matrices of $\mathrm{IMM}_{n,d}(\mathrm{X})$ and $\det((A_{k,\ell}(\mathrm{X}))_{1\leq k,\ell\leq m})$ at X_0, we obtain $\mathrm{H}_{\mathrm{IMM}_{n,d}}(\mathrm{X}_0) = \mathrm{L}\mathrm{H}_{\det}(\mathrm{Y}_0)\mathrm{L}^T$ where L is a $n^2 d \times m^2$ matrix with entries from the field. It follows that $\mathrm{rank}(\mathrm{H}_{\mathrm{IMM}_{n,d}}(\mathrm{X}_0)) \leq \mathrm{rank}(\mathrm{H}_{\det}(\mathrm{Y}_0))$. It was observed in the earlier work of [11] and [3] that it is relatively easy to get an upper bound for $\mathrm{rank}(\mathrm{H}_{\det}(\mathrm{Y}_0))$. The main task is to construct a point X_0 such that $\mathrm{IMM}_{n,d}(\mathrm{X}_0) = 0$, yet the rank of $\mathrm{H}_{\mathrm{IMM}_{n,d}}(\mathrm{X}_0)$ is high. We give an explicit construction of a point $\mathrm{X}_0 \in \mathbb{F}^{n^2 d}$ such that $\mathrm{IMM}_{n,d}(\mathrm{X}_0) = 0$ and $\mathrm{rank}(\mathrm{H}_{\mathrm{IMM}_{n,d}}(\mathrm{X}_0)) \geq d(n-1)$. First for the sake of completeness, we briefly recall the upper bound argument for the rank of $\mathrm{H}_{\det}(\mathrm{Y}_0)$ from Section 2.1 of [3].

5.1 Upper bound for the rank of $\mathrm{H}_{\det}(\mathrm{Y}_0)$

When we take a partial derivative $\frac{\partial}{\partial x_{ij}}$ of the determinant, we get the minor after striking out the row i and column j. The second order derivative of $\det(\mathrm{Y})$ with respect to the variables y_{ij} and $y_{k\ell}$ eliminates the rows $\{i,k\}$ and the columns $\{j,\ell\}$. Considering the form of Y_0, the non-zero entries in $\mathrm{H}_{\det}(\mathrm{Y}_0)$ are obtained only if $1 \in \{i,k\}$ and $1 \in \{j,\ell\}$ and thus $(ij,k\ell)$ are of the form $(11,tt)$ or $(t1,1t)$ or $(1t,t1)$ for any $t > 1$. Thus, $\mathrm{rank}(\mathrm{H}_{\det}(\mathrm{Y}_0)) = O(m)$.

5.2 Lower bound for the rank of $\mathrm{H}_{\mathrm{IMM}_{n,d}}(\mathrm{X}_0)$

In this section, we prove Theorem 5. In particular, we give a polynomial time algorithm to construct a point X_0 explicitly such that $\mathrm{IMM}_{n,d}(\mathrm{X}_0) = 0$ and $\mathrm{rank}(\mathrm{H}_{\mathrm{IMM}_{n,d}}(\mathrm{X}_0)) \geq d(n-1)$. Since $\mathrm{rank}(\mathrm{H}_{\det}(\mathrm{Y}_0)) = O(m)$ and $\mathrm{rank}(\mathrm{H}_{\mathrm{IMM}_{n,d}}(\mathrm{X}_0)) \leq \mathrm{rank}(\mathrm{H}_{\det}(\mathrm{Y}_0))$, we get that $m = \Omega(dn)$. As mentioned in the section 1, the determinantal complexity of $\mathrm{IMM}_{n,d}(\mathrm{X})$ is $O(dn)$. Together, it implies that $m = \Theta(dn)$.

▶ **Theorem 15.** *For any integers $n, d > 1$, there is a point $\mathrm{X}_0 \in \mathbb{F}^{n^2 d}$ such that $\mathrm{IMM}_{n,d}(\mathrm{X}_0) = 0$ and $\mathrm{rank}(\mathrm{H}_{\mathrm{IMM}_{n,d}}(\mathrm{X}_0)) \geq d(n-1)$. Moreover, the point X_0 can be constructed explicitly in polynomial time.*

Proof. We prove the theorem by induction on d. For the purpose of induction, we maintain that the entries indexed by the indices $(1,2),(1,3),\ldots,(1,n)$ of the matrix obtained after multiplying the first $(d-1)$ matrices are not all zero at X_0.

We first prove the base case for $d = 2$. The corresponding polynomial is $\mathrm{IMM}_{n,2}(\mathrm{X}) = \sum_{i=1}^{n} x_{1i}^{(1)} x_{i1}^{(2)}$. It is easy to observe that the rank of the Hessian matrix is $2n > 2(n-1)$ at any point since each non-zero entry of the Hessian matrix is 1 and the structure of the Hessian matrix is the following:

$$\mathrm{H}_{\mathrm{IMM}_{n,2}}(\mathrm{X}) = \begin{bmatrix} 0 & B_{12} \\ B_{21} & 0 \end{bmatrix}$$

where $B_{21} = B_{12}^T$. The matrix B_{12} is formally described as follows.

$$(B_{12})_{x_{ij}^{(1)} x_{kl}^{(2)}} = \begin{cases} 1 & \text{if } i = l = 1 \text{ and } j = k \\ 0 & \text{otherwise.} \end{cases}$$

We set the values of the variables as follows: $x_{11}^{(1)} = 0$, $x_{11}^{(2)} = 1$, $x_{21}^{(2)} = x_{31}^{(2)} = \cdots = x_{n1}^{(2)} = 0$ and $x_{12}^{(1)}, x_{13}^{(1)}, \ldots, x_{1n}^{(1)}$ arbitrarily but not all to zero. The point thus obtained (say X_0) is clearly a zero of the polynomial $\mathrm{IMM}_{n,2}(\mathrm{X})$.

For induction hypothesis, assume that the statement of the theorem is true for the case where the number of matrices being multiplied is $\leq d$. Consider the polynomial $\mathrm{IMM}_{n,(d+1)}(\mathrm{X})$:

$$\mathrm{IMM}_{n,(d+1)}(\mathrm{X}) = \sum_{i_1,i_2,\ldots,i_{d-1},i_d \in [n]} x_{1i_1}^{(1)} x_{i_1 i_2}^{(2)} \ldots x_{i_{(d-2)} i_{(d-1)}}^{(d-1)} x_{i_{(d-1)} i_d}^{(d)} x_{i_d 1}^{(d+1)}.$$

Let the matrix obtained after multiplying the first d matrices be the following:

$$\begin{bmatrix} P_{11}(\mathrm{X}) & P_{12}(\mathrm{X}) & \cdots & P_{1n}(\mathrm{X}) \\ P_{21}(\mathrm{X}) & P_{22}(\mathrm{X}) & \cdots & P_{2n}(\mathrm{X}) \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1}(\mathrm{X}) & P_{n2}(\mathrm{X}) & \cdots & P_{nn}(\mathrm{X}) \end{bmatrix}$$

where

$$P_{k\ell}(\mathrm{X}) = \sum_{i_1,i_2,\ldots,i_{d-1}\in[n]} x^{(1)}_{ki_1} x^{(2)}_{i_1i_2} \ldots x^{(d-1)}_{i_{(d-2)}i_{(d-1)}} x^{(d)}_{i_{(d-1)}\ell} \text{ for } 1 \le k,l \le n\,.$$

Thus, we have the following expression:

$$\mathrm{IMM}_{n,(d+1)}(\mathrm{X}) = P_{11}(\mathrm{X})x^{(d+1)}_{11} + P_{12}(\mathrm{X})x^{(d+1)}_{21} + \cdots + P_{1n}(\mathrm{X})x^{(d+1)}_{n1}\,.$$

Now consider the Hessian matrix $\mathrm{H}_{\mathrm{IMM}_{n,d+1}}(\mathrm{X})$ which is a $(d+1)n^2 \times (d+1)n^2$ sized matrix:

$$\mathrm{H}_{\mathrm{IMM}_{n,d+1}}(\mathrm{X}) = \begin{bmatrix} 0 & B_{1,2} & B_{1,3} & B_{1,4} & \cdots & B_{1,(d+1)} \\ B_{2,1} & 0 & B_{2,3} & B_{2,4} & \cdots & B_{2,(d+1)} \\ B_{3,1} & B_{3,2} & 0 & B_{3,4} & \cdots & B_{3,(d+1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ B_{(d+1),1} & B_{(d+1),2} & \cdots & \cdots & B_{(d+1),d} & 0 \end{bmatrix}.$$

Each $B_{i,j}$ is a block of size $n^2 \times n^2$ which is indexed by the variables from the matrices $M^{(i)}$ and $M^{(j)}$ with the corresponding variable sets $\mathrm{X}^{(i)}$ and $\mathrm{X}^{(j)}$. Consider the block $B_{(d+1),d}$ which is indexed by the variable sets $\mathrm{X}^{(d+1)}$ and $\mathrm{X}^{(d)}$. The only non-zero rows in $B_{(d+1),d}$ are indexed by the variables $x^{(d+1)}_{11}, x^{(d+1)}_{21}, \ldots, x^{(d+1)}_{n1}$. The potential non-zero entries for the row $x^{(d+1)}_{11}$ are indexed by the columns $x^{(d)}_{11}, x^{(d)}_{21}, \ldots, x^{(d)}_{n1}$. Similarly the potential non-zero entries for the row $x^{(d+1)}_{21}$ are indexed by the columns $x^{(d)}_{12}, x^{(d)}_{22}, \ldots, x^{(d)}_{n2}$ and so on.

Consider the entries indexed by the indices $(x^{(d+1)}_{11}, x^{(d)}_{11}), (x^{(d+1)}_{11}, x^{(d)}_{21}), \ldots, (x^{(d+1)}_{11}, x^{(d)}_{n1})$. They are $s_1, s_2, \ldots, s_n$ respectively and they can be expressed as follows:

$$s_j = \sum_{i_1,i_2,\ldots,i_{d-2}\in[n]} x^{(1)}_{1i_1} x^{(2)}_{i_1i_2} \ldots x^{(d-1)}_{i_{(d-2)}j} \text{ for } 1 \le j \le n.$$

For the other rows indexed by the variables $x^{(d+1)}_{21}, x^{(d+1)}_{31}, \ldots, x^{(d+1)}_{n1}$, the sequence of potential non-zero entries is the same $(s_1, s_2, \ldots, s_n)$ but their positions are shifted by a column compared to the previous non-zero row. Formally, we have the following:

$$(B_{(d+1),d})_{x^{(d+1)}_{ij} x^{(d)}_{kl}} = \begin{cases} s_k & \text{if } j = 1,\ l = i, \text{ and } i,k \in [n] \\ 0 & \text{otherwise.} \end{cases}$$

$s_1, s_2, \ldots, s_n$ are also the entries indexed by the indices $(1,1), (1,2), \ldots, (1,n)$ of the matrix obtained after multiplying the first $(d-1)$ matrices. By induction hypothesis, we know that the entries indexed by the indices $(1,2), \ldots, (1,n)$ are not all zero at the point X_0 which is a zero of the polynomial $\mathrm{IMM}_{n,d}(\mathrm{X})$. This also makes the rows indexed by the variables $x^{(d+1)}_{11}, x^{(d+1)}_{21}, \ldots, x^{(d+1)}_{n1}$ linearly independent. It is important to note that $P_{11}(\mathrm{X}) = \mathrm{IMM}_{n,d}(\mathrm{X})$.

Now, let us define a point such that it is a zero of the polynomial $\mathrm{IMM}_{n,(d+1)}(\mathrm{X})$. Let X_0 be the zero of the polynomial $P_{11}(\mathrm{X}) = \mathrm{IMM}_{n,d}(\mathrm{X})$. Now to construct the new point, we inductively fix the variables appearing in $P_{11}(\mathrm{X})$ by the values assigned by X_0. We set $x^{(d+1)}_{11} = 1$ and $x^{(d+1)}_{21} = x^{(d+1)}_{31} = \cdots = x^{(d+1)}_{n1} = 0$. We will fix the rest of the variables later. We call the new point which is a zero of the polynomial $\mathrm{IMM}_{n,(d+1)}(\mathrm{X})$, as X_0 as well.

Now, consider the first $d \times d$ blocks of the Hessian matrix $\mathrm{H}_{\mathrm{IMM}_{n,(d+1)}}(\mathrm{X}_0)$. It precisely represents the Hessian matrix of $P_{11}(\mathrm{X})$ which is also the Hessian matrix of the polynomial $\mathrm{IMM}_{n,d}(\mathrm{X})$ at the point X_0[1]. By induction hypothesis, the rank of this minor of $\mathrm{H}_{\mathrm{IMM}_{n,(d+1)}}(\mathrm{X}_0)$ is at least $d(n-1)$. The only non-zero entries in the columns indexed by the variable set $\mathrm{X}^{(d)}$ are indexed by the variables $x_{11}^{(d)}, x_{21}^{(d)}, \ldots, x_{n1}^{(d)}$. This is because the other variables of $\mathrm{X}^{(d)}$ do not appear in $\mathrm{IMM}_{n,d}(\mathrm{X})$. The row in $B_{(d+1)d}$ indexed by $x_{11}^{(d+1)}$ is the only row that interferes with any of the rows of $B_{1d}, B_{2d}, \ldots, B_{dd}$. The rows indexed by the variables $x_{21}^{(d+1)}, x_{31}^{(d+1)}, \ldots, x_{n1}^{(d+1)}$ in $B_{(d+1)d}$ are linearly independent of the rows of $B_{1d}, B_{2d}, \ldots, B_{dd}$. Hence the rank of $\mathrm{H}_{\mathrm{IMM}_{n,(d+1)}}$ at the point described is $\geq (d+1)(n-1)$.

For the purpose of induction, we must verify that the entries indexed by the indices $(1,2), (1,3), \ldots, (1,n)$ of the matrix obtained after multiplying the first d matrices are not all zero at X_0. These entries are the polynomials $P_{12}, P_{13}, \ldots, P_{1n}$. We shall express each of the polynomials in terms of $s_1, s_2, \ldots, s_n$ as follows:

$$P_{1j} = s_1 x_{1j}^{(d)} + s_2 x_{2j}^{(d)} + \cdots + s_n x_{nj}^{(d)} \text{ for } 2 \leq j \leq n.$$

By induction hypothesis, we already know that $s_2, s_3, \ldots, s_n$ are not all zero at X_0. Notice that the variables in $\mathrm{X}^{(d)} \setminus \{x_{11}^{(d)}, x_{21}^{(d)}, \ldots, x_{n1}^{(d)}\}$ were never set in the previous steps of induction[2]. Therefore, we can fix these variables suitably such that $P_{12}, P_{13}, \ldots, P_{1n}$ are not all zero when evaluated at the point X_0 (in fact, we can make all of them non-zero). It is clear that we construct the point X_0 in polynomial time. This completes the proof. ◀

6 Open Problems

In [5] it was proved that any $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuit for computing the determinant or the permanent polynomial of a $n \times n$ matrix must be of size $2^{\Omega(\sqrt{n})}$. A natural question is to ask whether one can improve the lower bound to $2^{\Omega(\sqrt{n}\log n)}$. It is unclear whether the leading monomial distance property can be applied directly to Determinant or Permanent to prove such a result. We suspect that it will require a new idea.

▶ **Problem 16.** *Prove that any $\Sigma\Pi^{[O(\sqrt{n})]}\Sigma\Pi^{[\sqrt{n}]}$ circuit computing Determinant or Permanent of a $n \times n$ matrix must be of size $2^{\Omega(\sqrt{n}\log n)}$.*

We do not have a good understanding of the determinantal complexity of the $\mathrm{NW}_{n,\epsilon}(\mathrm{X})$ polynomial. In particular, we would like to pose the following problem.

▶ **Problem 17.** *Prove that the determinantal complexity of the $\mathrm{NW}_{n,\epsilon}(\mathrm{X})$ polynomial is $\Omega_\epsilon(n^2)$.*

Acknowledgement. We are grateful to the anonymous STACS 2014 referees for their comments and suggestions.

[1] This can be easily seen from the setting of the variables $x_{11}^{(d+1)} = 1$ and $x_{21}^{(d+1)} = x_{31}^{(d+1)} = \cdots = x_{n1}^{(d+1)} = 0$.
[2] Because they do not appear in the polynomial P_{11}.

References

1 Manindra Agrawal and V Vinay. Arithmetic circuits: A chasm at depth four. In *Proceedings-Annual Symposium on Foundations of Computer Science*, pages 67–75. IEEE, 2008.

2 Jin-Yi Cai. A note on the determinant and permanent problem. *Information and Computation*, 84(1):119–127, 1990.

3 Jin-Yi Cai, Xi Chen, and Dong Li. A quadratic lower bound for the permanent and determinant problem over any characteristic$\neq$ 2. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 491–498. ACM, 2008.

4 Hervé Fournier, Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower bounds for depth 4 formulas computing iterated matrix multiplication. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:100, 2013.

5 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Approaching the chasm at depth four. In *Proceedings of the Conference on Computational Complexity (CCC)*, 2013.

6 Maurice Jansen. Lower bounds for the determinantal complexity of explicit low degree polynomials. *Theory of Computing Systems*, 49(2):343–354, 2011.

7 Neeraj Kayal, Chandan Saha, and Ramprasad Saptharishi. A super-polynomial lower bound for regular arithmetic formulas. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:91, 2013.

8 Pascal Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theor. Comput. Sci.*, 448:56–65, 2012.

9 Mrinal Kumar and Shubhangi Saraf. The limits of depth reduction for arithmetic formulas: It's all about the top fan-in. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:153, 2013.

10 Roy Meshulam. On two extremal matrix problems. *Linear Algebra and its Applications*, 114:261–271, 1989.

11 Thierry Mignon and Nicolas Ressayre. A quadratic bound for the determinant and permanent problem. *International Mathematics Research Notices*, 2004(79):4241–4253, 2004.

12 Noam Nisan. Lower bounds for non-commutative computation. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 410–418. ACM, 1991.

13 Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

14 Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. In *MFCS*, pages 813–824, 2013.

15 Leslie G Valiant. Completeness classes in algebra. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 249–261. ACM, 1979.

16 Joachim von zur Gathen. Permanent and determinant. In *FOCS*, pages 398–401. IEEE Computer Society, 1986.

17 Joachim von zur Gathen. Permanent and determinant. *Linear Algebra and its Applications*, 96:87–100, 1987.

Constant Factor Approximation for Capacitated k-Center with Outliers*

Marek Cygan and Tomasz Kociumaka

Institute of Informatics, University of Warsaw, Poland
{cygan, kociumaka}@mimuw.edu.pl

Abstract

The k-center problem is a classic facility location problem, where given an edge-weighted graph $G = (V, E)$ one is to find a subset of k vertices S, such that each vertex in V is "close" to some vertex in S. The approximation status of this basic problem is well understood, as a simple 2-approximation algorithm is known to be tight. Consequently different extensions were studied.

In the capacitated version of the problem each vertex is assigned a capacity, which is a strict upper bound on the number of clients a facility can serve, when located at this vertex. A constant factor approximation for the capacitated k-center was obtained last year by Cygan, Hajiaghayi and Khuller [FOCS'12], which was recently improved to a 9-approximation by An, Bhaskara and Svensson [arXiv'13].

In a different generalization of the problem some clients (denoted as outliers) may be disregarded. Here we are additionally given an integer p and the goal is to serve exactly p clients, which the algorithm is free to choose. In 2001 Charikar et al. [SODA'01] presented a 3-approximation for the k-center problem with outliers.

In this paper we consider a common generalization of the two extensions previously studied separately, i.e. we work with the capacitated k-center with outliers. We present the first constant factor approximation algorithm with approximation ratio of 25 even for the case of non-uniform hard capacities.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases approximation algorithms, k-center, capacities, outliers, LP rounding

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.251

1 Introduction

The k-center problem is a classic facility location problem and is defined as follows: given a finite set V and a symmetric distance (cost) function $d : V \times V \to \mathbb{R}_{\geq 0}$ satisfying the triangle inequality, find a subset $S \subseteq V$ of size k such that each vertex in V is "close" to some vertex in S. More formally, once we choose S the objective function to be minimized is $\max_{v \in V} \min_{u \in S} d(v, u)$. The vertices of S are called *centers* or *facilities*. The problem is known to be NP-hard [12]. Approximation algorithms for the k-center problem have been well studied and are known to be optimal [13, 15, 16, 17].

In the capacitated setting, studied for twenty years already, we are additionally given a capacity function $L : V \to \mathbb{Z}_{\geq 0}$ and no more than $L(u)$ vertices (called *clients*) may be assigned to a chosen center at $u \in V$. For the special case when all the capacities are *identical* (denoted as the *uniform* case), a 6-approximation was developed by Khuller and Sussmann [19] improving the previous bound of 10 by Bar-Ilan, Kortsarz and Peleg [4]. In the

* This work is partially supported by Foundation for Polish Science grant HOMING PLUS/2012-6/2.



© Marek Cygan and Tomasz Kociumaka;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 251–262



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



soft capacities version, in contrast to the standard (*hard* capacities), we are allowed to open several facilities in a single location, i.e. the facilities may form a multiset. For the uniform soft capacities version the best known approximation ratio equals 5 [19]. For general hard capacities a constant factor approximation has been obtained only recently [11], somewhat surprisingly by using LP rounding. It was followed by a cleaner and simpler approach of An, Bhaskara and Svensson [1] who gave a 9-approximation algorithm. From the hardness perspective a $(3-\varepsilon)$ lower bound on the approximation ratio is known [9, 11].

Another natural direction in generalizing the problem is an assumption that instead of serving all the clients we are given an integer p and we are to select exactly p clients to serve. The disregarded clients are in the literature called *outliers*. The k-center problem with outliers admits a 3-approximation algorithm, which was obtained by Charikar et al. [8].

In this article we study a common generalization of the two mentioned variants of the k-center problem, i.e. involving both capacities and outliers. In order to simplify our algorithms we work with a slight generalization, the Capacitated k-supplier with Outliers problem, where vertices are either clients or potential facility locations. These vertices may coincide, so that one may have both a client and a potential facility location at the same point, as in k-center. Below we give the formal problem definition.

Capacitated k-supplier with Outliers
Input: Integers $k, p \in \mathbb{Z}_{\geq 0}$, finite sets $\mathcal{C}$ and $\mathcal{F}$, a symmetric distance (cost) function $d : (\mathcal{C} \cup \mathcal{F}) \times (\mathcal{C} \cup \mathcal{F}) \to \mathbb{R}_{\geq 0}$ satisfying the triangle inequality, and a capacity function $L : \mathcal{F} \to \mathbb{Z}_{\geq 0}$
Find: Sets $C \subseteq \mathcal{C}$, $F \subseteq \mathcal{F}$, and a function $\phi : C \to F$ satisfying
- $|C| = p$,
- $|F| = k$,
- $|\phi^{-1}(u)| \leq L(u)$ for each $u \in F$.

Minimize: $\max_{v \in C} d(v, \phi(v))$.

Again, in the *soft* capacities version, F is allowed to be a multiset, and in the *uniform* capacities version, the capacity function L is constant.

Existence of an r-approximation algorithm for Capacitated k-center with Outliers can be shown to be equivalent to existence of an r-approximation algorithm for Capacitated k-supplier with Outliers.[1] Interestingly, such an equivalence is not known to hold if we do not allow outliers: the best known approximation factor for the Capacitated k-supplier is 11 while for the Capacitated k-center it is 9, see [1].

1.1 Our results and organization of the paper

The following is the main result of this paper.

▶ **Theorem 1.** *The* Capacitated k-supplier with Outliers *problem, both in hard and soft capacities version, admits a 25-approximation algorithm. The hard uniform capacities version admits a 23-approximation, and soft uniform capacities – a 13-approximation.*

Note that taking $\mathcal{C} = \mathcal{F} = V$ shows that the k-supplier problem generalizes the k-center problem, and consequently gives the same approximation bounds for the latter.

▶ **Corollary 2.** *The* Capacitated k-center with Outliers *problem, both in hard and soft capacities version, admits a 25-approximation algorithm. The hard uniform capacities version admits a 23-approximation, and soft uniform capacities – a 13-approximation.*

[1] The proof is left for the full version of the paper.

It is worth noting, that the already known approximation algorithm for the k-center problem with outliers relies on the fact that a single vertex can serve all the clients that are its neighbors, i.e. there are no capacity constraints. At the same time the previous approximation algorithms for the capacitated k-center problem (both in the uniform and non-uniform case) heavily used the fact that each vertex of the graph is close to some center in any solution. For this reason it was possible to create a path-like [11] or tree-like [1] structure with integrally opened non-leaf vertices, that was the crux in the rounding process. Consequently none of the algorithms for the two previously independently studied extensions of the basic problem, i.e. capacities and outliers, works for the problem we are interested in.

The first step of our algorithm (Section 3) is the standard thresholding technique, where we reduce a general metric to a distance metric of an unweighted graph. In Section 4 we introduce our main conceptual contribution, i.e. the notion of a *skeleton*. A skeleton is a set S of vertices, for which there exists an optimum solution $F \subseteq \mathcal{F}$, such that each vertex of S can be injectively mapped to a nearby vertex of F and moreover each vertex of F is close to some vertex of S. Intuitively a skeleton is not yet a solution, but it looks similar to at least one optimum solution. If no outliers are allowed, any inclusionwise maximal subset of $\mathcal{F}$ with vertices far enough from each other, is a skeleton. In [11] and [1], such a set is then mapped to non-leaf vertices of the structure steering the rounding process. We use a skeleton in a similar way, but before we are able to do that, we need to bound the integrality gap. Without outliers, it was sufficient to take the standard LP relaxation and decompose the graph into connected components. Although with outliers this is no longer the case, as shown in Section 5, a skeleton lets us both strengthen the LP relaxation, adding an appropriate constraint, and obtain a more granular decomposition of the initial instance into several subinstances, for which the strengthened LP relaxation is feasible and has bounded integrality gap. Further in Section 6 we show how each of these smaller instances can be independently rounded using tools previously applied for the capacitated setting [1].[2] Section 7 contains a wrap-up of the whole algorithm.

The improvements in the approximation ratio when soft or uniform capacities are considered, are postponed to the full version of the paper.

1.2 Related facility location work

The *facility location* problem is a central problem in operations research and computer science and has been a testbed for many new algorithmic ideas resulting a number of different approximation algorithms. In this problem, given a metric (via a weighted graph G), a set of nodes called *clients*, and opening costs on some nodes called *facilities*, the goal is to open a subset of facilities such that the sum of their opening costs and connection costs of clients to their nearest open facilities is minimized. Up to now, the best known approximation ratio is 1.488, due to Li [21] who used a randomized selection in Byrka's algorithm [6]. Guha and Khuller [14] showed that this problem is hard to approximate within a factor better than 1.463, assuming $NP \not\subseteq DTIME\big[n^{O(\log\log n)}\big]$.

When the facilities have capacities, the problem is called the *capacitated facility location* problem. It has also received a great deal of attention in recent years. Two main variants of the problem are soft-capacitated facility location and hard-capacitated facility location: in the latter problem, each facility is either opened at some location or not, whereas in

[2] The final rounding step can be also done using the path-like structures notion of [11], however we use the ideas of [1] as it allows cleaner presentation.

the former, one may specify any integer number of facilities to be opened at that location. Soft capacities make the problem easier and by modifying approximation algorithms for the uncapacitated problems, we can also handle this case [23, 18]. To the best of our knowledge all the existing constant-factor approximation algorithms for the general case of hard capacitated facility location are local search based, and the most recent of them is the 5-approximation algorithm of Bansal, Garg and Gupta [3]. The only LP-relaxation based approach for this problem is due to Levi, Shmoys and Swamy [20] who gave a 5-approximation algorithm for the special case in which all facility opening costs are equal (otherwise the LP does not have a constant integrality gap). Obtaining an LP based constant factor approximation algorithm for capacitated facility location is considered a major problem in approximation algorithms [24].

A problem very close to both facility location and k-center is the *k-median* problem in which we want to open at most k facilities and the goal is to minimize the sum of connection costs of clients to their nearest open facilities. Very recently Li and Svensson [22] obtained an LP rounding $(1+\sqrt{3})$-approximation algorithm, improving upon the previously best $(3+\varepsilon)$-approximation local search algorithm of Arya et al. [2]. Unfortunately obtaining a constant factor approximation algorithm for capacitated k-median still remains open despite consistent effort. The only previous attempts with constant approximation factors for this problem violate the capacities within a constant factor for the uniform capacity case [7] and the non-uniform capacity case [10] or exceed the number k of facilities by a constant factor [5].

2 Preliminaries

For a fixed instance of the Capacitated k-supplier with Outliers, we call (C, F, ϕ) *a solution* if it satisfies the required conditions. We often identify the solution by ϕ only (considering it as a partial function from $\mathcal{C}$ to $\mathcal{F}$), using C_ϕ and F_ϕ to refer to the other elements of the triple. If ϕ satisfies $\max_{v\in C} d(v, \phi(v)) \leq \tau$, we say that ϕ is a distance-τ solution.

Let $G = (V, E)$ be an undirected graph. By d_G we denote the metric defined by G. For sets $A, B \subseteq V$ we define $d_G(A, B) = \min_{a\in A, b\in B} d_G(a, b)$. If $B = \{b\}$ we write $d_G(A, b)$ instead of $d_G(A, B)$.

For a vertex $v \in V$ and an integer $k \in \mathbb{Z}_{\geq 0}$ we denote $N^k_G(v) = \{u \in V : d_G(u, v) = k\}$ and $N^k_G[v] = \{u \in V : d_G(u, v) \leq k\}$. We omit the superscript for $k = 1$ and the subscript if there is no confusion which graph we refer to.

For a set S and an element s by $S + s$ we denote $S \cup \{s\}$.

3 Reduction to graphic instances

As usual when working with a min max problem we start with the standard thresholding argument, i.e. reduce a general metric function to a metric defined by an unweighted graph.

We say that an instance of the k-supplier problem is *graphic*, if d is defined as the distance function of an unweighted bipartite graph $G = (\mathcal{C}, \mathcal{F}, E)$, and the goal is to find a distance-1 solution. An r-approximation algorithm is then allowed to either give a distance-r solution, or, only if it finds out that no distance-1 solution exists, a NO answer.

Below we show how to build an r-approximation algorithm for Capacitated k-supplier with Outliers given an r-approximation (in the aforementioned sense) for the graphic instances. Correctness of the reduction is standard. If an optimal solution exists, then

its value OPT belongs to T. In particular, in the phase corresponding to OPT, there is a distance-1 solution in $G_{\leq OPT}$. Thus the algorithm for graphic instances is required to find a solution. Therefore returns a solution ϕ for the first time at phase corresponding to $\tau^* \leq OPT$. Since $d(v,u) \leq \tau^* d_{G_{\leq \tau^*}}(v,u)$, ϕ is a distance-$r \cdot \tau^*$ solution, hence also distance-$r \cdot OPT$ solution.

$T := \{d(v,u) : v \in \mathcal{C}, u \in \mathcal{F}\}$;
foreach $\tau \in T$ *in ascending order* **do**
  $G_{\leq \tau} := (\mathcal{C}, \mathcal{F}, \{(v,u) : d(v,u) \leq \tau\})$;
  solve the graphic instance for $G_{\leq \tau}$;
  if *a solution* ϕ *found* **then return** ϕ;
return NO;

Algorithm 1: Reduction to graphic instances

4 Finding a skeleton

From now on we work with graphic instances only. Without loss of generality we may assume that $L(u) \leq deg(u)$ for each $u \in \mathcal{F}$. Indeed, setting $L(u) := \min(L(u), deg(u))$ has no influence on distance-1 solutions, while no additional distance-r solutions are created.

The first phase of the algorithm outputs several subsets of $\mathcal{F}$. If a distance-1 solution exists, at least one of them resembles (in a certain sense, to be defined later) a distance-1 solution and can be successfully used by the subsequent phases as a hint for constructing a distance-r solution. We formalize the features of a good hint in the following definition.

▶ **Definition 3.** A set $S \subseteq \mathcal{F}$ is called a *skeleton* if
- (**separation property**) $d(u,u') \geq 6$ for any $u, u' \in S$, $u \neq u'$,
- there exists a distance-1 solution (C_ϕ, F_ϕ, ϕ) such that:
  - (**covering property**) $d(u,S) \leq 4$ for each $u \in F_\phi$,
  - (**injection property**) there exists an injection $f : S \hookrightarrow F_\phi$ satisfying $d(u, f(u)) \leq 2$ for each $u \in S$.

If just separation and injection properties are satisfied, we call S a *preskeleton*.

In other words a skeleton is a set S, each vertex of which can be injectively mapped to a vertex of a distance-1 solution F_ϕ, and at the same time no two vertices of S are close and $N^4[S]$ contains the whole set F_ϕ.

Note that the separation property implies that sets $N^2[u]$ are pairwise disjoint for $u \in S$, hence any function $f : S \to F_\phi$ satisfying $d(u, f(u)) \leq 2$ is in fact an injection, however we make it explicit for the sake of presentation.

▶ **Lemma 4.** *Let S be a preskeleton and let $U = \{u \in \mathcal{F} : d(u,S) \geq 6\}$. Then S is a skeleton, or $U \neq \emptyset$ and $S + s$ is a preskeleton, where s is a highest-capacity vertex of U.*

Proof. Let ϕ be a distance-1 solution, which witnesses S being a preskeleton, where $f : S \hookrightarrow F_\phi$ satisfies the injection property. If ϕ witnesses S being a skeleton, we are done. Otherwise the covering property is not satisfied, hence there exists $u \in F_\phi$ such that $d(u,S) > 4$. Since d is a distance function of a bipartite graph, this implies $d(u,S) \geq 6$, so $u \in U \neq \emptyset$. If $|F_\phi \cap N^2[s]| \geq 1$, then ϕ already witnesses $S + s$ being a preskeleton, as one can extend the injection f by mapping a vertex of $F_\phi \cap N^2[s]$ to s. Therefore, we may assume that $N^2[s] \cap F_\phi = \emptyset$. In particular, this means that the clients in $N(s)$ are not served by any facility of F_ϕ.

Let us modify ϕ to obtain ψ as follows: close the facility in u, opening one in s instead. Let c be the number of clients assigned to u in ϕ. No longer serve these, instead serve any c neighbors of s in ψ (as we have observed before, they are not served in ϕ). Note that $c \leq L(u) \leq L(s) \leq \deg(s)$ by the choice of u maximizing the capacity and by the assumption of L being bounded by deg. Consequently, there are enough neighbors of s to serve, and the capacity constraint for s is satisfied. Moreover, the number of open facilities and the number of served clients are preserved. Other open facilities remain unchanged, so ψ satisfies the capacity and distance constraints for them, and therefore is a distance-1 solution. Finally, consider a function $f' = f + (s, s)$. As s is at distance at least 6 from S, by the injection property for S we know that s does not belong to the image of f, hence f' is an injection. Consequently ψ and f' ensure $S + s$ satisfies the injection property. Moreover s is far from S, hence $S + s$ is a preskeleton. ◀

With $\emptyset$ being trivially a preskeleton provided that any distance-1 solution exists, Lemma 4 lets us generate a sequence of sets, which contains a skeleton (see Algorithm 2). Note that any skeleton, by the injection property, is of size at most k.

▶ **Lemma 5.** *If there exists a distance-1 solution, there is at least one skeleton among sets output by Algorithm 2.*

$S := \emptyset$;
while $|S| \leq k - 1$ **do**
  $U := \{u \in \mathcal{F} \ : \ d(u, S) \geq 6\}$;
  if $U = \emptyset$ **then break**;
  $s := \operatorname{argmax}\{L(u) \ : \ u \in U\}$;
  $S := S + s$;
output S;

Algorithm 2: Construction of a family of sets containing at least one skeleton.

5 Clustering

For a set $S \subseteq \mathcal{F}$ define the following linear program $LP_{k,p}(G, L, S)$, where a variable y_u for $u \in \mathcal{F}$ denotes whether we open a facility in u or not, while a variable x_{uv} for $u \in \mathcal{F}$, $v \in \mathcal{C}$ corresponds to whether u serves v or not.

$$\sum_{u \in \mathcal{F}} y_u = k \tag{1}$$

$$\sum_{u \in \mathcal{F}, v \in \mathcal{C}} x_{uv} = p \tag{2}$$

$$x_{uv} \leq y_u \qquad \text{for each } u \in \mathcal{F}, v \in \mathcal{C} \tag{3}$$

$$\sum_{v} x_{uv} \leq L(u) \cdot y_u \qquad \text{for each } u \in \mathcal{F} \tag{4}$$

$$\sum_{u} x_{uv} \leq 1 \qquad \text{for each } v \in \mathcal{C} \tag{5}$$

$$\sum_{u \in \mathcal{F} \cap N^2[s]} y_u \geq 1 \qquad \text{for each } s \in S \tag{6}$$

$$x_{uv} = 0 \qquad \text{for each } u \in \mathcal{F}, v \in \mathcal{C} \text{ such that } (v, u) \notin E \tag{7}$$

$$\mathbf{0} \leq x, y \leq \mathbf{1} \tag{8}$$

Constraints (1)–(5),(7) are the standard constraints for Capacitated k-supplier with Outliers, ensuring that we open exactly k facilities (1), serve exactly p clients (2), obey capacity constraints (3)–(5), and serve clients which are close to facilities (7).

Observe that if S is a skeleton and a distance-1 solution ϕ witnesses that fact, we get a feasible solution of $LP_{k,p}(G, L, S)$ setting $y_u = 1$ iff $u \in F_\phi$ and $x_{uv} = 1$ iff $v \in C_\phi$ and $v = \phi(u)$. Indeed the injection property ensures that constraint (6) is satisfied. However, as usual in a capacitated problem with hard constraints, the integrality gap of this LP is unbounded. Similarly to the standard capacitated k-center [11], this issue is addressed by considering the connected components of G separately. When all the clients need to be served having a connected graph with a feasible solution of the standard LP is enough to round it [1, 11]. However, if we allow outliers, there are sill connected instances with arbitrarily large integrality gap.[3] For this reason we use the additional constraint (6) together with the assumption that all the vertices are close to S. This way we crucially exploit the covering, injection and separation properties of a skeleton.

In the following we shall prove that any instance with a skeleton can be decomposed into several smaller instances with additional properties. In the next section we will show how to round the obtained smaller instances.

▶ **Lemma 6.** *Let $S \subseteq \mathcal{F}$, let $G_1, \ldots, G_\ell$ be components of G after all vertices v with $d(v, S) > 5$ are removed and let $S_i = S \cap V(G_i)$ for $1 \leq i \leq \ell$.*

If S is a skeleton, then in polynomial time one can find partitions $k = \sum_{i=1}^{\ell} k_i$ and $p = \sum_{i=1}^{\ell} p_i$ such that $LP_{k_i,p_i}(G_i, L, S_i)$ are all feasible.

Proof. Observe that if S is a skeleton, then a witness solution ϕ opens facilities at distance at most 4 from S, and thus serves clients with distance at most 5 from S. Consequently all vertices further from S can be safely removed and S remains a skeleton. Then G might contain several connected components $G_1, \ldots, G_\ell$ with $G_i = (\mathcal{C}_i, \mathcal{F}_i, E_i)$. The witness solution ϕ can be partitioned among these components so that we get assignments ϕ_i which in total open k facilities to serve p clients. In particular, this means that for some partitions $k = \sum_i k_i$ and $p = \sum_i p_i$ sets $S_i = S \cap \mathcal{F}_i$ are skeletons, and consequently $LP_{k_i,p_i}(G_i, L, S_i)$ are feasible. The latter condition can be tested efficiently for any values k_i and p_i. While we cannot exhaustively test all partitions of k and p, dynamic programming lets us find partitions such that these linear programs are feasible for each i.

For $i \in \{0, \ldots, \ell\}$, $k' \in \{0, \ldots, k\}$ and $p' \in \{0, \ldots, p\}$ define a boolean value $F[i][k'][p']$, which equals **true** iff there exist partitions $k' = \sum_{j=1}^{i} k_j$ and $p' = \sum_{j=1}^{i} p_j$ such that $LP_{k_j,p_j}(G_j, L, S_j)$ are all feasible for $j \leq i$.

Clearly $F[0][0][0]$ is true, while $F[0][k'][p']$ is false for any other pair (k', p'). For $i > 1$ the value $F[i][k'][p']$ is simply an alternative of $F[i-1][k'-k_i][p'-p_i]$ for every pair (k_i, p_i) such that $LP_{k_i,p_i}(G_i, L, S_i)$ is feasible, $k_i \leq k'$ and $p_i \leq p'$. Thus in polynomial time one can check whether the desired partitions exists, and provided that together with a true value we also store the witness partitions, also find these partitions. ◀

6 Rounding

In the previous section we have shown how given a skeleton S one can partition the initial instance into smaller subinstances with more structural properties. Our main goal in this

[3] The construction is simple, but due to space restrictions it is left for the full version of the paper.

section is to show that those structural properties are in fact sufficient to construct a solution for each of the subinstances, which is formalized in the following lemma.

▶ **Lemma 7.** *Let $I = (G = (\mathcal{C}, \mathcal{F}, E), L, k, p)$ be an instance of* Capacitated k-supplier with Outliers *and let $S \subseteq \mathcal{F}$. If the following four conditions are satisfied:*

(i) *G is connected,*
(ii) *for any $u, u' \in S$, $u \neq u'$ we have $d(u, u') \geq 6$,*
(iii) *$N^5[S] = \mathcal{F} \cup \mathcal{C}$,*
(iv) *$LP_{k,p}(G, L, S)$ admits a feasible solution,*

then one can find a distance-25 solution for I in polynomial time.

Before we give a proof of Lemma 7, in Section 6.1 we recall (an adjusted version) of a distance-r transfer, a very useful notion introduced in [1], together with its main properties. Next, in Section 6.2 we prove Lemma 7.

6.1 Distance r-transfer

▶ **Definition 8.** Given a graph $G = (V, E)$ with $W \subseteq V$, a capacity function $L : W \to \mathbb{Z}_{\geq 0}$ and $y \in \mathbb{R}^W_{\geq 0}$, a vector $y' \in \mathbb{R}^W_{\geq 0}$ is a distance-r transfer of (G, L, y) if

1. $\sum_{v \in W} y'_v = \sum_{v \in W} y_v$ and
2. $\sum_{v \in W : d(v,U) \leq r} L(v) y'_v \geq \sum_{u \in U} L(u) y_u$ for all $U \subseteq W$.

If y' is a characteristic vector of $F \subseteq W$, we say that F is an integral distance-r transfer of (G, L, y).

Less formally a distance-r transfer is a reassignment, where the sum of y-variables is preserved and locally for any set $U \subseteq W$ the total fractional capacity in a small neighborhood of U does not decrease.

Like in [1], an integral distance-r transfer of the fractional solution of the LP already gives a distance-$r+1$ solution (in particular point 2 of Definition 8 ensures that the Hall's condition is satisfied). The proof must be modified though, so that it encompasses outliers.

▶ **Lemma 9.** *Let $G = (\mathcal{C}, \mathcal{F}, E)$ be a bipartite graph with a capacity function $L : \mathcal{F} \to \mathbb{Z}_{\geq 0}$. Assume (x, y) is a feasible solution of $LP_{k,p}(G, L, S)$ and $F \subseteq \mathcal{F}$ is an integral distance-r transfer of y. Then one can find a distance-$r+1$ solution (C, F, ϕ) in polynomial time.*

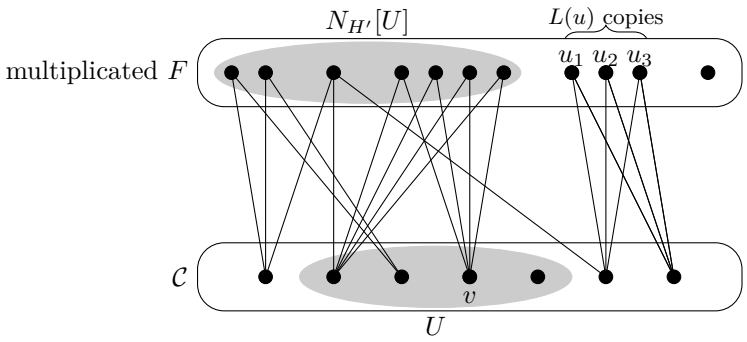


Figure 1 Graph H' obtained from H by removing vertices from $\mathcal{F} \setminus F$ and duplicating each vertex $u \in F$ to its capacity. Shaded ellipses represent sets used in Hall's theorem.

Proof. Consider a bipartite graph $H = (\mathcal{C}, \mathcal{F}, E_H)$ with $(v,u) \in E_H$ if $d_G(v,u) \leq r+1$. Modify H to obtain H' by removing vertices from $\mathcal{F} \setminus F$ and duplicating each vertex $u \in F$ to its capacity, i.e. $L(u)$ times, see also Fig. 1. Observe that cardinality-p matchings in this graph correspond to distance-$r+1$ solutions for G. If any, such a matching can clearly be found in polynomial time. We shall prove its existence by checking the deficit version of Hall's theorem, i.e. that for each $U \subseteq \mathcal{C}$ we have

$$\sum_{u \in F : d(u,U) \leq r+1} L(u) \geq |U| - |\mathcal{C}| + p$$

First, observe that

$$\sum_{v \in U, u \in \mathcal{F}} x_{uv} = \sum_{v \in \mathcal{C}, u \in \mathcal{F}} x_{uv} - \sum_{v \in \mathcal{C} \setminus U, u \in F} x_{uv} \overset{(2),(5)}{\geq} p - \sum_{v \in \mathcal{C} \setminus U} 1 = p - |\mathcal{C} \setminus U| = |U| - |\mathcal{C}| + p.$$

Moreover

$$\sum_{v \in U, u \in \mathcal{F}} x_{uv} = \sum_{v \in U, u \in N_G(U)} x_{uv} \leq \sum_{u \in N_G(U)} \sum_{v \in \mathcal{C}} x_{uv} \overset{(4)}{\leq} \sum_{u \in N_G(U)} L(u) y_u$$
$$\overset{\text{Def. 8 point 2}}{\leq} \sum_{u \in \mathcal{F} : d_G(u, N_G(U)) \leq r} L(u) = \sum_{u \in \mathcal{F} : d_G(u,U) \leq r+1} L(u)\,.$$

Together these equalities conclude the proof. ◀

We proceed with a pair of simple properties of transfers.

▶ **Fact 10.** *Let $G = (V,E)$ be a graph with $W \subseteq V$ and a capacity function $L : W \to \mathbb{Z}_{\geq 0}$, and let $y, y', y'' \in \mathbb{R}^W_{\geq 0}$. Assume y' is a distance-r transfer of (G, L, y) and y'' is a distance-r' transfer of (G, L, y'). Then y'' is a distance-$r + r'$ transfer of (G, L, y).*

▶ **Fact 11.** *Let $G = (V,E)$ and $G' = (V', E')$ be graphs with $W \subseteq V$ and $W \subseteq V'$ and a capacity function $L : W \to \mathbb{Z}_{\geq 0}$. Let $y, y' \in \mathbb{R}^W_{\geq 0}$ and let $f : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be a monotonic function such that $d_G(u,v) \leq f(d_{G'}(u,v))$ for any $u, v \in W$. Assume y' is a distance-r transfer of (G', L, y). Then y' is a distance-$f(r)$ transfer of (G, L, y).*

The following is the main technical contribution of [1].

▶ **Lemma 12** ([1])**.** *Let $T = (V,E)$ be a tree with a capacity function $L : V \to \mathbb{Z}_{\geq 0}$ and let $y \in [0,1]^V$ be a vector such that $y_v = 1$ for every non-leaf $v \in V$ and $\sum_{v \in V} y_v \in \mathbb{Z}_{\geq 0}$. Then one can find in polynomial time an integral distance-2 transfer of (T, L, y).*

6.2 Final rounding

▶ **Lemma 13.** *Let $G = (\mathcal{C}, \mathcal{F}, E)$ be a connected bipartite graph and let $S \subseteq \mathcal{F}$ such that $d(v,S) \leq 5$ for every $v \in \mathcal{C} \cup \mathcal{F}$. There exists an auxiliary tree $T = (S, E_T)$ such that $d(u,u') \leq 10$ for any $\{u,u'\} \in E_T$. Moreover, such a tree can be computed in polynomial time.*

Proof. We shall grow a tree adding a leaf in each step. At the beginning we select any $s \in S$ and initialize with a single-vertex tree. Assume we have already grown a tree with vertex-set $S' \subseteq S$. Choose a shortest path connecting S' to $S' \setminus S$. Such a path exists since G is connected. If its length is at most 10, we add the endpoint in $S \setminus S'$ to the tree,

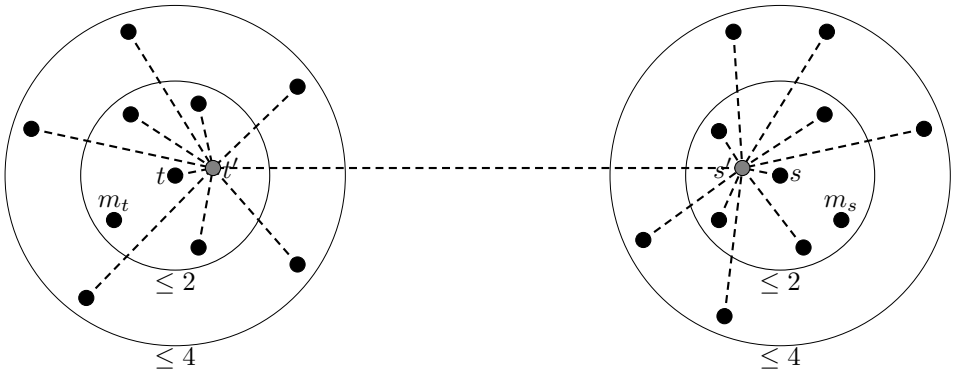


Figure 2 A fragment of the tree T' with $s, t \in S$. Nodes of $\mathcal{F}$ are marked in black, of S' in gray. Edges of T' are represented as dashed lines. Note that m_s and m_t are not vertices of T'.

joining it with the other endpoint. For a proof by contradiction assume that a shortest path has length greater than 10. Since G is bipartite, its length needs to be even, and thus at least 12. Choose the midpoint of such a path. Its distance both to S' and to $S' \setminus S$ is at least 6, otherwise the path could be shortened. This vertex contradicts the assumption that $d(v, S) \leq 5$ for every $v \in \mathcal{C} \cup \mathcal{F}$. ◀

We are ready to prove Lemma 7.

Proof of Lemma 7. Since G is connected and every vertex of G is within distance 5 from S, we can use Lemma 13 to construct a tree $T = (S, E_T)$. Let us add a duplicate s' of every $s \in S$ to create a bipartite graph $G' = (\mathcal{C}, \mathcal{F}', E')$, where $\mathcal{F}' = \mathcal{F} \cup S'$ and $S' = \{s' : s \in S\}$. For each $s \in S$ choose $m_s = \operatorname{argmax}\{L(u) : u \in N^2[s] \cap \mathcal{F}\}$ and set $L(s') = L(m_s)$. Let us create a tree T' with $V(T') = \mathcal{F}' \setminus \{m_s : s \in S\}$. We build it in two steps, see also Fig. 2:

1. create a tree with vertex set S' so that $\{u', v'\}$ is an edge iff $\{u, v\} \in E(T)$,
2. connect each vertex in $\mathcal{F} \setminus \{m_s : s \in S\}$ to the closest vertex in S'.

Observe that endpoints of the edges created in the first step are at most at distance 10 in G', while endpoints of the edges created in the second step, at most at distance 4. Consequently, $d_{G'}(u, v) \leq 10 d_{T'}(u, v)$ for any $u, v \in V(T')$. Moreover, note that all non-leaves of T' belong to S'.

Let (x, y) be a feasible solution of $LP_{k,p}(G, L, S)$. Note that y can be interpreted as a vector in $\mathbb{R}_{\geq 0}^{\mathcal{F}'}$ extending with zeroes at S'. We shall give an integral distance-24 transfer F of (G', L, y). Despite it being formally a transfer in G', F will be a subset of $\mathcal{F}$, i.e. a transfer of (G, L, y) as well.

Recall that by (2), the sets $N^2[s]$ are pairwise disjoint and in particular m_s are pairwise different. This lets us use (6) to gather in s' one unit from $N^2[s]$ for every $s \in S$ so that the whole value in m_s is transferred to s'. Note that $L(s') \geq L(u)$ for each $u \in N^2[s]$, so this way we obtain a distance-2 transfer y' of (G', L, y). Additionally, we have made sure that $y'_{m_s} = 0$, so y' can be interpreted as a vector in $\mathbb{R}_{\geq 0}^{V(T')}$, and that $y'_{s'} = 1$, so y' is 1 for all non-leaves of T'. This lets us use Lemma 12 to obtain an integral distance-2 transfer $F' \subseteq V(T')$ of (T', L, y'). According to Fact 11 it can be interpreted as a distance-20 transfer of (G', L, y'). Finally we move the value from s' to m_s for each $s \in S$. Note that these vertices have equal capacities, so this step can be interpreted as an integral distance-2 transfer.

The final transfer is therefore a composition of a distance-2 transfer, a distance-20 transfer and a distance-2 transfer. Thus, by Fact 10 it is a distance-24 transfer.[4] By Lemma 9 having an integral distance-24 transfer is enough to construct a distance-25 solution ϕ in polynomial time, which concludes the proof of Lemma 7. ◀

7 Wrap-up

With the results of previous section, we are ready to the prove the main theorem.

▶ **Theorem 14.** *The* Capacitated k-supplier with Outliers *problem admits a 25-approximation algorithm.*

Proof. Section 3 with Algorithm 1 provides (a Turing-like) reduction to graphic instances. Algorithm 2 of Section 4 given such an instance outputs several sets. Provided that a distance-1 solution exists, one of them is guaranteed to be a skeleton. Each of these sets is then processed separately. As described in Section 5, some redundant vertices are removed and the graph is partitioned into connected components. Dynamic programming (Lemma 6) is then used to find a compatible partition of k and p, so that each linear program $LP_{k_i,p_i}(G_i, L, S_i)$ admits a feasible solution. While this procedure might fail in general, it is guaranteed to succeed for a skeleton, hence at least once if a distance-1 solution exists.

Note that if such a partition is found, then for each of the instances (G_i, L, k_i, p_i) together with sets S_i, we can use Lemma 7 as all the conditions $(i) - (iv)$ are satisfied. A sum of solutions for these ℓ instances is finally returned as a distance-25 solution for the original graphic instance. ◀

Acknowledgements. We would like to thank Samir Khuller for suggesting the study of this variant of the k-center problem and helpful discussions.

References

1 Hyung-Chan An, Aditya Bhaskara, and Ola Svensson. Centrality of trees for capacitated k-center. *CoRR*, abs/1304.2983, 2013.

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 Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *STOC*, pages 21–29. ACM, 2001.

3 Manisha Bansal, Naveen Garg, and Neelima Gupta. A 5-approximation for capacitated facility location. In Leah Epstein and Paolo Ferragina, editors, *ESA*, volume 7501 of *Lecture Notes in Computer Science*, pages 133–144. Springer, 2012.

4 Judit Bar-Ilan, Guy Kortsarz, and David Peleg. How to allocate network centers. *Journal of Algorithms*, 15(3):385–415, 1993.

5 Yair Bartal, Moses Charikar, and Danny Raz. Approximating min-sum k-clustering in metric spaces. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *STOC*, pages 11–20. ACM, 2001.

[4] A simpler construction gives a distance-30 transfer, without introducing additional vertices S'. It is enough first to gather one unit from $N^2[s]$ in m_s and build a tree on vertices m_s, where adjacent vertices of the tree are at distance at most 14 in G. By using Lemma 12 one obtains a distance-28 transfer, which together with the initial distance-2 transfer gives an integral distance-30 transfer.

6 Jaroslaw Byrka and Karen Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM J. Comput.*, 39(6):2212–2231, 2010.

7 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002.

8 Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In S. Rao Kosaraju, editor, *SODA*, pages 642–651. ACM/SIAM, 2001.

9 Julia Chuzhoy, Sudipto Guha, Eran Halperin, Sanjeev Khanna, Guy Kortsarz, Robert Krauthgamer, and Joseph Naor. Asymmetric k-center is $\log^* n$-hard to approximate. *Journal of the ACM*, 52(4):538–551, 2005.

10 Julia Chuzhoy and Yuval Rabani. Approximating k-median with non-uniform capacities. In *SODA*, pages 952–958. SIAM, 2005.

11 Marek Cygan, MohammadTaghi Hajiaghayi, and Samir Khuller. LP rounding for k-centers with non-uniform hard capacities. In *FOCS*, pages 273–282. IEEE Computer Society, 2012.

12 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

13 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

14 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999.

15 Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10:180–184, 1985.

16 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33(3):533–550, 1986.

17 Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1:209–216, 1979.

18 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

19 Samir Khuller and Yoram J. Sussmann. The capacitated k-center problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000.

20 Retsef Levi, David B. Shmoys, and Chaitanya Swamy. LP-based approximation algorithms for capacitated facility location. *Mathematical Programming*, 131(1-2):365–379, 2012.

21 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013.

22 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *STOC*, pages 901–910. ACM, 2013.

23 David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In Frank Thomson Leighton and Peter W. Shor, editors, *STOC*, pages 265–274. ACM, 1997.

24 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

Bounds on the Cover Time of Parallel Rotor Walks*

Dariusz Dereniowski[1], Adrian Kosowski[2,3], Dominik Pająk[2], and Przemysław Uznański[2,4]

1 **Department of Algorithms and System Modeling, Gdańsk University of Technology, Gdańsk, Poland**
2 **CEPAGE Project, Inria Bordeaux Sud-Ouest – LaBRI, Talence, France**
3 **GANG Project, Inria Paris Rocquencourt – LIAFA, Paris, France**
4 **CNRS and Aix-Marseille Université – LIF, Marseille, France**

Abstract

The *rotor-router mechanism* was introduced as a deterministic alternative to the random walk in undirected graphs. In this model, a set of k identical walkers is deployed in parallel, starting from a chosen subset of nodes, and moving around the graph in synchronous steps. During the process, each node maintains a cyclic ordering of its outgoing arcs, and successively propagates walkers which visit it along its outgoing arcs in round-robin fashion, according to the fixed ordering.

We consider the *cover time* of such a system, i.e., the number of steps after which each node has been visited by at least one walk, regardless of the starting locations of the walks. In the case of $k = 1$, Yanovski et al. (2003) and Bampas et al. (2009) showed that a single walk achieves a cover time of exactly $\Theta(mD)$ for any n-node graph with m edges and diameter D, and that the walker eventually stabilizes to a traversal of an Eulerian circuit on the set of all directed edges of the graph. For $k > 1$ parallel walks, no similar structural behaviour can be observed.

In this work we provide tight bounds on the cover time of k parallel rotor walks in a graph. We show that this cover time is at most $\Theta(mD/\log k)$ and at least $\Theta(mD/k)$ for any graph, which corresponds to a speedup of between $\Theta(\log k)$ and $\Theta(k)$ with respect to the cover time of a single walk. Both of these extremal values of speedup are achieved for some graph classes. Our results hold for up to a polynomially large number of walks, $k = O(poly(n))$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Distributed graph exploration, Rotor-Router, Collaborative robots, Parallel random walks, Derandomization

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.263

1 Introduction

In graph exploration problems, a walker or group of walkers (agents) is placed on a node of a graph and moves between adjacent nodes, with the goal of visiting all the nodes of the graph. The study of graph exploration is closely linked to central problems of theoretical computer science, such as the question of deciding if two nodes of the graph belong to the same connected component (*st-connectivity*). For example, fast approaches to connectivity testing

* Research partially supported by ANR project DISPLEXITY and by NCN under contract DEC-2011/02/A/ST6/00201. Dariusz Dereniowski was partially supported by a scholarship for outstanding young researchers founded by the Polish Ministry of Science and Higher Education. The full text of the paper is available online at: `http://hal.inria.fr/hal-00865065`.



© Dariusz Dereniowski, Adrian Kosowski, Dominik Pająk, and Przemysław Uznański;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 263–275



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



in little memory rely on the deployment of multiple random walks [6, 11]. In these algorithms, the initial locations of the walkers are chosen according to a specific probability distribution.

More recently, multiple walks have been studied in a worst-case scenario where the k agents are placed on some set of starting nodes and deployed in parallel, in synchronous steps. The considered parameter is the *cover time* of the process, i.e., the number of steps until each node of the graph has been visited by at least one walker. Alon *et al.* [2], Efremenko and Reingold [9], and Elsässer and Sauerwald [10] have studied the notion of the *speedup* of the random walk for an undirected graph G, defined as the ratio between the cover time of a k-agent walk in G for worst-case initial positions of agents and that of a single-agent walk in G starting from a worst-case initial position, as a function of k. A characterization of the speedup has been achieved for many graph classes with special properties, such as small mixing time compared to cover time. However, a central question poised in [2] still remains open: what are the minimum and maximum values of speed-up of the random walk in arbitrary graphs? The smallest known value of speedup is $\Theta(\log k)$, attained e.g. for the cycle, while the largest known value is $\Theta(k)$, attained for many graph classes, such as expanders, cliques, and stars.

In this work, we consider a deterministic model of walks on graphs, known as the *rotor-router*. The rotor-router model, introduced by Priezzhev *et al.* [14], provides a mechanism for the environment to control the movement of the agent deterministically, mimicking the properties of exploration as the random walk. In the rotor-router, the agent has no operational memory and the whole routing mechanism is provided within the environment. The edges outgoing from each node v are arranged in a fixed cyclic order known as a *port ordering*, which does not change during the exploration. Each node v maintains a *pointer* which indicates the edge to be traversed by the agent during its next visit to v. If the agent has not visited node v yet, then the pointer points to an arbitrary edge adjacent to v. The next time when the agent enters node v, it is directed along the edge indicated by the pointer, which is then advanced to the next edge in the cyclic order of the edges adjacent to v.

For a single agent, the (deterministic) cover time of the rotor-router and the (expected) cover time of the random walk prove to be surprisingly convergent for many graph classes. In general, it is known that for any n-node graph of m edges and diameter D, the cover time of the rotor-router in a worst-case initialization is precisely $\Theta(mD)$ [16, 3]. By comparison, the random walk satisfies an upper bound of $O(mD \log n)$ on the cover time, though this bound is far from tight for many graph classes.

The behavior of the rotor-router model with multiple agents appears to be much more complicated. Since the parallel walkers interact with the pointers of a single rotor-router system, they cannot be considered independent (in contrast to the case of parallel random walks). In the first work on the topic, Yanovski *et al.* [16] showed that adding a new agent to a rotor-router system with k agents cannot increase the cover time, and showed experimental evidence suggesting that a speedup does indeed occur. Klasing *et al.* [13] have provided the first evidence of speedup, showing that for the special case when G is a cycle, a k-agent system explores an n-node cycle $\Theta(\log k)$ times more quickly than a single agent system.

In this work we completely resolve the question of the possible range of speedups of the parallel rotor-router model in a graph, showing that its value is between $\Theta(\log k)$ and $\Theta(k)$, for any graph. Both of these bounds are tight. Thus, the proven range of speedup for the rotor-router corresponds precisely to the conjectured range of speedup for the random walk.

1.1 Related work

The rotor-router model. Studies of the rotor-router started with works of Wagner *et al.* [15] who showed that in this model, starting from an arbitrary configuration (arbitrary cyclic

orders of edges, arbitrary initial values of the port pointers and an arbitrary starting node) the agent covers all m edges of an n-node graph within $O(nm)$ steps. Bhatt *et al.* [5] showed later that within $O(nm)$ steps the agent not only covers all edges but enters (establishes) an Eulerian cycle. More precisely, after the initial stabilization period of $O(nm)$ steps, the agent keeps repeating the same Eulerian cycle of the directed symmetric version $\vec{G}$ of graph $\vec{G}$ (see Section 3 for a definition). Subsequently, Yanovski *et al.* [16] and Bampas *et al.* [3] showed that the Eulerian cycle is in the worst case entered within $\Theta(mD)$ steps in a graph of diameter D. Considerations of specific graph classes were performed in [12]. Robustness properties of the rotor-router were further studied in [4], who considered the time required for the rotor-router to stabilize to a (new) Eulerian cycle after an edge is added or removed from the graph. Regarding the terminology, we note that the rotor-router model has also been referred to as the *Propp machine* [3] or *Edge Ant Walk algorithm* [15, 16], and has also been described in [5] in terms of traversing a maze and marking edges with pebbles. Studies of the multi-agent rotor-router was performed by Yanovski *et al.* [16] and Klasing *et al.* [13], and its speedup was considered for both worst-case and best-case scenarios.

A variant of the multi-agent rotor-router mechanism has been extensively studied in a different setting, in the context of balancing the workload in a network. The single agent is replaced with a number of agents, referred to as *tokens*. Cooper and Spencer [7] study d-dimensional grid graphs and show a constant bound on the discrepancy, defined as the difference between the number of tokens at a given node v in the rotor-router model and the expected number of tokens at v in the random-walk model. Subsequently, Doerr and Friedrich [8] analyze in more detail the distribution of tokens in the rotor-router mechanism on the 2-dimensional grid. Akbari and Berenbrink [1] showed an upper bound of $O(\log^{3/2} n)$ on the discrepancy for hypercubes and a bound of $O(1)$ for a constant-dimensional torus.

Parallel random walks. Alon *et al.* [2] introduced the notion of the speed-up of k independent random walks as the ratio of the cover time of a single walk to the cover time of k random walks. They conjectured that the speed-up is between $\log k$ and k for any graph. The speedup was shown to be k for many graph classes, such as complete graphs [2], d-dimensional grids [2, 10], hypercubes [2, 10], expanders [2, 10], and different models of random graphs [2, 10]. For the cycle, the speed-up is equal to $\log k$ [2]. For general graphs, an upper bound $\min\{k \log n, k^2\}$ on the speed-up was obtained by Efremenko *et al.* [9]. Independently, Elsässer *et al.* [10] showed the $k \log n$ upper bound. Another measure studied by Efremenko *et al.* [9] concerns the speedup with respect to a different exploration parameter — the maximing hitting time, i.e., the maximum over all pairs of nodes of the graph of the expected time required by the walk to move from one node to the other. For this parameter, they show a bound on speedup of $O(k)$, mentioning that it is tight in many graph classes.

1.2 Our results and overview of the paper

In this work we establish bounds on the minimum and maximum possible cover time for a worst-case initialization of a k-rotor-router system in a graph G with m edges and diameter D.

We start by providing a formal definition of the rotor-router model and recalling its basic properties in Section 2. In Section 3, we first prove that the cover time t_C satisfies $t_C \in O(mD/\log k)$, when $k < 2^{16D}$. We then extend this result to the case of $k \in O(poly(n))$, i.e., $k < n^c$ for some absolute constant c. The main part of our proofs relies on a global analysis of the number of visits to edges in successive time steps, depending on the number of times that these edges have been traversed in the past. We first prove a stronger version of local structural lemmas proposed by Yanovski *et al.* [16], and apply them within a global

Table 1 Values of speed-up for k-agent exploration with the rotor-router and parallel random walks. All results hold at least for $k \leq n$, except for those cited from [13] which hold for $k \leq n^{1/11}$.

Graph class	Speedup of Rotor-Router *for cover time*		Speedup of Random Walk *for cover time*		Speedup of Random Walk *for max hitting time*	
General case:	$\Omega(\log k)$, $O(k)$	**(Thm. 8, 9)**	$O(k^2)$, $O(k \log n)$	[9, 10]	$O(k)$	[10]
Cycle:	$\Theta(\log k)$	[13]	$\Theta(\log k)$	[2]	$\Theta(\log k)$	[2]
Star:	$\Theta(k)$	**(Prop. 10)**	$\Theta(k)$	[2]	$\Theta(k)$	[2]

amortization argument over all time steps and all edges in the graph. The extension to the case of $k \in O(poly(n))$ relies on a variant of a similar amortized analysis, and also makes use of a technique known as *delayed deployments* introduced by Klasing *et al.* [13], which we briefly recall in Section 2. We remark that by [13], a cover time of $\Theta(mD/\log k)$ is achieved when G is a cycle with all agents starting from one node, when $k < n^{1/11}$.

In Section 4, we show a complementary lower bound on the cover time of the k-agent rotor-router in worst case initialization, namely, $t_C \in \Omega(mD/k)$. As a starting point, the proof uses a decomposition of the edge set of a graph, introduced by Bampas *et al.* [3], into a "heavy part" containing a constant proportion of the edges and a "deep part", having diameter linear in D. The main part of the analysis is to show that an appropriate initialization of k agents in the heavy part takes a long time to reach the most distant nodes of the deep part. The argument also takes advantage of the delayed deployment technique. We close the section by remarking that a cover time of $\Theta(mD/k)$ is, in fact, achieved for some graphs, such as stars.

Table 1 contains a summary of our results on the speed-up of the k-agent rotor-router, compared to corresponding results from the literature for parallel random walks. Note that for a deterministic process such as the rotor-router, the notions of cover time and maximum hitting are equivalent, and hence we only refer to cover times.

2 Model and preliminaries

Let $G = (V, E)$ be an undirected connected graph with n nodes, m edges and diameter D. We denote the neighborhood of a node $v \in V$ by $\Gamma(v)$. The directed graph $\vec{G} = (V, \vec{E})$ is the directed symmetric version of G, where the set of arcs $\vec{E} = \{(v, u) : \{v, u\} \in E\}$. We will denote arc (v, u) by $v \to u$.

Model definition. We consider the rotor-router model (on graph G) with $k \geq 1$ indistinguishable agents, which run in steps, synchronized by a global clock. In each step, each agent moves in discrete steps from node to node along the arcs of graph $\vec{G}$. A *configuration* at the current step is defined as a triple $((\rho_v)_{v \in V}, (\pi_v)_{v \in V}, \{r_1, \ldots, r_k\})$, where ρ_v is a cyclic order of the arcs (in graph $\vec{G}$) outgoing from node v, π_v is an arc outgoing from node v, which is referred to as *the (current) port pointer at node* v, and $\{r_1, \ldots, r_k\}$ is the (multi-)set of nodes currently containing an agent. For each node $v \in V$, the cyclic order ρ_v of the arcs outgoing from v is fixed at the beginning of exploration and does not change in any way from step to step.

For an arc $v \to u$, let $next(v \to u)$ denote the arc next after arc $(v \to u)$ in the cyclic order ρ_v. The exploration starts from some initial configuration and then keeps running in all future rounds, without ever terminating. During the current step, first each agent i is moved from node r_i traversing the arc π_{r_i}, and then the port pointer π_{r_i} at node r_i is

advanced to the next arc outgoing from r_i (that is, π_{r_i} becomes $next(\pi_{r_i})$). This is performed sequentially for all k agents. Note that the order in which agents are released within the same step is irrelevant from the perspective of the system, since agents are indistinguishable. For example, if a node v contained two agents at the start of a step, then it will send one of the agents along the arc π_v, and the other along the arc $(v, next(\pi_v))$.

Notation. Throughout the paper, $\mathbb{N}_+$ denotes the set of positive integers, and $\mathbb{N} = \mathbb{N}_+ \cup \{0\}$. We introduce compact notation for discrete intervals of integers: $[a, b] \equiv \{a, a+1, \ldots, b\}$, and $[a, b) \equiv [a, b-1]$, for $a, b \in \mathbb{N}$.

We will denote by $a^{(t)}(e)$ the number of agents traversing directed arc $e \in \vec{E}$ during step $t+1$. We recall that multiple agents traversing one arc $e \in \vec{E}$ in the same time step t are considered to move simultaneously. By $d^{(t)}(e)$ we denote the number of traversals of directed arc $e \in \vec{E}$ till the end of step t, $d^{(t)}(e) = \sum_{t' \in [0,t)} a^{(t')}(e)$. For a node $v \in V$, let $d^{(t)}(v) = \min_{w \in \Gamma(v)}\{d^{(t)}(v \to w)\}$ be the number of fully completed rotations of the rotor at node v at the end of step t. We note that for any arc $u \to v \in \vec{E}$, $0 \leq d^{(t)}(u \to v) - d^{(t)}(u) \leq 1$ [16].

We also denote $V_i^{(t)} = \{v \in V : d^{(t)}(v) \leq i\}$ and $E_i^{(t)} = \{e \in \vec{E} : d^{(t)}(e) \leq i\}$. Given a graph $G = (V, E)$ and a subset $X \subseteq V$, $G[X]$ denotes the subgraph of G *induced by* X, $G[X] = (X, \{\{u, v\} \in E \mid u, v \in X\})$.

Delayed deployment technique. In some of the proofs, we will make use of modified executions of the k-agent rotor-router system called *delayed deployments* [13], in which some agents may be stopped at a node, skipping their move for some number of rounds. Formally, a delayed deployment D of k agents is defined as a function $D : V \times \mathbb{N} \to \mathbb{N}$, where $D(v, t) \geq 0$ represents the number of agents which are stopped in node v in step t of the execution of the system. Delayed deployments may be conveniently viewed as algorithmic procedures for delaying agents, and are introduced for purposes of analysis, only. The following lemma relates the cover time of the rotor-router system to that of its delayed deployment.

▶ **Lemma 1.** *[13] Let R be a k-rotor router system with an arbitrarily chosen initialization, and let D be any delayed deployment of R. Suppose that deployment D covers all the nodes of the graph after T rounds, and in at least τ of these rounds, all k agents were active in D. Then, the cover time t_C of the rotor-router system R can be bounded by: $\tau \leq t_C \leq T$.*

3 Upper bound on cover time

In this section, we will show that a k-agent parallel rotor-router system explores a graph in $O(mD/\log k)$ steps, regardless of initialization. We start by providing an informal intuition of the main idea of the proof. After some initialization phase of duration t_0, but before exploration is completed at time t_C, we consider a shortest path connecting the arc of the graph which has already been visited many times at time t_0, with an arc which will remain unvisited at time t_C. We look at the number of visits to consecutive arcs on this path. It turns out that the rotor-router admits a property which can be informally stated as follows: if, up to some step t of exploration, an arc e_{l+1} of the considered path has been traversed more times than the next arc e_l on the path by some difference of δ, then in the next step $t+1$ of exploration, at least $\delta - O(1)$ agents will traverse arcs which have, so far, been visited not more often (up to a constant additive factor) than e_l. In this way, the larger the discrepancy between the number of visits to adjacent arcs, the more activity will the rotor-router perform to even out this discrepancy, by traversing under-visited arcs. This

load-balancing behavior of the system will be shown to account for the $(\log k)$-speedup in cover time with respect to the case of a single agent.

We start by proving two structural lemmas which generalize the results of Yanovski *et al.* [16, Theorem 2]. The first lemma establishes a connection between the existence of an arc entering a subset of nodes $S \subseteq V$ that has been traversed more times than all arcs outgoing from S, and the number of agents currently located within set S.

▶ **Lemma 2.** *For any time $t \in \mathbb{N}$ and $d \in \mathbb{N}$, consider the partition of the set of nodes $V = S \cup T$ such that each node in set S (set T) has completed at most d (more than d) full cycles of if its rotor, $S = V_d^{(t)}$ and $T = V \setminus S$. Suppose that for some nodes $v \in S$, $u \in T$, and some $\delta \in \mathbb{N}$, there exists an arc $u \to v$, such that $d^{(t)}(u \to v) \geq d + \delta$. Then, the set of arcs having their tail at a node of S will be traversed by at least $\delta - 1$ agents in total in step $t + 1$.*

By an application of the above lemma, we obtain the key property of a pair of consecutive arcs which have a different number of traversals at time t.

▶ **Lemma 3.** *Let $G = (V, E)$ be any undirected graph and let $e_2 = u \to v$, $e_1 = v \to w$ be two consecutive arcs of $\vec{G}$. Fix a time step $t \in \mathbb{N}_+$. Then, for any $x \geq d^{(t)}(e_1) + 1$, the number of agents that traverse arcs from set $E_x^{(t)}$ in time step $t + 1$ satisfies:*

$$\sum_{e \in E_x^{(t)}} a^{(t)}(e) \geq d^{(t)}(e_2) - d^{(t)}(e_1) - 1.$$

The property of the rotor-router captured by the above lemma is, in fact, sufficient to prove the main results of the section, following the general approach outlined at the beginning of the section. To show a bound of $t_C \in O(mD/\log k)$, we will apply two separate arguments, first one for the range of relative small k ($k \in 2^{O(D)}$, which corresponds to $t_C \in \Omega(m)$), and then one for values of k which are larger, but polynomially bounded with respect to n.

▶ **Theorem 4.** *Let $G = (V, E)$ be any undirected graph with arbitrary initialization of pointers and let D be the diameter of G. If $k \leq 2^{16D}$, then a team of k agents performing in parallel the rotor-router movement explores G in less than $500mD/\log k$ steps, regardless of the initial positions of agents.*

Proof. First, assume that $k > 2^{160}$ and fix $b = \lfloor (\log k)/2 \rfloor$. Consider the first t_0 steps, where $t_0 = \lceil 2^{b+1} mD/k \rceil$. Since in every step there are exactly k arc traversals, the total number of them during the first t_0 steps is at least $2^{b+1}mD$. We have $2m$ arcs in total. Thus, there exists an arc e' such that $d^{(t_0)}(e') \geq 2^b D$. These first t_0 steps we will call as a form of setup stage, after which we begin to analyze the behavior of the rotor-router process.

Denote by t_C the cover time of G with k agents for a given initialization. We will assume that $t_C > t_0$, i.e., at least one arc of the graph has not been explored at time t_0; otherwise, $t_C \leq t_0 = \lceil 2^{b+1} mD/k \rceil \leq \lceil 2mD/\sqrt{k} \rceil$, since $b = \lfloor (\log k)/2 \rfloor$, and the claim of the theorem holds for all k.

Take $e'' \in \vec{E}$ to be an arc which is explored for the first time in step t_C, i.e., such that $d^{(t_C - 1)}(e'') = 0$. Since the diameter of G is D, there exists a path $\mathcal{P} = \langle e'' = e_1, e_2, \ldots e_{D'} = e' \rangle$ such that $D' \leq D + 2$, and for each $l \in [1, D']$, $e_l = v_{l+1} \to v_l$ where $v_l, v_{l+1} \in V$.

Fix a time step $t \in [t_0, t_C)$. We will place some of the arcs of path $\mathcal{P}$ in groups (buckets) $I_1, I_2, \ldots, I_b$, such that all arcs in bucket I_i have been traversed between $2^{i-1}D$ and $2^i D$ times until step t. Formally, denote:

$$I_i = \left\{ l : d^{(t)}(e_l) \in [2^{i-1}D, 2^i D) \right\} \subseteq [1, D'], \quad \text{for } i \in [1, b].$$

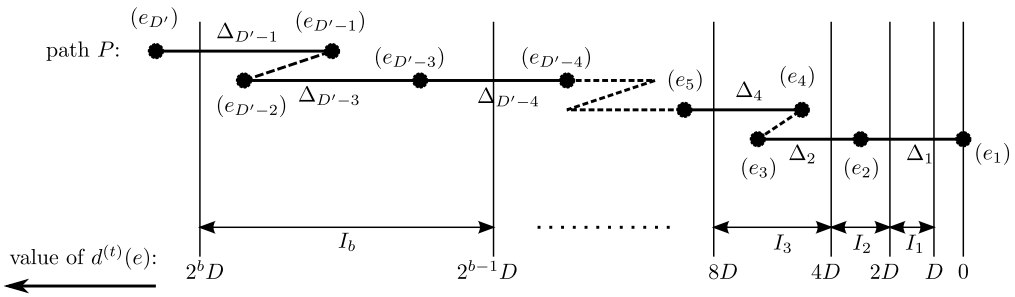


Figure 1 An illustration of sets I_i and Δ_l in the proof of Theorem 4.

We now analyze which buckets successive arcs of the path $\mathcal{P}$ fall into. For $l \in [1, D')$, define

$$\Delta_l = \begin{cases} [d^{(t)}(e_l), d^{(t)}(e_{l+1})), & \text{if } d^{(t)}(e_l) < d^{(t)}(e_{l+1}), \\ \emptyset, & \text{otherwise.} \end{cases}$$

Note that the union of all Δ_l covers the interval $[0, 2^bD)$, since for any $x \in [0, 2^bD)$ there exists $l^* \in [1, D')$ such that $x \in \Delta_{l^*}$ because $d^{(t)}(e_1) = 0$ and $d^{(t)}(e_{D'}) \geq 2^bD$ (see Fig. 1 for an illustration).

The intuition of the proof is now as follows: Since there are at most D' non-empty intervals Δ_l spanning the total range $[0, 2^bD)$ of all buckets $I_1, I_2, \ldots, I_b$, in a large number (linear in b) of these buckets I_i, the average length of an intervals Δ_l starting in bucket I_i will be at least $|I_i|b/D = 2^{i-1}b$, up to a constant factor. The existence of such long intervals Δ_l beginning in I_i will allow us to exploit Lemma 3 to show that arcs e_l, e_{l+1} differ in the number of traversals by a constant times $2^{i-1}b$. This implies that for the considered bucket indices i, the number of agents active at time t on edges from buckets $I_1, \ldots, I_i$ will be at least $2^{i-1}b$, up to constant factors and minor shifts at bucket boundaries. We now proceed to formalize the above arguments.

For $i \in [1, b]$, denote by $\mathcal{X}_i$ the set of intervals Δ_l beginning in bucket I_i: $\mathcal{X}_i = \bigcup_{l \in I_i} \Delta_l$. Consider any $x \in [0, 2^bD)$, and let l^* be such that $x \in \Delta_{l^*}$. We have $d^{(t)}(e_{l^*}) \leq x < 2^bD$, hence $l^* \in I_{i^*}$, for some $i^* \in [1, b]$, and $x \in \mathcal{X}_{i^*}$. It follows that:

$$[0, 2^bD) \subseteq \bigcup_{i \in [1,b]} \mathcal{X}_i. \tag{1}$$

For $i \in \mathbb{N}$, denote by $a_i^{(t)}$ the number of agents that traverse arcs from set $E^{(t)}_{2^iD}$ in step $t+1$, $a_i^{(t)} \equiv \sum_{e \in E^{(t)}_{2^iD}} a^{(t)}(e)$, and let $a_{-1}^{(t)} = 0$. (We remark that $E^{(t)}_{2^iD} \supseteq I_1 \cup \ldots \cup I_i$.) First, note that for all $i \in [1, b]$ and for $l \in I_i$, we have $d^{(t)}(e_l) < 2^iD$. So, by Lemma 3:

$$a_i^{(t)} \geq d^{(t)}(e_{l+1}) - d^{(t)}(e_l) - 1 = |\Delta_l| - 1 \quad \Longrightarrow \quad |\Delta_l| \leq a_i^{(t)} + 1. \tag{2}$$

Now, observe that for any $i \in [1, b]$:

$$\max \mathcal{X}_i = \max_{l \in I_i} (\max \Delta_l) \leq \max_{l \in I_i} \left(d^{(t)}(e_l) + |\Delta_l| - 1 \right) < 2^iD + a_i^{(t)}, \tag{3}$$

where we took into account inequality (2) and that $d^{(t)}(e_l) < 2^iD$ for $l \in I_i$.

Next, we will show that for all $i \in [1, b]$:

$$2^{i-1}D - a_{i-1}^{(t)} \leq |\mathcal{X}_i| \leq |I_i|(a_i^{(t)} + 1). \tag{4}$$

The right inequality in (4) is proved as follows: $|\mathcal{X}_i| \leq \sum_{l \in I_i} |\Delta_l| \leq |I_i|(a_i^{(t)} + 1)$, where the latter inequality is a consequence of (2).

We now prove the left inequality in (4). If $a_{i-1}^{(t)} \geq 2^{i-1}D$, then the bound is trivial. In the case when $a_{i-1}^{(t)} < 2^{i-1}D$, we will first prove that:

$$[2^{i-1}D + a_{i-1}, 2^i D) \subseteq \mathcal{X}_i. \tag{5}$$

To this end, take any $x \in [2^{i-1}D + a_{i-1}, 2^i D)$ and observe that by (1), there exists some $j \in [1, b]$ such that $x \in \mathcal{X}_j$. Moreover, note that:

1. For any $j < i$, $x \notin \mathcal{X}_j$, because, by (3), $\max \mathcal{X}_j < 2^j D + a_j^{(t)} \leq 2^{i-1}D + a_{i-1}^{(t)} \leq x$.
2. For any $j > i$, $x \notin \mathcal{X}_j$, because: $\min \mathcal{X}_j = \min_{l \in I_j, \Delta_l \neq \emptyset} \min \Delta_l = \min_{l \in I_j, \Delta_l \neq \emptyset} d^{(t)}(e_l) \geq 2^{j-1}D \geq 2^i D > x$.

Thus, $x \in \mathcal{X}_i$, and (5) follows. Equation (5) implies that $|\mathcal{X}_i| \geq 2^{i-1}D - a_{i-1}^{(t)}$, which completes the proof of (4). Next, by (4), $|I_i| \geq \frac{2^{i-1}D - a_{i-1}^{(t)}}{a_i^{(t)}+1}$ for all $i \in [1, b]$. The buckets $I_1, I_2, \ldots, I_b$ are pairwise disjoint by definition and contain at most D' elements altogether, which gives:

$$D + 2 \geq D' \geq \sum_{i=1}^{b} |I_i| \geq \sum_{i=1}^{b} \frac{2^{i-1}D - a_{i-1}^{(t)}}{a_i^{(t)} + 1} \geq \sum_{i=1}^{b} \frac{2^{i-1}D}{a_i^{(t)} + 1} - b,$$

where in the last inequality we used the fact that $a_i^{(t)} \geq a_{i-1}^{(t)}$ for $i \in [2, b]$. Dividing the sum in the last inequality by bD, we get the following expression for the arithmetic average:

$$\frac{1}{b} \sum_{i=1}^{b} \frac{2^{i-1}}{a_i^{(t)} + 1} \leq \frac{D + b + 2}{bD} = \frac{1}{b} + \frac{1 + 2/b}{D} < \frac{9.2}{b},$$

where in the last inequality we took into account that $k \leq 2^{16D}$ and $b \leq (\log k)/2$ by assumption, hence $D \geq (\log k)/16 \geq b/8$, and that $b = \lfloor (\log k)/2 \rfloor \geq 80$. All the elements of the considered sum are positive, hence by Markov's inequality, there exists a subset of indices $S^{(t)} \subseteq [1, b]$, with $|S^{(t)}| \geq b/2$, such that for all $j \in S^{(t)}$ we have:

$$\frac{2^{j-1}}{a_j^{(t)} + 1} \leq 2 \cdot \frac{1}{b} \sum_{i=1}^{b} \frac{2^{i-1}}{a_i^{(t)} + 1} \leq \frac{18.4}{b}.$$

This implies that for all $j \in S^{(t)}$:

$$a_j^{(t)} \geq \tfrac{b}{18.4} \cdot 2^{j-1} - 1 > \tfrac{b}{25} \cdot 2^{j-1}, \tag{6}$$

where we again took into account that $b \geq 80$. Fix $t_1 = \lceil 100mD/b \rceil$. We now prove that

$$t_C \leq t_0 + 2t_1 + 4m. \tag{7}$$

Suppose, by contradiction, that $t_C > t_0 + 2t_1 + 4m$. We will say that an index $j \in [1, b]$ is *good* after time t if $j \in S^{(t)}$. Since for all $t \in [t_0, t_C)$ we have $|S^{(t)}| \geq b/2$ and $S^{(t)} \subseteq [1, b]$, by the pigeon-hole principle there must exist an index j^* that is good in at least $(t_C - t_0)/2 = t_1 + 2m$ steps in $[t_0, t_C)$; we will call these steps *good steps*.

For an arc e of the graph, we denote by t_e the so called *exit time step* for arc e, after which the total number of visits to arc e of the graph for the first time exceeds $2^{j^*}D$: $d^{(t_e)}(e) \leq 2^{j^*}D < d^{(t_e+1)}(e)$. The set of all exit time steps, taken over all arcs of the graph,

is denoted $\hat{T} = \{t_e : e \in \vec{E}\}$. Note that $e \in E^{(t)}_{2^{j^*}D}$ if and only if $t \leq t_e$, and therefore we may write:

$$\sum_{t\in[0,t_C)\setminus\hat{T}} a^{(t)}_{j^*} = \sum_{t\in[0,t_C)\setminus\hat{T}} \sum_{e\in E^{(t)}_{2^{j^*}D}} a^{(t)}(e) \leq \sum_{e\in\vec{E}} \sum_{t=0}^{t_e-1} a^{(t)}(e) = \sum_{e\in\vec{E}} d^{(t_e)}(e) \leq 2m \cdot 2^{j^*} D. \quad (8)$$

Now, recall that there are at least $t_1 + 2m$ good time steps $t \in [t_0, t_C)$ for which index j^* satisfies (6), and that $|\hat{T}| \leq 2m$. It follows that:

$$\sum_{t\in[0,t_C)\setminus\hat{T}} a^{(t)}_{j^*} > t_1 \cdot \frac{b}{25} \cdot 2^{j^*-1} = \left\lceil \frac{100mD}{b} \right\rceil \frac{b}{25} \cdot 2^{j^*-1} \geq 2m \cdot 2^{j^*} D,$$

a contradiction with (8). Thus, we have proved (7). By (7), we obtain

$$t_C \leq t_0 + 2t_1 + 4m = \left\lceil \frac{2^{b+1}mD}{k} \right\rceil + 2\left\lceil \frac{100mD}{b} \right\rceil + 4m \leq$$
$$\leq \frac{mD}{\log k}\left(\frac{2^{b+1}\log k}{k} + \frac{200\log k}{b} + \frac{4\log k}{D} + \frac{3\log k}{mD}\right) \quad (9)$$

Taking into account that $b = \lfloor(\log k)/2\rfloor$, $k \leq 2^{16D}$, and $k > 2^{160}$, we obtain that the expression in the above bracket can be bounded by a constant, giving: $t_C < 500\frac{mD}{\log k}$. This completes the proof for the case $k > 2^{160}$.

Suppose now that $k \leq 2^{160}$. Yanovski *et al.* [16] showed that a single agent explores the graph in at most $2mD$ steps regardless of the initialization, and moreover, that adding agents cannot decrease the number of traversals on any edge. We thus trivially obtain the claim: $t_C \leq 2mD < 500\frac{mD}{\log k}$. ◀

We now consider the case when $k \geq 2^{16D}$. Here, we first make the additional assumption that each agent starts from a distinct node. We show that additional assumption implies that no arc is traversed by more than one agent in a single step. The proof then proceeds along similar lines as that of Theorem 4, and we show that in many time steps t, there exists a pair of arcs e_{l+1}, e_l in $\mathcal{P}$ with a large difference in the number of traversals up to time t. However, instead of counting the number of long arcs on path $\mathcal{P}$ belonging to a bucket I_i, in this proof we take advantage of the fact that the length of the path $D' \leq D + 2$ is small compared to $\log k$, which can be used to infer the existence of the sought arc pairs.

▶ **Lemma 5.** *Let $G = (V, E)$ be any undirected graph with arbitrary initialization of pointers and let D be the diameter of G. If $k \geq 2^{16D}$, then a team of k agents performing parallel rotor-router movement, with each agent starting from a distinct node of the graph, explores G in time $16mD/\log k$.*

It remains to consider the case not covered by the above lemma, when not all agents start from distinct positions. In fact, we will reduce such a case to the one already considered by making use of the concept of delayed deployments discussed in Section 2.

▶ **Lemma 6.** *Let R and R' be two starting configurations of the k-agent rotor-router system with cover times t_C and t'_C, respectively. Suppose that there exists a delayed deployment D of R whose execution transforms the starting configuration of R into the starting configuration of R' in $\hat{t}$ time steps. Then, $t_C \leq \hat{t} + t'_C$.*

The next lemma provides an upper bound on the time of transforming a rotor-router configuration with at most n agents into one in which agents occupy distinct starting nodes.

▶ **Lemma 7.** *For any initialization R of the rotor-router system with k agents, $k \leq n$, there exists a delayed deployment D of R which terminates in a configuration in which all agents occupy distinct positions after $\hat{t} \leq k^4$ steps.*

When $1 < k \leq \lceil n^{1/5} \rceil$, we can bound the time $\hat{t}$ in the above lemma as: $\hat{t} \leq k^4 \leq 32n/k \leq 64m/k \leq 128\frac{mD}{\log k}$.

Combining the above result with Lemmas 5 and 6, we obtain that for any rotor router initialization with k agents, $k \leq \lceil n^{1/5} \rceil$ and $k \geq 2^{16D}$, exploration is completed within time $t_C = \hat{t} + t'_C \leq 128\frac{mD}{\log k} + 16\frac{mD}{\log k} = 144\frac{mD}{\log k}$. On the other hand, when $k < 2^{16D}$, by Theorem 4, the cover time is $t_C \leq 500\frac{mD}{\log k}$. It follows that the bound $t_C \leq 500\frac{mD}{\log k}$ holds for all starting configurations with $k \leq \lceil n^{1/5} \rceil$.

When $k > \lceil n^{1/5} \rceil$, we can make use of a result of Yanovski *et al.* [16], stating that the worst-case initialization of a rotor-router system with k agents cannot have greater cover time than the worst-case initialization of a system with $k' < k$ agents. Putting $k' = \lceil n^{1/5} \rceil$, for any $k > \lceil n^{1/5} \rceil$ we obtain: $t_C \leq 500\frac{mD}{\log k'} \leq 2500\frac{mD}{\log n}$. Finally, combining the results for $k \leq \lceil n^{1/5} \rceil$ and $k > \lceil n^{1/5} \rceil$ gives the following theorem.

▶ **Theorem 8.** *Let $G = (V, E)$ be any undirected graph with arbitrary initialization of pointers and let D be the diameter of G. A team of k agents performing in parallel the rotor-router movement explores G in time $\max\{500mD/\log k, 2500mD/\log n\}$, regardless of the initial positions of agents. In particular, if $k \leq n^c$ for some $c > 0$, then the cover time is at most $2500c \cdot mD/\log k$.* ◀

Theorems 4 and 8 imply that the cover time of the rotor-router is $O(mD/\log k)$ for all graphs, whenever $k \in 2^{O(D)}$ or $k \in O(poly(n))$.

4 Lower bound on cover time

▶ **Theorem 9.** *Let $G = (V, E)$ be any undirected graph of diameter D. There exists a port labeling of the edges of G, an initialization of pointers and an assignment of starting positions to a team of k agents, such that the exploration performed in parallel with the rotor-router movement has cover time $t_C \geq \frac{1}{4}mD/k$.*

Proof. If $k > m$, we make all agents start from an arbitrarily chosen single node, and choose an arbitrary pointer initialization. In such scenario, the exploration will be completed after time at least $D > \frac{mD}{k}$. Thus, we can safely assume that $k \leq m$.

For any graph $G = (V, E)$, as shown in [3, Theorem 2], there exists a partition of the edge set $E = E_1 \cup E_2$, such that:

- **(i)** $|E_1| \geq \frac{m}{2}$,
- **(ii)** there exist $V_1 \subseteq V$ and $V_2 \subseteq V$ such that the subgraphs $H_1 = G[V_1]$ and $H_2 = G[V_2]$ are connected and their edge sets are E_1 and E_2, respectively,
- **(iii)** there exists a node $v \in V_2$ being at distance at least $\frac{D}{2}$ from each node of H_1.

Denote by $F \subset E_2$ the set of edges incident to some node from H_1. Now, let $C = \{e_1, e_2, \ldots, e_{2|E_1|}\}$ be a directed Eulerian cycle in $\vec{H}_1$ (the bidirected subgraph corresponding to H_1) traversing every edge in E_1 exactly once in each direction. To simplify notation, let $\Delta = \left\lfloor \frac{2|E_1|}{k} \right\rfloor$. We choose an arbitrary set of indexes $1 = j_1 < j_2 < \ldots < j_k \leq 2|E_1|$ such that they are spread (almost-)equidistantly in $\{1, \ldots, 2|E_1|\}$, that is:

$$\forall_{1 \leq i < k} \quad j_{i+1} - j_i \in \{\Delta, \Delta + 1\} \quad \text{and} \quad j_1 - j_k + 2|E_1| \in \{\Delta, \Delta + 1\}\,.$$

This is possible because, due to 1, $2|E_1| \geq k$. We partition E_1 into Δ sets $S_1, \ldots, S_\Delta$ of size k:

$$S_{i+1} = \{e_{j_1+i}, e_{j_2+i}, \ldots, e_{j_k+i}\}, \quad \text{for } 0 \leq i < \Delta,$$

and one set for all remaining edges: $R = E_1 \setminus \bigcup_{t=1}^{\Delta} S_t$.

We choose the starting positions of k agents, the port assignment, and the initialization of pointers for the edges in E_1 such that in their first $\Delta + 1$ steps, the k agents traverse all edges in E_1 in the following delayed deployment: for each $t \in \{1, \ldots, \Delta\}$, in the t-th step, exactly the edges in S_t are traversed, whereas in the $(\Delta+1)$-th step we delay some agents so that exactly the edges in R are traversed. We achieve this by setting outgoing ports so that, for every node u in H_1, we order the edges in E_1 incident to u by assigning smaller ports to edges in S_t than to the edges in S_{t+1}, for each $t \in \{1, \ldots, \Delta\}$, where $S_{\Delta+1} = R$. Such a port ordering is enough to explore the graph H_1, with delayed deployment, with the property that every edge is visited once every $\Delta + 1$ steps.

Now we assign ports to the edges in F. To this end, we consider the subgraph of G, denoted by $\widetilde{G}$, consisting of the edges in $E_1 \cup F$. In other words, we take H_1 (together with the port assignment obtained above) and we add the edges in F, obtaining $\widetilde{G}$. Note that, by 2, each edge in F has one endpoint in V_1 and the other endpoint in $V \setminus V_1$. The ports in F are determined by analyzing the behavior of agents in the graph $\widetilde{G}$ in the delayed deployment described above. Whenever any set of agents are about to leave H_1 and traverse any edge from F, we select a single agent in a deterministic way (for example, by choosing the agent located on a node with the smallest index, having indexes assigned to nodes). We stop all other agents and perform traversals only with the selected agent, until it returns to H_1. We set the ports of the edges in F so that whenever an agent leaves H_1 through an edge $(v \to u) \in F$ $(v \in V_1, u \notin V_1)$, it returns to H_1 through the edge $(u \to v)$ (we call this property *the property of return*). Having the property of return, we achieve that the agents patrol E_1, and whenever an agent is about to leave H_1, the other agents are delayed until the agent returns to the same node. Since the selection of agents is done deterministically, the edges in F are always traversed in separated periods of time (when one agent is traversing edges from F, all other agents are stopped) in a cyclic fashion, i.e., the sequence of traversal of the edges in F is $(f_1, f_1', f_2, f_2', \ldots, f_{|F|}, f_{|F|}')^*$, where f' means the reversed edge to an edge f, i.e., if $f = (u \to v)$, then $f' = (v \to u)$. Denote $f_i = (u_i \to v_i)$ for each $i \in \{1, \ldots, |F|\}$.

It remains to assign port labels to the edges in $E_2 \setminus F$, and to initialize the pointers for the nodes in $V \setminus V(\widetilde{G})$. This is done by first constructing a multigraph G' and then by analyzing a single agent movement in G'. The node set of G' is $\{h\} \cup (V \setminus V_1)$. For each $(u \to v) \in E_2 \setminus F$, let $(u \to v)$ be an edge of G', and for each $i \in \{1, \ldots, |F|\}$, let (h, v_i) and (v_i, h) be the edges of G'. In other words, we construct G' by taking G, leaving the edges in $E \setminus E_1$ untouched, and contracting (identifying) the nodes of H_1 into the single node h. (The loops at h formed by the edges in E_1 are discarded.) For each $i \in \{1, \ldots .|F|\}$, the ports of $(h \to v_i)$ and $(v_i \to h)$ equal the ports of (u_i, v_i) and (v_i, u_i), respectively.

We set the remaining ports in G' and pointer initialization so that a single agent that starts at h explores G' in the following way:

(a) The edges in F are traversed according to the order

$$\big((h \to v_1), (v_1 \to h), (h \to v_2), (v_2 \to h), \ldots, (h \to v_{|F|}), (v_{|F|} \to h)\big).$$

Later on, we use the port labeling of G' to assign port labels to the edges in E_2 in G, and the above allows us to maintain the return property in G.

(b) The agent requires $D/2$ traversals through at least one edge in F (and $D/2 - 1$ through every other edge from F). This follows from the fact that, due to 3, there exists a node in G' being at distance at least $D/2$ from h.

The above process assigns port labels to the edges in E_2 and sets initial values of all pointers in G', which completes the construction of G and the initial setup of the rotor-router.

Now we analyze the delayed deployment performed by the k agents in G. We divide the exploration of G into phases. The i-th phase starts in the step in which each edge in S_1 is traversed for the i-th time, and ends in the step preceding the beginning of the $(i+1)$-th stage. Note that each stage contains at least Δ steps in which all agent move simultaneously. By 3a, the property of return holds in G, and therefore each edge in F is traversed exactly once in each of the phases except the 1st phase. (In the 1st phase, agents only traverse edges from E_1.) Thus, by 3b, at least $D/2 - 1 + 1$ full phases are required in the delayed deployment to explore G (not counting the very last, partial phase in which the exploration of last vertex happens, but counting the initial phase in which no edges from F are traversed). This means that we need τ steps in which all agents move simultaneously to fully explore the graph G, where:

$$\tau \geq \Delta \cdot D/2 = \left\lfloor \tfrac{2|E_1|}{k} \right\rfloor \cdot D/2 \geq \left\lfloor \tfrac{m}{k} \right\rfloor \cdot D/2 \geq \tfrac{1}{4} mD/k \,.$$

We can now apply Lemma 1 for the considered deployment, obtaining that the cover time of G is $t_C \geq \tau \geq \frac{1}{4} mD/k$. ◀

The bound in Theorem 9 is asymptotically tight, e.g., for the class of stars.

▶ Proposition 10. Let G be a star on n nodes. A team of $k \leq n$ agents covers G in time $t_C \leq 2\lceil n/k \rceil$, for any initialization of the rotor-router and any initial positions of agents. ◀

References

1 Hoda Akbari and Petra Berenbrink. Parallel rotor walks on finite graphs and applications in discrete load balancing. In *SPAA*, pages 186–195, 2013.

2 Noga Alon, Chen Avin, Michal Koucký, Gady Kozma, Zvi Lotker, and Mark R. Tuttle. Many random walks are faster than one. *Combinatorics, Probability & Computing*, 20(4):481–502, 2011.

3 Evangelos Bampas, Leszek Gasieniec, Nicolas Hanusse, David Ilcinkas, Ralf Klasing, and Adrian Kosowski. Euler tour lock-in problem in the rotor-router model. In *DISC*, pages 423–435, 2009.

4 Evangelos Bampas, Leszek Gasieniec, Ralf Klasing, Adrian Kosowski, and Tomasz Radzik. Robustness of the rotor-router mechanism. In *OPODIS*, volume 5923 of *LNCS*, pages 345–358, 2009.

5 S. N. Bhatt, S. Even, D. S. Greenberg, and R. Tayar. Traversing directed eulerian mazes. *J. Graph Algorithms Appl.*, 6(2):157–173, 2002.

6 Andrei Z. Broder, Prabhakar Raghavan, Robert W. Taylor, Anna R. Karlin, Anna R. Karlin, Eli Upfal, and Eli Upfal. Trading space for time in undirected s-t connectivity. In *In Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 543–549, 1991.

7 J. N. Cooper and J. Spencer. Simulating a random walk with constant error. *Combinatorics, Probability & Computing*, 15(6):815–822, 2006.

8 B. Doerr and T. Friedrich. Deterministic random walks on the two-dimensional grid. *Combinatorics, Probability & Computing*, 18(1-2):123–144, 2009.

9 Klim Efremenko and Omer Reingold. How well do random walks parallelize? In *APPROX-RANDOM*, pages 476–489, 2009.

10 Robert Elsässer and Thomas Sauerwald. Tight bounds for the cover time of multiple random walks. *Theor. Comput. Sci.*, 412(24):2623–2641, 2011.

11 Uriel Feige. A Spectrum of Time–Space Trade-offs for Undirected s-t Connectivity. *Journal of Computer and System Sciences*, 54(2):305 – 316, 1997.

12 Tobias Friedrich and Thomas Sauerwald. The cover time of deterministic random walks. In *COCOON*, volume 6196 of *LNCS*, pages 130–139, 2010.

13 Ralf Klasing, Adrian Kosowski, Dominik Pajak, and Thomas Sauerwald. The multi-agent rotor-router on the ring: a deterministic alternative to parallel random walks. In *PODC*, pages 365–374, 2013.

14 V.B. Priezzhev, D. Dhar, A. Dhar, and S. Krishnamurthy. Eulerian walkers as a model of self-organized criticality. *Phys. Rev. Lett.*, 77(25):5079–5082, Dec 1996.

15 I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Trans. Robotics and Automation*, 15:918–933, 1999.

16 V. Yanovski, I. A. Wagner, and A. M. Bruckstein. A distributed ant algorithm for efficiently patrolling a network. *Algorithmica*, 37(3):165–186, 2003.

Packing a Knapsack of Unknown Capacity

Yann Disser*, **Max Klimm**, **Nicole Megow**†, **and Sebastian Stiller**

Department of Mathematics, Technische Universität Berlin, Germany
{disser,klimm,nmegow,stiller}@math.tu-berlin.de

Abstract

We study the problem of packing a knapsack without knowing its capacity. Whenever we attempt to pack an item that does not fit, the item is discarded; if the item fits, we have to include it in the packing. We show that there is always a policy that packs a value within factor 2 of the optimum packing, irrespective of the actual capacity. If all items have unit density, we achieve a factor equal to the golden ratio $\varphi \approx 1.618$. Both factors are shown to be best possible.

In fact, we obtain the above factors using packing policies that are *universal* in the sense that they fix a particular order of the items and try to pack the items in this order, independent of the observations made while packing. We give efficient algorithms computing these policies. On the other hand, we show that, for any $\alpha > 1$, the problem of deciding whether a given universal policy achieves a factor of α is coNP-complete. If α is part of the input, the same problem is shown to be coNP-complete for items with unit densities. Finally, we show that it is coNP-hard to decide, for given α, whether a set of items admits a universal policy with factor α, even if all items have unit densities.

1998 ACM Subject Classification F.2.2 Sequencing and Scheduling

Keywords and phrases Knapsack, unknown capacity, robustness, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.276

1 Introduction

In the standard knapsack problem we are given a set of items, each associated with a size and a value, and a capacity of the knapsack. The goal is to find a subset of the items with maximum value whose size does not exceed the capacity. In this paper, we study the *oblivious* knapsack problem where the capacity of the knapsack is not given. Whenever we try to pack an item, we observe whether or not it fits the knapsack. If it does, the item is packed into the knapsack and cannot be removed later. If it does not fit, we discard it and continue packing with the remaining items. The central question of this paper is how much we loose by not knowing the capacity, in the worst case. The oblivious variant of the knapsack problem naturally arises whenever items are prioritized by a different entity or at a different time than the actual packing of the knapsack. For example, waiting lists for stand-by capacities and services, e.g., on an airplane, have to be fixed before the exact amount of free seats due to no-shows is known. The order of the list must be determined based only on the size of the items (e.g., the sizes of groups of travelers) and their value (e.g., compensation for service denial).

A solution to the oblivious knapsack problem is a policy that governs the order in which we attempt to pack the items, depending only on the observation which of the previously attempted items did fit into the knapsack and which did not. In other words, a policy is a

* Supported by the Alexander von Humboldt Foundation.
† Supported by the German Science Foundation (DFG) under contract ME 3825/1.



© Yann Disser, Max Klimm, Nicole Megow, and Sebastian Stiller;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 276–287



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



binary decision tree with the item that is tried first at its root. The two children of the root are the items that are tried next, which of the two depends on whether or not the first item fits the knapsack, and so on. We aim for a solution that is good for *every* possible capacity, compared to the best solution of the standard knapsack problem for this capacity. Formally, a policy has *robustness factor* α if, for any capacity, packing according to the policy results in a value that is at least a $1/\alpha$-fraction of the optimum value for this capacity.

We show that the oblivious knapsack problem always admits a robustness factor of 2. In fact, this robustness factor can be achieved with a policy that packs the items according to a fixed order, irrespective of the observations made while packing. Such a policy is called *universal*. We provide an algorithm that computes a 2-robust, universal policy in time $\Theta(n \log n)$ for a given set of n items. We complement this result by showing that no robustness factor better than 2 can be achieved in general, even by policies that are not universal. In other words, the cost of not knowing the capacity is exactly 2.

We give a different efficient algorithm for the case that all items have unit density, i.e., size and value of each item coincide. This algorithm produces a universal policy with a robustness factor of at most the golden ratio $\varphi \approx 1.618$. Again, we show that no better robustness factor can be achieved in general, even by policies that are not universal.

While good universal policies can be found efficiently, it is intractable to compute the robustness factor of a *given* universal policy and it is intractable to compute the best robustness factor an instance admits. Specifically, we show that, for any fixed $\alpha \in (1, \infty)$, it is coNP-complete to decide whether a given universal policy is α-robust. For unit densities we establish a slightly weaker hardness result by showing that it is coNP-complete to decide whether a given universal policy achieves a *given* robustness factor α. Finally, we show that, for given α, it is coNP-hard to decide whether an instance of the oblivious knapsack problem admits a universal policy with robustness factor α, even when all items have unit density.

Related work

The knapsack problem has been studied for different models of imperfect information. In the *stochastic* knapsack problem, sizes and values of the items are random variables. It is known that a policy maximizing the expected value is PSPACE-hard to compute, see Dean et al. [7]. The authors assume that the packing stops when the first item does not fit the knapsack, and give a universal policy that approximates the value obtained by an optimal, not necessarily universal, policy by a factor of 2. They also provide a non-universal policy within a factor of $3 + \epsilon$ of the optimal policy. Bhalgat et al. [3] give an algorithm with an improved approximation guarantee of $8/3 + \epsilon$. They also give a PTAS for the case that it is allowed to violate the capacity of the knapsack by a factor of $1 + \epsilon$.

In *robust* knapsack problems, a set of possible scenarios for the sizes and values of the items is given. Yu [23], Bertsimas and Sim [2], Goetzmann et al. [11], and Monaci and Pferschy [18] study the problem of maximizing the worst-case value of a knapsack under various models. Büsing et al. [5] and Bouman et al. [4] study the problem from a computational point of view. Both allow for an adjustment of the solution after the realization of the scenario. Similar to our model, Bouman et al. consider uncertainty in the capacity.

The notion of a *robustness factor* that we adopt in this work is due to Hassin and Rubinstein [12] and is defined as the worst-case ratio of solution and optimum, over all realizations. Kakimura et al. [14] analyze the complexity of deciding whether an α-robust solution exists for a knapsack instance with an unknown bound on the number of items that can be packed. Megow and Mestre [16] study a variant of the knapsack problem with unknown capacity closely related to ours. In contrast to our model, they assume that the

packing stops once the first item does not fit the remaining capacity. In this model, a universal policy with a constant robustness factor may fail to exist, and, thus, Megow and Mestre resort to *instance-sensitive* performance guarantees. They provide a PTAS that constructs a universal policy with robustness factor arbitrarily close to the best possible robustness factor for every particular instance.

The concept of *obliviousness* is used in various other contexts (explicitly or implicitly), such as hashing (Carter and Wegman [6]), caching (Frigo et al. [10], Bender et al. [1]) routing (Valiant and Brebner [22], Räcke [20]), TSP (Papadimitriou [19], Deineko et al. [8], Jia et al, [13]), Steiner tree and set cover (Jia et al, [13]), and scheduling (Epstein et al. [9], Megow and Mestre [16]). In all of these works, the general idea is that specific parameters of a problem instance are unknown, e.g., the cache size or the set of vertices to visit in a TSP tour, and the goal is to find a *universal solution* that performs well for all realizations of the hidden parameters.

Universal policies for the oblivious knapsack problem play a role in the design of public key cryptosystems. One of the first such systems – the Merkle–Hellman knapsack cryptosystem [17] – is based on particular instances that allow for a 1-robust universal policy for the oblivious knapsack problem. The basic version of this cryptosystem can be attacked efficiently, e.g., by the famous attack of Shamir [21]. This attack uses the fact that the underlying knapsack instance has exponentially increasing item sizes. A better understanding of universal policies may help to develop knapsack-based cryptosystems that avoid the weaknesses of Merkle and Hellman's.

2 Preliminaries

An instance of the *oblivious knapsack problem* is given by a set of n items $\mathcal{I}$, where each item $i \in \mathcal{I}$ has a non-negative *value* $v(i) \in \mathbb{Q}_{\geq 0}$ and a strictly positive *size* $l(i) \in \mathbb{Q}_{>0}$. For a subset $S \subseteq \mathcal{I}$ of items, we write $v(S) = \sum_{i \in S} v(i)$ and $l(S) = \sum_{i \in S} l(i)$ to denote its total value and total size, respectively, of the items in S. A *solution* for instance $\mathcal{I}$ is a policy $\mathcal{P}$ that governs the order in which the items are considered for packing into the knapsack. The policy must be independent of the capacity of the knapsack, but the choice which item to try next may depend on the observations which items did and which items did not fit the knapsack so far. Formally, a solution policy is a binary decision tree that contains every item exactly once along each path from the root to a leaf. The *packing* $\mathcal{P}(C) \subseteq \mathcal{I}$ of $\mathcal{P}$ for a fixed capacity C is obtained as follows: We start with $\mathcal{P}(C) = \emptyset$ and check whether the item r at the root of $\mathcal{P}$ fits the knapsack, i.e., whether $l(r) + l(\mathcal{P}(C)) \leq C$. If the item fits, we add r to $\mathcal{P}(C)$ and continue packing recursively with the left subtree of r. Otherwise, we discard r and continue packing recursively with the right subtree of r.

A *universal policy* Π for instance $\mathcal{I}$ is a policy that does not depend on observations made while packing, i.e., the decision tree for a universal policy has a fixed permutation of the items along every path from the root to a leaf. We identify a universal policy with this fixed permutation and write $\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n)$. Analogously to general policies, the packing $\Pi(C) \subseteq \mathcal{I}$ of a universal policy Π for capacity $C \leq l(\mathcal{I})$ is obtained by considering the items in the order given by the permutation Π and adding every item if it does not exceed the remaining capacity. We measure the quality of a policy for the oblivious knapsack problem by comparing its packing with the optimal packing for each capacity. More precisely, a policy $\mathcal{P}$ for instance $\mathcal{I}$ is called *α-robust for capacity C*, $\alpha \geq 1$, if it holds that $v(\textsc{Opt}(\mathcal{I}, C)) \leq \alpha \cdot v(\mathcal{P}(C))$, where $\textsc{Opt}(\mathcal{I}, C)$ denotes an optimal packing for capacity C. We say $\mathcal{P}$ is *α-robust* if it is α-robust for all capacities. In this case, we call α the *robustness factor* of policy $\mathcal{P}$.

3 Solving the Oblivious Knapsack Problem

In this section, we describe an efficient algorithm that constructs a universal policy for a given instance of the oblivious knapsack problem. The solution produced by our algorithm is guaranteed to pack at least half the value of the optimal solution for any capacity C. We show that this is the best possible robustness factor.

The analysis of our algorithm relies on the classical *modified greedy* algorithm (cf. [15]). We compare the packing of our policy, for each capacity, to the packing obtained by the modified greedy algorithm instead of the actual optimum. As the modified greedy is a 2-approximation, to show that our policy is 2-robust it is sufficient to show that its packing is never worse than the one obtained by the modified greedy algorithm. We briefly review the modified greedy algorithm.

Let $d(i) = v(i)/l(i)$ denote the *density* of item i. The modified greedy algorithm (MGreedy) for a set of items $\mathcal{I}$ and known knapsack capacity C first discards all items that are larger than C from $\mathcal{I}$. The remaining items are sorted in non-increasing order of their densities, breaking ties arbitrarily. The algorithm then either takes the longest prefix P of the resulting sequence that still fits into capacity C, or the first item s that does not fit anymore, depending on which of the two has a greater value. In the latter case, we say that s is a *swap item* (for capacity C) and that C is a *swap capacity*. In both cases, we refer to P as the *greedy set* for capacity C. See Algorithm 1 for a formal description.

For our analysis, it is helpful to fix the tie-breaking rule under which MGreedy initially sorts the items. To this end, we assume that there is a bijection $t : \mathcal{I} \to \{1, 2, \dots, n\}$, that maps every item $i \in \mathcal{I}$ to a *tie-breaking index* $t(i)$, and that the modified greedy algorithm initially sorts the items decreasingly with respect to the tuple $\tilde{d}(\cdot) = (d(\cdot), t(\cdot))$, i.e., the items are sorted non-increasingly by density and whenever two items have the same density, they are sorted by decreasing tie-breaking index. In the following, for two items i, j, we write $\tilde{d}(i) \succ \tilde{d}(j)$ if and only if $d(i) > d(j)$, or $d(i) = d(j)$ and $t(i) > t(j)$, and say that i has higher density than j.

We evaluate the quality of our universal policy by comparing it for every capacity with the solution of MGreedy. This analysis suffices because of the following well-known property of the modified greedy algorithm.

▶ **Theorem 1** (cf. [15]). *For every instance $(\mathcal{I}, C)$ of the standard knapsack problem with known capacity, $v(\textsc{Opt}(\mathcal{I}, C)) \leq 2 \cdot v(\text{MGreedy}(\mathcal{I}, C))$.*

We are now ready to describe our algorithm Universal (Algorithm 2) that produces a universal policy tailored to imitate the behavior of MGreedy without knowing the capacity.

Algorithm 1: MGreedy($\mathcal{I}$, C)

Input: set of items $\mathcal{I}$, capacity C
Output: subset $S \subseteq \mathcal{I}$ such that $l(S) \leq C$ and $v(S) \geq v(\textsc{Opt}(\mathcal{I}, C))/2$
$D \leftarrow \langle$items in $\{i \in \mathcal{I} \mid l(i) \leq C\}$ sorted decreasingly by density $\tilde{d}\rangle$
$k \leftarrow \max\{j \mid l(\{D_1, \dots, D_j\}) \leq C\}$
$P \leftarrow (D_1, \dots, D_k)$, $s \leftarrow D_{k+1}$
if $v(P) \geq v(s)$ **then**
  return P
else
  return $\{s\}$

Algorithm 2: UNIVERSAL($\mathcal{I}$)

Input: set of items $\mathcal{I}$
Output: sequence of items Π
$L \leftarrow \langle$items in $\mathcal{I}$ sorted by non-decreasing size$\rangle$
$\Pi^{(0)} \leftarrow \emptyset$
for $r \leftarrow 1, \dots, n$ **do**
  if L_r *is a swap item* **then**
    $\Pi^{(r)} \leftarrow (L_r, \Pi^{(r-1)})$
  else
    $j \leftarrow 1$
    while $j \leq |\Pi^{(r-1)}|$ **and** $\tilde{d}(\Pi_j^{(r-1)}) \succ \tilde{d}(L_r)$ **do**
      $j \leftarrow j + 1$
    $\Pi^{(r)} \leftarrow (\Pi_1^{(r-1)}, \dots, \Pi_{j-1}^{(r-1)}, L_r, \Pi_j^{(r-1)}, \dots)$
return $\Pi^{(n)}$

First, UNIVERSAL determines which items are swap items. It then starts with an empty permutation, and considers the items in order of non-decreasing sizes, inserting each item into the permutation. Swap items are always placed in front of all items already in the permutation, and all other items are inserted in front of the first item in the permutation that has a lower density.

We prove the following result.

▶ **Theorem 2.** *The algorithm* UNIVERSAL *constructs a universal policy of robustness factor* 2*.*

Before we prove this theorem, we first analyze the structure of the permutation output by UNIVERSAL in terms of density, size, and value. First, we prove that every item following a non-swap item has lower density.

▶ **Lemma 3.** *For a sequence* Π *returned by* UNIVERSAL*, we have* $\tilde{d}(\Pi_k) \succ \tilde{d}(\Pi_{k+1})$ *for every non-swap item* Π_k*,* $1 \leq k < n$*.*

Proof. For $j \in \{k, k+1\}$, let $r(j) \in \{1, \dots, n\}$ be the index of the iteration in which UNIVERSAL inserts Π_j into Π. We distinguish two cases.

If $r(k) < r(k+1)$, then the item Π_{k+1} cannot be a swap item, since it would appear in front of the item Π_k if it was. As each non-swap item is inserted into Π such that all items left of it are larger with respect to $\tilde{d}$, the claim follows.

If $r(k) > r(k+1)$, since it is not a swap item, Π_k is put in front of Π_{k+1} because it has a higher density. ◀

We prove that no item preceding a swap item has smaller size.

▶ **Lemma 4.** *For a permutation* Π *returned by* UNIVERSAL*, we have* $l(\Pi_j) \geq l(\Pi_k)$ *for every swap item* $\Pi_k, 1 < k \leq n$*, and every other item* $\Pi_j, 1 \leq j < k$*.*

Proof. Since Π_k is a swap item, it stands in front of all items inserted earlier into Π. Hence, all items that appear in front of Π_k in Π have been inserted in a later iteration of UNIVERSAL. Since UNIVERSAL processes items in order of non-decreasing sizes, we have $l(\Pi_j) \geq l(\Pi_k)$. ◀

We prove that no item preceding a swap item has smaller value.

▶ **Lemma 5.** *For a permutation* Π *returned by* Universal, *we have* $v(\Pi_j) \geq v(\Pi_k)$ *for every swap item* $\Pi_k, 1 < k \leq n$, *and every other item* $\Pi_j, 1 \leq j < k$.

Proof. We distinguish three cases.

First case: Π_j is a swap item and $\tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)$. By Lemma 4, we have $l(\Pi_j) \geq l(\Pi_k)$, and the claim trivially holds.

Second case: Π_j is a swap item and $\tilde{d}(\Pi_j) \prec \tilde{d}(\Pi_k)$. Since Π_j is a swap item, there is a capacity $C \geq l(\Pi_j)$ such that

$$v(\Pi_j) > v(\{i \in \mathcal{I} \mid l(i) \leq C \text{ and } \tilde{d}(i) \succ \tilde{d}(\Pi_j)\}).$$

In particular, for $C = l(\Pi_j)$ we obtain

$$v(\Pi_j) > v(\{i \in \mathcal{I} \mid l(i) \leq l(\Pi_j) \text{ and } \tilde{d}(i) \succ \tilde{d}(\Pi_j)\}). \tag{1}$$

Since, by Lemma 4, $l(\Pi_j) \geq l(\Pi_k)$, the item Π_k is included in the set on the right hand side of (1). We conclude that $v(\Pi_j) > v(\Pi_k)$.

Third case: Π_j is not a swap item. Let $\Pi_{j'}$ be the first swap item after Π_j in Π, i.e.,

$$j' = \min\{i \in \{j+1, \dots, k\} \mid \Pi_i \text{ is a swap item } \}.$$

Note that the minimum is attained as Π_k is a swap item. The analysis of the first two cases implies that $v(\Pi_{j'}) \geq v(\Pi_k)$. By Lemma 3 we have $\tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_{j+1}) \succ \cdots \succ \tilde{d}(\Pi_{j'})$, and by Lemma 4 we have $l(\Pi_j) \geq l(\Pi_{j'})$. Hence, $v(\Pi_j) \geq v(\Pi_{j'}) \geq v(\Pi_k)$. ◀

Finally, the next lemma gives a legitimation for the violation of the density order in the output permutation. Essentially, whenever an item precedes denser items, we guarantee that it is worth at least as much as all of them combined.

▶ **Lemma 6.** *For a permutation* Π *returned by* Universal, *we have*

$$v(\Pi_k) \geq v\big(\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\}\big)$$

for every item $\Pi_k, 1 \leq k < n$.

Proof. We distinguish whether Π_k is a swap item, or not.

If Π_k is a swap item, by definition, Π_k is worth more than the greedy set for some capacity $C \geq l(\Pi_k)$. Thus,

$$\begin{aligned} v(\Pi_k) &> v\big(\{\Pi_j \mid l(\Pi_j) \leq C \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\}\big) \\ &\geq v\big(\{\Pi_j \mid l(\Pi_j) \leq l(\Pi_k) \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\}\big). \end{aligned}$$

Since items whose size is strictly larger than $l(\Pi_k)$ are inserted into Π at a later iteration of Universal, they can only end up behind Π_k if they are smaller with respect to $\tilde{d}$. Hence,

$$\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\} \subseteq \{\Pi_j \mid l(\Pi_j) \leq l(\Pi_k) \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\},$$

and thus $v(\Pi_k) > v(\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\})$, as claimed.

If, on the other hand, Π_k is not a swap item, let $\Pi_{k'}$ be the first swap item after it in Π. If no such item exists, the claim holds by Lemma 3, since

$$\big\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\big\} = \emptyset.$$

Otherwise, by Lemma 3, we obtain $\tilde{d}(\Pi_k) \succ \tilde{d}(\Pi_{k+1}) \succ \cdots \succ \tilde{d}(\Pi_{k'})$ and hence

$$\begin{aligned}\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\} &= \{\Pi_j \mid j > k' \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\} \\ &\subseteq \{\Pi_j \mid j > k' \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_{k'})\}.\end{aligned}$$

Consequently, and by the argument above for swap items,

$$v(\Pi_{k'}) > v(\{\Pi_j \mid j > k' \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_{k'})\}) \geq v(\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) > \tilde{d}(\Pi_k)\})).$$

Finally, by Lemma 5, we have $v(\Pi_k) \geq v(\Pi_{k'}) \geq v(\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\})$. ◀

We now prove Theorem 2.

Proof of Theorem 2. We show that for every item set $\mathcal{I}$, the permutation $\Pi = \textsc{Universal}(\mathcal{I})$ satisfies $v(\textsc{Opt}(\mathcal{I}, C)) \leq 2v(\Pi(C))$ for every capacity $C \leq l(\mathcal{I})$. By Theorem 1, it suffices to show $v(\Pi(C)) \geq v(\textsc{MGreedy}(\mathcal{I}, C))$ for all capacities. We distinguish between swap capacities and capacities where MGreedy outputs a greedy set.

First, assume that C is a swap capacity, and let $\{\Pi_k\} = \textsc{MGreedy}(\mathcal{I}, C)$ be the swap item returned by the modified greedy algorithm. Then, $\Pi(C)$ contains at least one item Π_j with $j \leq k$. By Lemma 5, we have $v(\Pi(C)) \geq v(\Pi_j) \geq v(\Pi_k) = v(\textsc{MGreedy}(\mathcal{I}, C))$.

Now assume that C is not a swap capacity. Let $G^+ = \textsc{MGreedy}(\mathcal{I}, C) \setminus \Pi(C)$ be the set of items in the greedy set for capacity C that are not packed by the permutation Π. Similarly, let $U^+ = \Pi(C) \setminus \textsc{MGreedy}(\mathcal{I}, C)$. If $G^+ = \emptyset$, then $v(\Pi(C)) \geq v(\textsc{MGreedy}(\mathcal{I}, C))$ and we are done. Suppose now that $G^+ \neq \emptyset$. Then, also $U^+ \neq \emptyset$, since $\Pi(C)$ is inclusion maximal. For all items $i \in U^+$, we have $l(i) \leq C$ and $i \notin \textsc{MGreedy}(\mathcal{I}, C)$. Since C is not a swap capacity, $\textsc{MGreedy}(\mathcal{I}, C)$ is the greedy set for capacity C, and thus $\tilde{d}(i) \prec \tilde{d}(i')$ for all $i \in U^+$ and $i' \in G^+$. By definition of $\Pi(C)$ and since $U^+ \neq \emptyset$, we also have $k = \min\{j \mid \Pi_j \in U^+\} < \min\{k' \mid \Pi_{k'} \in G^+\}$, i.e., the first item $\Pi_k \in U^+$ in Π is encountered before every item from G^+. It follows that

$$G^+ \subseteq \big\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\big)\big\}.$$

By Lemma 6, $v(\Pi_k) \geq v(G^+)$, and hence we obtain

$$\begin{aligned}v(\Pi(C)) &= v\big(\Pi(C) \cap \textsc{MGreedy}(\mathcal{I}, C)\big) + v(U^+) \\ &\geq v\big(\Pi(C) \cap \textsc{MGreedy}(\mathcal{I}, C)\big) + v(\Pi_k) \\ &\geq v\big(\Pi(C) \cap \textsc{MGreedy}(\mathcal{I}, C)\big) + v(G^+) = v(\textsc{MGreedy}(\mathcal{I}, C)).\end{aligned}$$ ◀

While it is obvious that Universal runs in polynomial time, we show that it can be modified to run in time $\Theta(n \log n)$. The proof is omitted due to space constraints.

▶ **Theorem 7.** *The algorithm* Universal *can be implemented to run in time* $\Theta(n \log n)$.

We now give a general lower bound on the robustness factor of any policy for the oblivious knapsack problem. This shows that Universal is best possible.

▶ **Theorem 8.** *For every* $\delta > 0$*, there are instances of the oblivious knapsack problem where no policy achieves a robustness factor of* $2 - \delta$.

Proof. We give a family of instances, one for each size $n \geq 3$. We ensure that for every item i of the instance of size n, there is a capacity C, such that packing item i first can only lead to a solution that is worse than $\textsc{Opt}(\mathcal{I}, C)$ by a factor of at least $(2 - 4/n)$. This completes the proof, as the factor approaches 2 for increasing values of n. The instance of size n is

given by $\mathcal{I} = \{1, 2, \dots, n\}$ with $l(i) = F_n + F_i - 1$, and $v(i) = 1 + \frac{i}{n}$, where F_i denotes the i-th Fibonacci number ($F_1 = 1, F_2 = 1, F_3 = 2, \dots$).

We need to show that, no matter which item is tried first (i.e., no matter which item is the root of the policy), there is a capacity for which this choice ruins the solution. Observe that both values and sizes of the items are strictly increasing. Assume that item $i \geq 3$ is packed first. Since the smallest item has size $l(1) = F_n$, for capacity $C_i = 2F_n + F_i - 2 < 2F_n + F_i - 1 = l(1) + l(i)$, no additional item fits the knapsack. However, the unique optimum solution in this case is $\textsc{Opt}(\mathcal{I}, C_i) = \{i - 1, i - 2\}$. These two items fit the knapsack, as $l(i-1) + l(i-2) = 2F_n + F_{i-1} + F_{i-2} - 2 = 2F_n + F_i - 2 = C_i$. By definition,

$$\frac{v(i-1) + v(i-2)}{v(i)} = \frac{2n + 2i - 3}{n + i} = 2 - \frac{3}{n+i} \geq 2 - \frac{3}{n}.$$

Thus, policies that first pack item $i \geq 3$ cannot attain a robustness factor better than $2 - 3/n$.

Now, assume that one of the two smallest items is packed first. For capacity $C_{1,2} = l(n) = 2F_n - 1 < 2F_n = l(1) + l(2)$, no additional item fits the knapsack. The unique optimum solution, however, is to pack item n. It remains to compute the ratios

$$\frac{v(n)}{v(1)} > \frac{v(n)}{v(2)} = \frac{2n}{n+2} = 2 - \frac{4}{n+2} > 2 - \frac{4}{n}.$$

Hence, policies that first pack item 1 or item 2 do not achieve a robustness factor better than $2 - 4/n$. ◀

4 Unit Densities

In this section we restrict ourselves to instances of the oblivious knapsack problem, where all items have unit density, i.e., $v(i) = l(i)$ for all items $i \in \mathcal{I}$. For two items $i, j \in \mathcal{I}$ we say that i is smaller than j and write $i \prec j$ if $v(i) < v(j)$, or $v(i) = v(j)$ and $t(i) < t(j)$, where t is the tiebreaking index introduced in Section 3. We give an algorithm UniversalUD (cf. Algorithm 3) that produces a universal policy tailored to achieve the best possible robustness factor equal to the golden ratio $\varphi \approx 1.618$. The algorithm considers the items from smallest to largest, and inserts each item into the output sequence as far to the right as possible, such that the item is not preceded on its left by other items that are more than a factor φ smaller. Intuitively, the algorithm tries as much as possible to keep the resulting order sorted increasingly by size; only when an item dominates another item by a factor of at least φ the algorithm ensures that it precedes this item in the final sequence. Note that, even though φ is irrational, for rationals a, b the condition $a < \varphi b$ can be tested efficiently by testing the equivalent condition $a/b < 1 + b/a$.

▶ **Theorem 9.** *The algorithm* UniversalUD *constructs a universal policy of robustness factor φ when all items have unit density.*

Proof. Given an instance $\mathcal{I}$ of the oblivious knapsack problem with unit densities and any capacity $C \leq v(\mathcal{I})$, we compare the packing $\Pi(C)$ that results from the solution $\Pi = \textsc{UniversalUD}(\mathcal{I})$ with an optimal packing $\textsc{Opt}(\mathcal{I}, C)$. We define the set M of items in $\Pi(C)$ for which at least one smaller item is not in $\Pi(C)$, i.e., more precisely, let $M = \{i \in \Pi(C) \mid \exists j \in \mathcal{I} \setminus \Pi(C) : j \prec i\}$.

We first consider the case that $M \neq \emptyset$ and set $i = \min_{\prec} M$ to be the smallest item in M with respect to '$\prec$'. Consider the iteration r of UniversalUD in which i is inserted into Π, i.e., $i = L_r$. By definition of M, there is an item $j \prec i$ with $j \notin \Pi(C)$. Let j be

Algorithm 3: UniversalUD($\mathcal{I}$)

Input: set of items $\mathcal{I}$
Output: sequence of items Π
$L \leftarrow \langle$items in $\mathcal{I}$ sorted such that $L_1 \prec \cdots \prec L_n\rangle$
$\Pi^{(0)} \leftarrow \emptyset$
for $r \leftarrow 1, \ldots, n$ **do**
  $j \leftarrow 1$
  while $j \leq |\Pi^{(r-1)}|$ **and** $v(L_r) < \varphi v(\Pi_j^{(r-1)})$ **do**
    $j \leftarrow j + 1$
  $\Pi^{(r)} \leftarrow (\Pi_1^{(r-1)}, \ldots, \Pi_{j-1}^{(r-1)}, L_r, \Pi_j^{(r-1)}, \ldots)$
return $\Pi^{(n)}$

the first such item in Π. Since $j \prec i$, we have $j \in \Pi^{(r)}$. From $i \in \Pi(C)$ and $j \notin \Pi(C)$, it follows that i precedes j in Π (and thus in $\Pi^{(r)}$). Let i' be the item directly preceding j in $\Pi^{(r)}$. If $i' = i$, i was compared with j when it was inserted into $\Pi^{(r)}$, with the result that $v(i) \geq \varphi v(j)$ and thus $v(\Pi(C)) \geq \varphi v(j)$. If $i' \neq i$, by definition of j, we still have $i' \in \Pi(C)$. Also, either $i' \succ j$ and thus $v(i') \geq v(j)$, or j was compared with i' when it was inserted into Π in an earlier iteration of UniversalUD, with the result that $v(i') > \frac{1}{\varphi} v(j)$. Again, $v(\Pi(C)) \geq v(i) + v(i') > v(j) + \frac{1}{\varphi} v(j) = \varphi v(j)$.

In both cases it follows from $j \notin \Pi(C)$ that $v(\textsc{Opt}(\mathcal{I}, C)) \leq C < v(\Pi(C)) + v(j)$, and using $v(j) \leq \frac{1}{\varphi} v(\Pi(C))$ we get

$$\frac{v(\textsc{Opt}(\mathcal{I}, C))}{v(\Pi(C))} < \frac{v(\Pi(C)) + v(j)}{v(\Pi(C))} < 1 + \frac{1}{\varphi} = \varphi.$$

Now, assume that $M = \emptyset$. Intuitively, this means that $\Pi(C)$ consists of a prefix of L (the smallest items). Let $i_1 \succ \cdots \succ i_k$ be the items in $\Pi(C) \setminus \textsc{Opt}(\mathcal{I}, C)$, and let $j_1 \succ \cdots \succ j_l$ be the items in $\textsc{Opt}(\mathcal{I}, C) \setminus \Pi(C)$. As $\Pi(C)$ consists of a prefix of L, we have $|\Pi(C)| \geq |\textsc{Opt}(\mathcal{I}, C)|$ and thus $k \geq l$. If $k = 0$, the claim trivially holds. Otherwise, since M is empty, we have $j_l \succ i_1$. Is suffices to show $v(j_h) \leq \varphi v(i_h)$ for all $h \leq l$. To this end, we consider any fixed $h \leq l$. From $v(\{i_1, \ldots, i_{h-1}\}) \leq v(\{j_1, \ldots, j_{h-1}\})$ it follows that

$$v(j_h) \leq v(\textsc{Opt}(\mathcal{I}, C)) - v(\{j_1, \ldots, j_{h-1}\}) \leq C - v(\{i_1, \ldots, i_{h-1}\}).$$

This implies that j_h cannot precede all items of $\{i_h, \ldots, i_k\}$ in Π, as $j_h \notin \Pi(C)$. Hence, there is an item $i'' \in \{i_h, \ldots, i_k\}$ that precedes j_h in Π. Since $j_h \succ i''$, in the iteration when UniversalUD inserted j_h into Π, i'' was already present. From the fact that i'' ended up preceding j_h it follows that j_k was compared with i'' and thus $v(j_h) < \varphi v(i'') \leq \varphi v(i_h)$. We obtain

$$\frac{v(\textsc{Opt}(\mathcal{I}, C))}{v(\Pi(C))} \leq \frac{v(\textsc{Opt}(\mathcal{I}, C) \setminus \Pi(C))}{v(\Pi(C) \setminus \textsc{Opt}(\mathcal{I}, C))} = \frac{\sum_{h=1}^{l} v(j_h)}{\sum_{h=1}^{k} v(i_h)} \leq \frac{\sum_{h=1}^{l} \varphi v(i_h)}{\sum_{h=1}^{l} v(i_h)} = \varphi. \qquad \blacktriangleleft$$

A naïve implementation of UniversalUD runs in time $\Theta(n^2)$. We improve this running time to $\Theta(n \log n)$. The proof of the following theorem is omitted due to space constraints.

▶ **Theorem 10.** *The algorithm* UniversalUD *can be implemented to run in time* $\Theta(n \log n)$.

We now establish that UniversalUD is best possible, even if we permit non-universal policies (the proof is omitted due to space constraints).

▶ **Theorem 11.** *There are instances of the oblivious knapsack problem where no policy achieves a robustness factor of $\varphi - \delta$, for any $\delta > 0$, even when all items have unit density.*

5 Hardness

Although we can always find a 2-robust universal policy in polynomial time, we show in this section that, for any fixed $\alpha \in (1, \infty)$, it is intractable to decide whether a given universal policy is α-robust. This hardness result also holds for instances with unit densities when α is part of the input. As the final – and arguably the most interesting – result of this section, we establish coNP-hardness of the the problem to decide for a given instance and given $\alpha > 1$, whether the instance admits a universal policy with robustness factor α. All proofs rely on the hardness of the following version of SubsetSum.

▶ **Lemma 12.** *Let $W = \{w_1, w_2, \dots, w_n\}$ be a set of positive integer weights and $T \leq \sum_{k=1}^{n} w_k$ be a target sum. The problem of deciding whether there is a subset $U \subseteq W$ with $\sum_{w \in U} w = T$ is NP-complete, even when*
1. *$T = 2^k$ for some integer $k \geq 3$,*
2. *all weights are in the interval $[2, T/2)$,*
3. *all weights have a difference of at least 2 to the closest power of 2.*

We first show that it is intractable to determine the robustness factor of a given universal policy.

▶ **Theorem 13.** *For any fixed and polynomially representable $\alpha > 1$ it is coNP-complete to decide whether a given universal policy for the oblivious knapsack problem is α-robust.*

Sketch of proof. For the proof of the coNP-hardness, we reduce from the variant of SubsetSum specified in Lemma 12. An instance of this problem is given by a set $W = \{w_1, w_2, \dots, w_n\}$ of positive integer weights in the range $[2, T/2)$ and a target sum $T = 2^k$ for some integer $k \geq 3$. Let $\alpha > 1$ be polynomially representable. We construct an instance $\mathcal{I}$ and a sequence Π such that Π is an α-robust universal policy for $\mathcal{I}$ if and only if the instance of SubsetSum has no solution. First, we introduce for each weight $w \in W$ a *regular item* with $l(i_w) = v(i_w) = w$. This ensures that the optimal knapsack solution for capacity T is at least T if the instance of SubsetSum has a solution. We further introduce a set of additional items that ensure that the robustness factor for all capacities except T is at least α, regardless whether the instance of SubsetSum has a solution, or not. Specifically, let $m = \log_2 T - 1$. Then, for each $k \in \{0, 1, \dots, m\}$ we introduce an *auxiliary item* j_k with size $l(j_k) = 2^k$ and value $v(j_k) = 2^k(1 - \epsilon)$, where $\epsilon > 0$ is suitably chosen depending on α. Intuitively, the set of auxiliary items ensures that all capacities less than T can be packed well enough. Finally, we introduce a *dummy item* with size $T + 1$ and large value. Intuitively, the dummy item ensures that all capacities larger than T can be packed well enough. The hardness is then established by showing that the sequence that contains first the dummy item, then the auxiliary items in decreasing order, and then the regular items is an α-robust universal solution if and only if the instance of SubsetSum has no solution. ◀

We give a result similar to Theorem 13 for unit densities. Note that this time we require α to be part of the input. The proof is technically more involved, and we only give a sketch due to space constraints.

▶ **Theorem 14.** *It is coNP-complete to decide whether, for given $\alpha > 1$, a given universal policy for the oblivious knapsack problem is α-robust, even when all items have unit density.*

Sketch of proof. We use a similar construction as in the proof of Theorem 13, with the additional complication that we are only permitted to use items of unit density. To meet this requirement, we modify the auxiliary items to have sizes and values equal to $(1-\varepsilon)2^k$, in such a way that all capacities up to $T-1$ are packed well (but not optimally) by our policy. We replace the dummy item with a series of items of exponentially growing sizes and values. These dummy items ensure that all capacities larger than T can be packed well. The crucial capacities are those close to T. For these capacities, we use that if the SubsetSum instance has no solution, a value of at most $T-\varepsilon$ can be packed, while if the instance has a solution, the optimum value is T. On the other hand, the policy we construct packs $(1-\varepsilon)(T-1)$. By setting $\alpha = \frac{T-\epsilon}{(1-\varepsilon)(T-1)}$, we ensure that our policy can only be α-robust if the SubsetSum instance has a solution. It remains to tune ε such that our policy is α-robust for all capacities (including fractional ones) except those close to T. Note that the proof relies on the fact that α is part of the input and may thus be set arbitrarily close to 1. ◀

Finally, we prove that it is hard to decide whether a given instance admits an α-robust universal policy when α is part of the input. The full proof is omitted due to space constraints.

▶ **Theorem 15.** *It is* coNP*-hard to decide whether, for given $\alpha > 1$, an instance of the oblivious knapsack problem admits an α-robust universal policy, even when all items have unit density.*

Sketch of proof. We consider the same set of items $\mathcal{I}$ as in the proof of Theorem 14. We show that, if an α-robust, universal policy exists, then it must be similar to the policy Π that we construct in the proof of Theorem 14. From this, we are able to infer that $\mathcal{I}$ admits an α-robust, universal policy if and only if Π is α-robust. As shown in the proof of Theorem 14, it follows that $\mathcal{I}$ admits an α-robust universal policy if and only if the underlying instance of SubsetSum has no solution. ◀

References

1 M. A. Bender, R. Cole, and E. D. Demaine. Scanning and traversing: maintaining data for traversals in a memory hierarchy. In *Proceedings of the 10th European Symposium on Algorithms (ESA)*, pages 139–151, 2002.

2 Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98:49–71, 2003.

3 A. Bhalgat, A. Goel, and S. Khanna. Improved approximation results for stochastic knapsack problems. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1647–1665, 2011.

4 P. C. Bouman, J. M. van den Akker, and J. A. Hoogeveen. Recoverable robustness by column generation. In *Proceedings of the 19th European Symposium on Algorithms (ESA)*, pages 215–226, 2011.

5 Christina Büsing, Arie M.C.A. Koster, and Manuel Kutschka. Recoverable robust knapsacks: the discrete scenario case. *Optimization Letters*, 5(3):379–392, 2011.

6 J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

7 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. In *Proceddings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 208–217, 2004.

8 Vladimir G. Deineko, Rüdiger Rudolf, and Gerhard J. Woeginger. Sometimes travelling is easy: The master tour problem. In *Proceedings of the 3rd European Symposium on Algorithms (ESA)*, pages 128–141. Springer, 1995.

9 Leah Epstein, Asaf Levin, Alberto Marchetti-Spaccamela, Nicole Megow, Julian Mestre, Martin Skutella, and Leen Stougie. Universal sequencing on an unreliable machine. *SIAM Journal on Computing*, 41(3):565–586, 2012.

10 M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS)*, pages 285–297, 1999.

11 Kai-Simon Goetzmann, Sebastian Stiller, and Claudio Telha. Optimization over integers with robustness in cost and few constraints. In *Proceedings of the 9th Workshop on Approximation and Online Algorithms (WAOA)*, pages 89–101, 2011.

12 Refael Hassin and Shlomi Rubinstein. Robust matchings. *SIAM Journal on Discrete Mathematics*, 15(4):530–537, 2002.

13 Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for TSP, Steiner tree, and set cover. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 386–395, 2005.

14 Naonori Kakimura, Kazuhisa Makino, and Kento Seimi. Computing knapsack solutions with cardinality robustness. In *Proceedings of the 22nd International Conference on Algorithms and Computation (ISAAC)*, pages 693–702, 2011.

15 Bernhard Korte and Jens Vygen. *Combinatorial Optimization. Theory and Algorithms.* Springer, 2nd edition, 2002.

16 Nicole Megow and Julián Mestre. Instance-sensitive robustness guarantees for sequencing with unknown packing and covering constraints. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 495–504, 2013.

17 Ralph Merkle and Martin E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978.

18 Michele Monaci and Ulrich Pferschy. On the robust knapsack problem. In *Proceedings of the 10th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW)*, pages 207–210, 2011.

19 Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

20 Harald Räcke. Survey on oblivious routing strategies. In *Proceedings of the 5th Conference on Computability in Europe: Mathematical Theory and Computational Practice (CiE)*, pages 419–429, 2009.

21 Adi Shamir. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. In *Proceedings of the 23rd Symposium on Foundations of Computer Science (FOCS)*, pages 145–152, 1982.

22 L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC)*, pages 263–277, 1981.

23 Gang Yu. On the max-min 0-1 knapsack problem with robust optimization applications. *Operations Research*, 44(2):407–415, 1996.

Exploring Subexponential Parameterized Complexity of Completion Problems*

Pål Grønås Drange, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger

University of Bergen, Norway
{Pal.Drange|Fedor.Fomin|Michal.Pilipczuk|Yngve.Villanger}@ii.uib.no

Abstract

Let $\mathcal{F}$ be a family of graphs. In the $\mathcal{F}$-Completion problem, we are given an n-vertex graph G and an integer k as input, and asked whether at most k edges can be added to G so that the resulting graph does not contain a graph from $\mathcal{F}$ as an induced subgraph. It appeared recently that special cases of $\mathcal{F}$-Completion, the problem of completing into a chordal graph known as Minimum Fill-in, corresponding to the case of $\mathcal{F} = \{C_4, C_5, C_6, \ldots\}$, and the problem of completing into a split graph, i.e., the case of $\mathcal{F} = \{C_4, 2K_2, C_5\}$, are solvable in parameterized subexponential time $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$. The exploration of this phenomenon is the main motivation for our research on $\mathcal{F}$-Completion.

In this paper we prove that completions into several well studied classes of graphs without long induced cycles also admit parameterized subexponential time algorithms by showing that:

- The problem Trivially Perfect Completion is solvable in parameterized subexponential time $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$, that is $\mathcal{F}$-Completion for $\mathcal{F} = \{C_4, P_4\}$, a cycle and a path on four vertices.
- The problems known in the literature as Pseudosplit Completion, the case where $\mathcal{F} = \{2K_2, C_4\}$, and Threshold Completion, where $\mathcal{F} = \{2K_2, P_4, C_4\}$, are also solvable in time $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$.

We complement our algorithms for $\mathcal{F}$-Completion with the following lower bounds:

- For $\mathcal{F} = \{2K_2\}$, $\mathcal{F} = \{C_4\}$, $\mathcal{F} = \{P_4\}$, and $\mathcal{F} = \{2K_2, P_4\}$, $\mathcal{F}$-Completion cannot be solved in time $2^{o(k)}n^{\mathcal{O}(1)}$ unless the Exponential Time Hypothesis (ETH) fails.

Our upper and lower bounds provide a complete picture of the subexponential parameterized complexity of $\mathcal{F}$-Completion problems for $\mathcal{F} \subseteq \{2K_2, C_4, P_4\}$.

1998 ACM Subject Classification G.2.2 Graph algorithms

Keywords and phrases edge completion, modification, subexponential parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.288

1 Introduction

Let $\mathcal{F}$ be a family of graphs. In this paper we study the following $\mathcal{F}$-Completion problem.

$\mathcal{F}$-Completion	**Parameter:** k
Input: A graph $G = (V, E)$ and a non-negative integer k.	
Question: Does there exist a supergraph $H = (V, E \cup S)$ of G, such that $\lvert S\rvert \leq k$ and H contains no graph from $\mathcal{F}$ as an induced subgraph?	

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 267959.



© Pål Grønås Drange, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 288–299



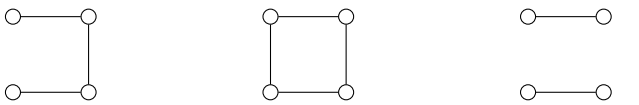
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



The $\mathcal{F}$-Completion problems form a subclass of graph modification problems where one is asked to apply a bounded number of changes to an input graph to obtain a graph with some property. Graph modification problems arise naturally in many branches of science and have been studied extensively during the past 40 years. Interestingly enough, despite the long study of the problem, there is no known dichotomy classification of $\mathcal{F}$-Completion explaining for which classes $\mathcal{F}$ the problem is solvable in polynomial time and for which the problem is NP-complete.

One of the motivations to study completion problems in graph algorithms comes from their intimate connections to different width parameters. For example, the treewidth of a graph, one of the most fundamental graph parameters, is the minimum over all possible completions into a chordal graph of the maximum clique size minus one [2]. The treedepth of a graph, also known as the vertex ranking number, the ordered chromatic number, and the minimum elimination tree height, plays a crucial role in the theory of sparse graphs developed by Nešetřil and Ossona de Mendez [20]. Mirroring the connection between treewidth and chordal graphs, the treedepth of a graph can be defined as the largest clique size in a completion to a *trivially perfect graph*. Similarly, the vertex cover number of a graph is equal to the minimum of the largest clique size taken over all completions to a *threshold graph*, minus one.

Parameterized algorithms for completion problems. For a long time in parameterized complexity the main focus of studies in $\mathcal{F}$-Completion was for the case when $\mathcal{F}$ was an infinite family of graphs, e.g., Minimum Fill-in or Interval Completion [15, 19, 21]. This was mainly due to the fact that when $\mathcal{F}$ is a finite family, $\mathcal{F}$-Completion is solvable on an n-vertex graph in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some function f by a simple branching argument; this was first observed by Cai [4]. More precisely, if the maximum number of non-edges in a graph from $\mathcal{F}$ is d, then the corresponding $\mathcal{F}$-Completion is solvable in time $d^k \cdot n^{\mathcal{O}(1)}$. The interest in $\mathcal{F}$-Completion problems started to increase with the advance of kernelization. It appeared that from the perspective of kernelization, even for the case of finite families $\mathcal{F}$ the problem is far from trivial. Guo [12] initiated the study of kernelization algorithms for $\mathcal{F}$-Completion in the case when the forbidden set $\mathcal{F}$ contains the graph C_4, see Figure 1. (In fact, Guo considered edge deletion problems, but they are polynomial time equivalent to completion problems to the complements of the forbidden induced subgraphs.) In the literature, the most studied graph classes containing no induced C_4 are the *split* graphs, i.e., $\{2K_2, C_4, C_5\}$-free graphs, *threshold* graphs, i.e., $\{2K_2, P_4, C_4\}$-free graphs, and $\{C_4, P_4\}$-free graphs, that is, *trivially perfect* graphs [3]. Guo obtained polynomial kernels for the completion problems for chain graphs, split graphs, threshold graphs and trivially perfect graphs and concluded that, as a consequence of his polynomial kernelization, the corresponding $\mathcal{F}$-Completion problems: Chain Completion, Split Completion, Threshold Completion and Trivially Perfect Completion are solvable in times $\mathcal{O}(2^k + mnk)$, $\mathcal{O}(5^k + m^4 n)$, $\mathcal{O}(4^k + kn^4)$, and $\mathcal{O}(4^k + kn^4)$, respectively.



(a) P_4 **(b)** C_4 **(c)** $2K_2 = \overline{C_4}$

Figure 1 Forbidden induced subgraphs. *Trivially perfect graphs* are $\{C_4, P_4\}$-free, threshold graphs are $\{2K_2, P_4, C_4\}$-free, and cographs are P_4-free.

Obstruction set $\mathcal{F}$	Graph class name	Complexity
$C_4, C_5, C_6, \ldots$	Chordal	SUBEPT [9]
C_4, P_4	Trivially Perfect	SUBEPT (Theorem 1)
$2K_2, C_4, C_5$	Split	SUBEPT [10]
$2K_2, C_4, P_4$	Threshold	SUBEPT (Theorem 10)
$2K_2, C_4$	Pseudosplit	SUBEPT (Theorem 11)
$\overline{P_3}, K_t$, $t = o(k)$	Co-t-cluster	SUBEPT [8]
$\overline{P_3}$	Co-cluster	E [16]
$2K_2$	$2K_2$-free	E (Theorem 12)
C_4	C_4-free	E (Theorem 12)
P_4	Cograph	E (Theorem 12)
$2K_2, P_4$	Co-Trivially Perfect	E (Theorem 12)

Figure 2 Known subexponential complexity of $\mathcal{F}$-Completion for different sets $\mathcal{F}$. SUBEPT means the problem is solvable in subexponential time $2^{o(k)}n^{\mathcal{O}(1)}$ and E means that the problem is not solvable in subexponential time unless ETH fails.

The work on kernelization of $\mathcal{F}$-Completion problems was continued by Kratsch and Wahlström [17] who showed that there exists a set $\mathcal{F}$ consisting of one graph on seven vertices for which $\mathcal{F}$-Completion does not admit a polynomial kernel. Guillemot et al. [11] showed that Cograph Completion, i.e., the case $\mathcal{F} = \{P_4\}$, admits a polynomial kernel, while for $\mathcal{F} = \{\overline{P_{13}}\}$, the complement of a path on 13 vertices, $\mathcal{F}$-Completion has no polynomial kernel. These results were significantly improved by Cai and Cai [5]: For $\mathcal{F} = \{P_\ell\}$ or $\mathcal{F} = \{C_\ell\}$, the problems $\mathcal{F}$-Completion and $\mathcal{F}$-Edge Deletion admit a polynomial kernel if and only if the forbidden graph has at most three edges.

It appeared recently that for some choices of $\mathcal{F}$, $\mathcal{F}$-Completion is solvable in *subexponential* time. The exploration of this phenomenon is the main motivation for our research on this problem. The last chapter of Flum and Grohe's textbook on parameterized complexity theory [7, Chapter 16] concerns subexponential fixed parameter tractability, the complexity class SUBEPT, which, loosely speaking—we skip here some technical conditions—is the class of problems solvable in time $2^{o(k)}n^{\mathcal{O}(1)}$, where n is the input length and k is the parameter. Until recently, the only notable examples of problems in SUBEPT were problems on planar graphs, and more generally, on graphs excluding some fixed graph as a minor [6]. In 2009, Alon et al. [1] used a novel application of color coding, dubbed *chromatic coding*, to show that parameterized Feedback Arc Set in Tournaments is in SUBEPT. As Flum and Grohe [7] observed, for most of the natural parameterized problems, already the classical NP-hardness reductions can be used to refute the existence of subexponential parameterized algorithms, unless the following well-known complexity hypothesis formulated by Impagliazzo, Paturi, and Zane [14] fails.

▶ **Exponential Time Hypothesis** (ETH). There exists a positive real number s such that 3-CNF-SAT with n variables cannot be solved in time 2^{sn}.

Thus, it is most likely that the majority of parameterized problems are not solvable in subexponential parameterized time and until very recently no natural parameterized problem solvable in subexponential parameterized time on general graphs was known. A subset of the authors recently showed that Minimum Fill-in, also known as Chordal Completion, which is equivalent to $\mathcal{F}$-Completion with $\mathcal{F}$ consisting of cycles of length at least four, is in SUBEPT [9], simultaneously establishing that Chain Completion is solvable in subexponential time. Later, Ghosh et al. [10] showed that Split Completion is solvable

in subexponential time. On the other hand, Komusiewicz and Uhlmann [16], showed that an edge modification problem known as Cluster Deletion, does not belong to SUBEPT unless ETH fails. Let us note that Cluster Deletion is equivalent to $\mathcal{F}$-Completion when $\mathcal{F} = \{\overline{P_3}\}$, the complement of the path P_3. On the other hand, it is interesting to note that by the result of Fomin et al. [8], Cluster Deletion into t Clusters, i.e., the complement problem for $\mathcal{F}$-Completion for $\mathcal{F} = \{\overline{P_3}, K_t\}$, is in SUBEPT for $t = o(k)$.

Our results. In this work we extend the class of $\mathcal{F}$-Completion problems admitting subexponential time algorithms, see Figure 2. Our main algorithmic result is the following:

> Trivially Perfect Completion is solvable in time $2^{\mathcal{O}(\sqrt{k}\log k)}n^{\mathcal{O}(1)}$ and is thus in SUBEPT. This problem is the $\mathcal{F}$-Completion problem for $\mathcal{F} = \{C_4, P_4\}$.

On a very high level, our algorithm is based on the same strategy as the algorithm for completion into chordal graphs [9]. Just like in that algorithm, we enumerate in parameterized subexponential time special structures called *potential maximal cliques* which are the maximal cliques in some minimal completion into a trivially perfect graph that uses at most k edges. As far as we succeed in enumerating these objects, we do dynamic programming to find an optimal completion. But here the similarities end. To enumerate potential maximal cliques for trivially perfect graphs, we have to use completely different structural properties from those used for the case of chordal graphs.

We also show that within the same running time the $\mathcal{F}$-Completion problem is solvable for $\mathcal{F} = \{2K_2, C_4\}$, and $\mathcal{F} = \{2K_2, P_4, C_4\}$. This corresponds to completion into threshold and pseudosplit graphs, respectively. Let us note that combined with the results of Fomin and Villanger [9] and Ghosh et al. [10], this implies that all four problems considered by Guo in [12] are in SUBEPT, in addition to admitting a polynomial kernel. We finally complement our algorithmic findings by showing the following:

> For $\mathcal{F} = \{2K_2\}$, $\mathcal{F} = \{C_4\}$, $\mathcal{F} = \{P_4\}$ and $\mathcal{F} = \{2K_2, P_4\}$, the $\mathcal{F}$-Completion problem cannot be solved in time $2^{o(k)}n^{\mathcal{O}(1)}$ unless ETH fails.

Thus, we obtain a complete classification for all $\mathcal{F} \subseteq \{2K_2, P_4, C_4\}$.

Organization of the paper. In Section 2 we give structural results about trivially perfect graphs and their completions, and give the main result of the paper: an algorithm solving Trivially Perfect Completion in subexponential time. In Section 3 we briefly discuss the tools needed to obtain subexponential time algorithms for Threshold Completion and Pseudosplit Completion. Due to space constraints, full expositions of these algorithms have been deferred to the full version. In Section 4, we mention the lower bounds on $\mathcal{F}$-Completion when $\mathcal{F}$ is $\{2K_2\}$, $\{C_4\}$, $\{P_4\}$, or $\{2K_2, P_4\}$. Full proofs for the lower bounds have also been deferred to the full version, where, in addition, proofs for results marked with ♠ can be found. Finally, in Section 5 we give some concluding remarks and state some interesting remaining questions.

Notation. We consider only finite simple undirected graphs. We use n to denote the number of vertices and m the number of edges in a graph G. If $G = (V, E)$ is a graph, and $A, B \subseteq V$, we write $E(A, B)$ for the edges with one endpoint in A and the other in B, and we write $E(A) = E(A, A)$ for the edges inside A and m_A for $|E(A)|$.

We write $N(U)$ for $U \subseteq V(G)$ to denote the open neighborhood $\bigcup_{v \in U}(N(v)) \setminus U$, and $N[U] = N(U) \cup U$ to denote the closed neighborhood. For a graph G and a set of edges S, we write $G + S = (V, E \cup S)$ and $G - S = (V, E \setminus S)$, and if $U \subseteq V$ is a set of vertices, then $G - U = G[V \setminus U]$. A *universal vertex* in a graph is a vertex v such that $N[v] = V(G)$. Let

uni(G) denote the set of universal vertices of G. Observe that uni(G), when non-empty, is always a clique, and we will refer to it as the *(maximal) universal clique*.

2 Completion to trivially perfect graphs

In this section we study the Trivially Perfect Completion problem which is the special case of $\mathcal{F}$-Completion for $\mathcal{F} = \{C_4, P_4\}$. The decision version of the problem was shown to be NP-complete by Yannakakis [22]. Since trivially perfect graphs are characterized by a finite set of forbidden induced subgraphs, it follows from Cai [4] that the problem also is fixed parameter tractable, i.e., it belongs to the class FPT.

The main result of this section is the following theorem.

▶ **Theorem 1.** *For an input* (G, k), Trivially Perfect Completion *is solvable in time* $2^{\mathcal{O}(\sqrt{k}\log k)} + \mathcal{O}(kn^4)$.

Throughout this section, an edge set S is called a *completion* for G if $G + S$ is trivially perfect. Furthermore, a set S is called a *minimal completion* for G if no proper subset of S is a completion for G. The main outline of the algorithm is as follows:

Step A: On input (G, k), we first apply the algorithm by Guo [12] to obtain a kernel of size $\mathcal{O}(k^3)$. The running time of this algorithm is $\mathcal{O}(kn^4)$.

Step B: Assuming our input instance is of size $\mathcal{O}(k^3)$, we show how to generate all special vertex subsets of the kernel which we call *vital potential maximal cliques* in time $2^{\mathcal{O}(\sqrt{k}\log k)}$. A vital potential maximal clique $\Omega \subseteq V(G)$ is a vertex subset which is a maximal clique in some minimal completion of size at most k.

Step C: Using dynamic programming, we show how to compute an optimal solution or to conclude that (G, k) is a ***no*** instance, in time polynomial in the number of vital potential maximal cliques.

2.1 Structure of trivially perfect graphs

Apart from the aforementioned characterization by forbidden induced subgraphs, several other equivalent definitions of trivially perfect graphs are known. These definitions reveal more structural properties of this graph class which will be essential in our algorithm. Therefore, before proceeding with the proof of Theorem 1, we establish a number of results on the structure of trivially perfect graphs and minimal completions which will be useful.

The trivially perfect graphs have a decomposition tree which we call a *universal clique decomposition*, in which each node in the tree corresponds to a maximal set of vertices that all are universal for the graph induced by the vertices in the subtree.

Let T be a rooted tree and t be a node of T. We denote by T_t the maximal subtree of T rooted in t. We can now use the universal clique uni(G) of a trivially perfect graph $G = (V, E)$ to make a decomposition structure.

▶ **Definition 2** (Universal clique decomposition)**.** A *universal clique decomposition* of a connected trivially perfect graph $G = (V, E)$ is a pair $(T = (V_T, E_T), \mathcal{B} = \{B_t\}_{t \in V_T})$, where T is a rooted tree and $\mathcal{B}$ is a partition of the vertex set V into disjoint non-empty subsets, such that

- if $vw \in E(G)$ and $v \in B_t$ and $w \in B_s$, then s and t are on a path from a leaf to the root, with possibly $s = t$, and
- for every node $t \in V_T$, the set of vertices B_t is the maximal universal clique in the subgraph $G[\bigcup_{s \in V(T_t)} B_s]$.

We call the vertices of T *nodes* and the sets in $\mathcal{B}$ *bags* of the universal clique decomposition $(T, \mathcal{B})$. By slightly abusing the notation, we often do not distinguish between nodes and bags. Note that by the definition, in a universal clique decomposition every non-leaf node has at least two children, since otherwise the universal clique contained in the corresponding bag would not be maximal.

▶ **Lemma 3 (♠).** *A connected graph G admits a universal clique decomposition if and only if it is trivially perfect. Moreover, such a decomposition is unique up to isomorphisms.*

For the purposes of the dynamic programming procedure, we define the following notion.

▶ **Definition 4** (Block)**.** Let $(T = (V_T, E_T), \mathcal{B} = \{B_t\}_{t \in V_T})$ be the universal clique decomposition of a connected trivially perfect graph $G = (V, E)$. For each node $t \in V_T$, we associate a *block* $L_t = (B_t, D_t)$, where

- B_t is the subset of V contained in the bag corresponding to t, and
- D_t is the set of vertices of V contained in the bags corresponding to the nodes of the subtree T_t.

The *tail* of a block L_t is the set of vertices Q_t contained in the bags corresponding to the nodes of the path from t to r in T, where r is the root of T.

When t is a leaf of T, we have that $B_t = D_t$ and we call the block $L_t = (B_t, D_t)$ a *leaf block*. If t is the root, we have that $D_t = V(G)$ and we call L_t the *root block*. Otherwise, we call L_t an *internal block*.

Observe that for every block $L_t = (B_t, D_t)$ with tail Q_t we have that $B_t \subseteq Q_t$, $B_t \subseteq D_t$, and $D_t \cap Q_t = B_t$. Note also that Q_t is a clique and the vertices of Q_t are universal to $D_t \setminus B_t$. The following lemma summarizes the properties of universal clique decompositions, maximal cliques, and blocks used in our proof.

▶ **Lemma 5 (♠).** *Let $(T, \mathcal{B})$ be the universal clique decomposition of a connected trivially perfect graph G and let $L = (B, D)$ be a block with Q as its tail.*

(i) *If L is a leaf block, then $Q = N_G[v]$ for every $v \in B$.*

(ii) *The following are equivalent:*

1. *L is a leaf block,*
2. *$D = B$, and*
3. *Q is a maximal clique of G.*

(iii) *If L is a non-leaf block, then for every two vertices u, v from different connected components of $G[D \setminus B]$, we have that $Q = N_G(u) \cap N_G(v)$.*

2.2 Structure of minimal completions

Before we proceed with the algorithm, we provide some properties of minimal completions. The following lemma gives insight to the structure of a ***yes*** instance.

▶ **Lemma 6 (♠).** *Let $G = (V, E)$ be a connected graph, S a minimal completion and $H = G + S$. Suppose $L = (B, D)$ is a block in some universal clique decomposition of H and denote by $D_1, D_2, \ldots, D_\ell$ the connected components of $H[D] - B$.*

(i) *If L is not a leaf block, then $\ell > 1$;*

(ii) *if $\ell > 1$, then in G every vertex $v \in B$ has at least one neighbor in each set $D_1, D_2, \ldots, D_\ell$;*

(iii) *the graph $G[D_i]$ is connected for every $i \in \{1, \ldots, \ell\}$; and*

(iv) *for every $i \in \{1, \ldots, \ell\}$, $B \subseteq N_G(D \setminus (B \cup D_i))$.*

2.3 The algorithm

As has been already mentioned, the following concept is crucial for our algorithm. Recall that when Ω is a set of vertices in a graph G, by m_Ω we mean the number of edges in $G[\Omega]$.

▶ **Definition 7** (Vital potential maximal clique)**.** Let (G,k) be an instance of Trivially Perfect Completion. A vertex set $\Omega \subseteq V(G)$ is a *potential maximal clique* if Ω is a maximal clique in some minimal trivially perfect completion of G. If moreover this trivially perfect completion contains at most k edges, then the potential maximal clique is called *vital*.

Observe that given a ***yes*** instance (G,k) and a minimal completion S of size at most k, every maximal clique in $G+S$ is a vital potential maximal clique in G. Note also that in particular, any vital potential maximal clique contains at most k non-edges.

The following definition will be useful:

▶ **Definition 8** (Fill number)**.** Let $G=(V,E)$ be a graph, S a completion and $H=G+S$. We define the *fill* of a vertex v, denoted by $\mathrm{fn}_H^G(v)$ as the number of edges incident to v in S.

Let us observe that there are at most $2\sqrt{k}$ vertices v such that $\mathrm{fn}_H^G(v) > \sqrt{k}$. It follows that for every set $U \subseteq V$ such that $|U| > 2\sqrt{k}$, there is a vertex $u \in U$ with $\mathrm{fn}_H^G(u) \leq \sqrt{k}$. Any vertex u such that $\mathrm{fn}_H^G(u) \leq \sqrt{k}$ will be referred to as a *cheap* vertex.

Everything is settled to start the proof of Theorem 1. Our algorithm proceeds in three steps. We first compress the instance to an instance of size $\mathcal{O}(k^3)$, then we enumerate all (subexponentially many) vital potential cliques in this new instance, and finally we do a dynamic programming procedure on these objects.

Step A. Kernelization. For a given input (G,k), we start by applying the kernelization algorithm by Guo [12] to construct in time $\mathcal{O}(kn^4)$ an equivalent instance (G',k'), where G' has $\mathcal{O}(k^3)$ vertices and $k' \leq k$. Therefore, from now on we can simply assume that the input graph G has $\mathcal{O}(k^3)$ vertices. Without loss of generality, we can also assume that G is connected, since we may treat each connected component of G separately.

Step B. Enumeration. In this step, we give an algorithm that in time $2^{\mathcal{O}(\sqrt{k}\log k)}$ outputs a family $\mathcal{C}$ of vertex subsets of G such that

- the size of $\mathcal{C}$ is $2^{\mathcal{O}(\sqrt{k}\log k)}$, and
- every vital potential maximal clique belongs to $\mathcal{C}$.

We identify five different types of vital potential maximal cliques. For each type i, $1 \leq i \leq 5$, we list a family $\mathcal{C}_i$ of $2^{\mathcal{O}(\sqrt{k}\log k)}$ subsets containing all vital potential maximal cliques of this type. Finally, $\mathcal{C} = \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_5$. We show that every vital potential maximal clique of (G,k) is of at least one type and that all objects of each type can be enumerated in $2^{\mathcal{O}(\sqrt{k}\log k)}$ time.

Let Ω be a vital potential maximal clique. By the definition of Ω, there exists a minimal completion with at most k edges into a trivially perfect graph H such that Ω is a maximal clique in H. Let $(T=(V_T,E_T),\mathcal{B}=\{B_t\}_{t\in V_T})$ be the universal clique decomposition of H. Recall that by Lemma 5, Ω corresponds to a path $P_{rt} = B_{t_0}B_{t_1}\cdots B_{t_q}$ in T from the root $r=t_0$ to a leaf $t=t_q$. Then for the corresponding leaf block (B_t,D_t) with tail Q_t, we have that $\Omega = Q_t$. To simplify the notation, we use B_i for B_{t_i}.

Note that the algorithm does not know neither the clique Ω nor the completed trivially perfect graph H. However, in the analysis we may partition all the vital potential maximal cliques Ω with respect to structural properties of Ω and H, and then provide simple enumeration rules that ensure that all vital potential maximal cliques of each type are indeed enumerated. We proceed to description of the types and enumeration rules.

Type 1. Potential maximal cliques of the first type are such that $|V \setminus \Omega| \leq 2\sqrt{k}+2$. The family $\mathcal{C}_1$ consists of all sets $W \subseteq V$ such that $|V \setminus W| \leq 2\sqrt{k}+2$. There are $\binom{\mathcal{O}(k^3)}{2\sqrt{k}+2}$ such sets and we can find all of them in time $2^{\mathcal{O}(\sqrt{k}\log k)}$ by the brute-force algorithm trying all vertex subsets of size at least $|V| - 2\sqrt{k} + 2$. Thus every Type 1 vital potential maximal clique is in $\mathcal{C}_1$.

Type 2. By Lemma 5 *(1)*, we have that $\Omega = Q_t = N_H[v]$ for each vertex $v \in D_t = B_t$. Vital potential maximal cliques of the second type are such that $|B_t| > 2\sqrt{k}$. Observe that then at least one vertex $v \in B_t$ should be *cheap*, i.e., $\mathrm{fn}_H^G(v) \leq \sqrt{k}$. We generate the family $\mathcal{C}_2$ as follows. Every set in $\mathcal{C}_2$ is of the form $W_1 \cup W_2$, where $W_1 = N_G[v]$ for some $v \in V$, and $|W_2| \leq \sqrt{k}$. There are at most $\mathcal{O}(\binom{\mathcal{O}(k^3)}{\sqrt{k}}k^3)$ such sets and they can be enumerated by computing for every vertex v the set $W_1 = N_G[v]$ and adding to each such set all possible subsets of size at most $\sqrt{k}$. Hence every Type 2 vital potential maximal clique is in $\mathcal{C}_2$.

Thus if Ω is not of Types 1 or 2, then $|V \setminus \Omega| > 2\sqrt{k}+2$ and for the corresponding leaf block we have $|B_t| \leq 2\sqrt{k}$. Since $|V \setminus \Omega| > 2\sqrt{k}+2$ it follows that if (G,k) is a ***yes*** instance, then $V \setminus \Omega$ contains at least two cheap vertices, i.e., vertices with fill number at most $\sqrt{k}$.

We partition the nodes of T that are not on the path $B_0, B_1, \ldots, B_q$ into q disjoint sets $Z_0, Z_1, \ldots, Z_{q-1}$ according to the nodes of the path P_{rt}. Node $x \notin V(P_{rt})$ belongs to Z_i, $i \in \{0, \ldots, q-1\}$, if i is the largest integer such that t_i is an ancestor of x in T. In other words, Z_i consists of bags of subtrees outside P_{rt} attached below t_i.

Let j be the maximum index such that a bag from Z_j contains a cheap vertex. We define the set of vertices $Z_{>j} = \bigcup_{i=j+1}^{q-1} Z_i$. Observe that since $Z_{>j}$ does not contain cheap vertices, then $|Z_{>j}| \leq 2\sqrt{k}$. We also define $V_{0,j}$ as the set of vertices contained in the bags corresponding to nodes $B_0, B_1, \ldots, B_j$ of P_{rt} and set $V_{j+1,q}$ as the set of vertices contained in bags $B_{j+1}, \ldots, B_q = B_t$. Observe also that $\Omega = V_{0,j} \cup V_{j+1,q}$ and by the definition of a block, $V_{0,j}$ is exactly the tail Q_j of the block (B_j, D_j). From Lemma 6 *(1, 4)* we have that $V_{j+1,q} \subseteq B_t \cup N_G(Z_{>j}) \subseteq \Omega$. This follows from the fact that every vertex in B_ℓ for $\ell < q$ has at least one neighbor in G in Z_ℓ.

Let v be a cheap vertex belonging to Z_j. The remaining types of vital potential maximal cliques are defined according to the existence and locations in T of a few other cheap vertices. We use C^v to denote the connected component of $G[D_j] - B_j$ containing v.

Type 3. For vital potential maximal cliques of this type there is a cheap vertex $u \neq v$ belonging to Z_j but not belonging to C^v. Since $V_{0,j} = Q_j$, by Lemma 5 *(3)*, we have that $V_{0,j} = N_H(u) \cap N_H(v)$ and $V_{j+1,q} \subseteq B_q \cup N_G(Z_{>j}) \subseteq \Omega$. Hence we arrive at $\Omega = V_{0,j} \cup V_{j+1,q} = \big(N_H(u) \cap N_H(v)\big) \cup B_t \cup N_G(Z_{>j})$.

The family $\mathcal{C}_3$ consists of all sets of the form $W_1 \cup W_2 \cup W_3$, where:

- $|W_1| \leq 2\sqrt{k}$. Enumerating sets W_1 corresponds to guessing B_t.
- W_2 is the open neighborhood in G of a set of size at most $2\sqrt{k}$. The set W_2 corresponds to $N_G(Z_{>j})$.
- W_3 is the intersection of the sets $N_G(x) \cup A$ and $N_G(y) \cup B$, where $x, y \in V$, and A, B are sets of size at most $\sqrt{k}$. The set W_3 corresponds to intersection of two neighborhoods in H of two cheap vertices u, v.

It is clear that the size of the family $\mathcal{C}_3$ is $2^{\mathcal{O}(\sqrt{k}\log k)}$ and that all sets from $\mathcal{C}_3$ can be listed using $2^{\mathcal{O}(\sqrt{k}\log k)}$ time. It follows from the construction that every Type 3 vital potential maximal clique is in $\mathcal{C}_3$.

Type 4. Let Z be the set of vertices of $V \setminus \Omega$ which do not belong to C^v. In other words, $Z = (V \setminus \Omega) \setminus V(C^v)$. Vital potential maximal cliques of Type 4 are such that Z contains no

cheap vertices. Thus the only cheap vertices among vertices of $V \setminus \Omega$ belong to C^v. In this case, we have that $|Z| \leq 2\sqrt{k}$.

Recall that $\Omega = V_{0,j} \cup V_{j+1,q}$, where $V_{0,j}$ and $V_{j+1,q}$ are the vertices contained in bags of paths from r to t_j, and correspondingly, from t_{j+1} to t in T. By Lemma 6, we have that $V_{j+1,t} = (B_t \cup N_G(Z_{>j})) \setminus N_H(v)$. Furthermore, by Lemma 6 *(4)* we infer that $V_{0,j} = N_G(Z \cup V_{j+1,t})$, so it follows that $\Omega = V_{0,j} \cup V_{j+1,t} = \big(N_G(Z \cup ((B_t \cup N_G(Z_{>j})) \setminus N_H(v)))\big) \cup \big((B_t \cup N_G(Z_{>j})) \setminus N_H(v)\big)$.

We therefore let the family $\mathcal{C}_4$ consist of all sets of $W_1 \cup W_2$, where

- $W_1 = (X_1 \cup N_G(X_2)) \setminus (N_G(v) \cup X_3)$ and the sets X_1, X_2, and X_3 are sets of size at most $2\sqrt{k}$ and $v \in V$. The set W_1 corresponds to guessing $V_{j+1,t}$, X_1 to B_t, X_2 to $Z_{>j}$, and $N_G(v) \cup X_3$ to $N_H(v)$, and
- $W_2 = N_G(X_4 \cup W_1)$, where X_4 is of size at most $2\sqrt{k}$ and corresponds to guessing Z.

By the construction, the size of $\mathcal{C}_4$ is $2^{\mathcal{O}(\sqrt{k}\log k)}$ and all sets from $\mathcal{C}_4$ can be listed in time $2^{\mathcal{O}(\sqrt{k}\log k)}$. It also follows from the construction that every Type 4 vital potential maximal clique is in $\mathcal{C}_4$.

Type 5. The only remaining type of vital potential maximal cliques are such that a cheap vertex $u \neq v$ is in Z. If Ω is not of Type 3, then we know that at least one cheap vertex is in some bag of Z_i, $i < j$. Let $j' < j$ be the largest index smaller than j such that $Z_{j'}$ contains a cheap vertex. Let u be such a vertex.

Let $V_{0,j'}$ be the set of vertices contained in the $B_0, B_1, \ldots, B_{j'}$. Then $V_{0,j'} = Q_{j'}$ and by Item *(3)* of Lemma 5, $V_{0,j'} = N_H(u) \cap N_H(v)$. Let $Z' = \bigcup_{i=j'+1}^{j} Z_i \setminus C^v$.

There is no cheap vertex in Z', hence $|Z'| \leq 2\sqrt{k}$. On the other hand, by Item *(4)* of Lemma 6, $V_{j'+1,j}$, that is, vertices contained in the bags $B_{j'+1}, \ldots, B_j$, is contained in $N_G(V_{j+1,t} \cup Z_{>j}) \cup N_G(Z') \subseteq \Omega$. Thus $\Omega = V_{j+1,t} \cup V_{0,j'} \cup V_{j'+1,j} = V_{j+1,t} \cup \big(N_H(u) \cap N_H(v)\big) \cup \big(N_G(V_{j+1,t} \cup Z_{>j}) \cup N_G(Z')\big)$.

Finally, as in Type 4 we have that $V_{j+1,t} = (B_t \cup N_G(Z_{>j})) \setminus N_H(v)$. Therefore, we let $\mathcal{C}_5$ consist of all sets of the form $W_1 \cup W_2 \cup W_3$, where

- $W_1 = (X_1 \cup N_G(X_2)) \setminus (N_G(v) \cup X_3)$ and sets X_1, X_2, and X_3 are sets of size at most $2\sqrt{k}$ and $v \in V$. As in the previous case for Type 4 vital potential maximal cliques, the set W_1 corresponds to $V_{j+1,t}$.
- $W_2 = (N_G(u) \cup X_4) \cap (N_G(v) \cup X_5)$. Here X_4, X_5, are sets of size at most $\sqrt{k}$ and $u, v \in V$. The set W_2 corresponds to $V_{0,j'}$, while $N_G(u) \cup X_4$ and $N_G(v) \cup X_5$ to $N_H(u)$ and $N_H(v)$ respectively.
- $W_3 = N_G(W_1 \cup X_2) \cup N_G(X_6)$, where X_6 is a set of size at most $2\sqrt{k}$ that was corresponds to Z', while X_2 was already chosen before and corresponds to $Z_{>j}$.

From the construction it immediately follows that the size of family $\mathcal{C}_5$ is $2^{\mathcal{O}(\sqrt{k}\log k)}$, that its elements can be enumerated in the same amount of time, and that every Type 5 vital potential maximal clique is in $\mathcal{C}_5$. Since every vital potential maximal clique is of Type 1, 2, 3, 4, or 5, we can infer the following lemma that formalizes the result of Step B.

▶ **Lemma 9** (Enumeration Lemma). *Let (G, k) be an instance of* Trivially Perfect Completion *such that $|V(G)| = \mathcal{O}(k^3)$. Then in time $2^{\mathcal{O}(\sqrt{k}\log k)}$, we can construct a family $\mathcal{C}$ consisting of $2^{\mathcal{O}(\sqrt{k}\log k)}$ subsets of $V(G)$ such that every vital potential maximal clique of (G, k) is in $\mathcal{C}$.*

Step C. Dynamic programming. At this point we assume that we have the family $\mathcal{C}$ containing all vital potential maximal cliques of (G, k). We start by generating in time

$2^{\mathcal{O}(\sqrt{k}\log k)}$ a family $\mathcal{S}$ of pairs (X, Y), where $X, Y \subseteq V(G)$, such that for every minimal completion S of size at most k, and the corresponding universal clique decomposition $(T, \mathcal{B})$ of $H = G + S$, it holds that every block (B, D) is in $\mathcal{S}$, and the size of $\mathcal{S}$ is $2^{\mathcal{O}(\sqrt{k}\log k)}$. The construction of $\mathcal{S}$ is based on the following observations about blocks and vital potential maximal cliques: Let G be a graph, S a minimal completion and $L = (B, D)$ a block of the universal clique decomposition of $H = G + S$, where H is not a complete graph, with Q being its tail. Then the following holds:

- If L is a leaf block, then $B = \Omega_1 \setminus \Omega_2$ for some vital potential maximal cliques Ω_1 and Ω_2, and $D = B$.
- If L is the root block, then the tail of L is B, $B = \Omega_1 \cap \Omega_2$ for some vital potential maximal cliques Ω_1 and Ω_2, and $D = V$.
- If L is an internal block, then Q is the intersection of two vital potential maximal cliques Ω_1 and Ω_2 of G, $B = Q \setminus \Omega_3$ for some vital potential maximal clique Ω_3, and D is the connected component of $G - (Q \setminus B)$ containing B.

From this observation, we can conclude that by going through all triples $\Omega_1, \Omega_2, \Omega_3$, we can compute the set $\mathcal{S}$ consisting of all blocks (B, D) of minimal completions. We now define value $\mathrm{dp}(B, D)$ as the minimum number of edges needed to be added to $G[D]$ to make it a trivially perfect graph with B being the universal clique contained in the root of the universal clique decomposition. It is easy to derive recurrence equations that enable us to compute all the relevant values of $\mathrm{dp}(\cdot, \cdot)$ using dynamic programming. Finally, the minimum cost of completing G to a trivially perfect graph is equal to $\min_{(B,V(G))\in\mathcal{S}} \mathrm{dp}(B, V(G))$.

3 Completion to threshold and pseudosplit graphs

▶ **Theorem 10 (♠).** Threshold Completion *is solvable in time* $2^{\mathcal{O}(\sqrt{k}\log k)} + \mathcal{O}(kn^4)$.

The proof of Theorem 10 is a combination of the following known techniques: the kernelization algorithm by Guo [12], the chromatic coding technique of Alon et al. [1], also used in the subexponential algorithm of Ghosh et al. [10] for split graphs, and the algorithm of Fomin and Villanger for chain completion [9].

▶ **Theorem 11 (♠).** Pseudosplit Completion *is solvable in time* $2^{\mathcal{O}(\sqrt{k}\log k)} n^{\mathcal{O}(1)}$.

The crucial property of pseudosplit graphs that will be of use is that a pseudosplit graph is either a split graph, or a split graph containing one induced C_5 which is completely non-adjacent to the independent set of the split graph, and completely adjacent to the clique set of the split graph [18]. Hence, assuming we are looking for the latter type of a pseudosplit graph, we can with $\mathcal{O}(n^5)$ overhead guess the correct set that will become the $S = C_5$, and after some preprocessing we can apply the subexponential algorithm of Ghosh et al. [10] solving Split Completion.

4 Lower bounds

To complete our study, we provide lower bounds based on the Exponential Time Hypothesis for the remaining subsets of $\{2K_2, P_4, C_4\}$. More precisely, we prove the following theorem:

▶ **Theorem 12 (♠).** *Unless the Exponential Time Hypothesis (ETH) fails, none of the following problems are solvable in* $2^{o(k)} n^{\mathcal{O}(1)}$ *time:*

- $2K_2$-Free Completion,
- C_4-Free Completion,
- P_4-Free Completion,
- $\{2K_2, P_4\}$-Free Completion *(also known as* Co-Trivially Perfect Completion*)*.

To prove each of the lower bounds above we give a linear reduction from 3Sat. That is, we provide an algorithm that, given a 3-CNF formula φ on n variables and m clauses, produces in polynomial-time an equivalent instance of the problem at hand with parameter $k = \mathcal{O}(n+m)$. Then pipelining the reduction with the assumed subexponential parameterized algorithm for the problem would give an algorithm for 3Sat working in $2^{o(n+m)}$ time. The existence of such an algorithm, however, would contradict ETH by the sparsification lemma of Impagliazzo et al. [14].

Our reductions follow in spirit those of, for instance Komusiewicz and Uhlmann [16], or Fomin et al. [8]: we create a gadget graph for each variable and each clause, and carefully wire the gadgets together so that they encode the input instance. However, since we are dealing with very particular graph classes with a lot of structure, the design and analysis of the gadgets requires a number of non-trivial ideas.

5 Conclusion and open problems

In this paper, we provided several upper and lower subexponential parameterized bounds for $\mathcal{F}$-Completion. The most natural open question would be to ask for a dichotomy type of result characterizing for which sets $\mathcal{F}$, $\mathcal{F}$-Completion problems are in P, in SUBEPT, and not in SUBEPT (under ETH). Keeping in mind the lack of such characterization concerning classes P and NP, an answer to this question can be very non-trivial. Even a more modest task—deriving general arguments explaining what causes a completion problem to be in SUBEPT—is an important open question.

Similarly, from an algorithmic perspective obtaining generic subexponential algorithms for completion problems would be a big step forwards. With the current knowledge, for different cases of $\mathcal{F}$, the algorithms are built on different ideas like chromatic coding, potential maximal cliques, k-cuts, etc. and each new case requires special treatment.

Finally, some concrete problems. We have the chain of graph classes

$$\textbf{threshold} \subset \textbf{trivially perfect} \subset \textbf{interval} \subset \textbf{chordal},$$

corresponding to the parameters vertex cover, treedepth, pathwidth, and treewidth, in the sense that the width parameter is the minimum, over all completions to the graph class mentioned, of the size of the maximum clique (± 1). We know that all of these problems have subexponential completion problems, except for Interval Completion. The problem is known to be in FPT [21]. It is natural to ask whether or not this problem also belongs to SUBEPT. Another chain connecting graph classes to width parameters is the chain corresponding to bandwidth, pathwidth and treewidth, **proper interval** $\subset$ **interval** $\subset$ **chordal**. The existence of a subexponential algorithm for Proper Interval Completion is also open.

References

1 Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In *ICALP 2009*, volume 5555 of *LNCS*, pages 49–58. Springer, 2009.

2 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretial Computer Science*, 209(1-2):1–45, 1998.

3 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes. A Survey*. SIAM Monographs on Discrete Mathematics and Applications. Philadelphia, USA, 1999.

4 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

5 Leizhen Cai and Yufei Cai. Incompressibility of H-free edge modification problems. In *IPEC 2013*. To appear.

6 Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs. *Journal of the ACM*, 52(6):866–893, 2005.

7 Jörg Flum and Martin Grohe. *Parameterized complexity theory*. Springer-Verlag New York Inc, 2006.

8 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing. In *STACS 2013*, volume 20 of *LIPIcs*, pages 32–43, 2013.

9 Fedor V. Fomin and Yngve Villanger. Subexponential parameterized algorithm for minimum fill-in. In *SODA 2012*, pages 1737–1746. SIAM, 2012.

10 Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, and M.S. Ramanujan. Faster parameterized algorithms for deletion to split graphs. In *SWAT 2012*, volume 7357 of *LNCS*, pages 107–118. Springer, 2012.

11 Sylvain Guillemot, Frédéric Havet, Christophe Paul, and Anthony Perez. On the (non-)existence of polynomial kernels for P_l-free edge modification problems. *Algorithmica*, 65(4):900–926, 2013.

12 Jiong Guo. Problem kernels for NP-complete edge deletion problems: Split and related graphs. In *ISAAC 2007*, volume 4835 of *LNCS*, pages 915–926. Springer, 2007.

13 Pinar Heggernes, and Federico Mancini. Minimal split completions. *Discrete Applied Mathematics*, 157(12):2659—2669, 2009.

14 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

15 Haim Kaplan, Ron Shamir, and Robert E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal on Computing*, 28:1906–1922, May 1999.

16 Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012.

17 Stefan Kratsch and Magnus Wahlström. Two edge modification problems without polynomial kernels. In *IWPEC 2009*, volume 5917 of *LNCS*, pages 264–275. Springer, 2009.

18 Frédéric Maffray and Myriam Preissmann. Linear recognition of pseudo-split graphs. *Discrete Applied Mathematics*, 52(3):307–312, 1994.

19 Assaf Natanzon, Ron Shamir, and Roded Sharan. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM Journal on Computing*, 30:1067–1079, 2000.

20 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

21 Yngve Villanger, Pinar Heggernes, Christophe Paul, and Jan Arne Telle. Interval completion is fixed parameter tractable. *SIAM Journal on Computing*, 38(5):2007–2020, 2009.

22 Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, 1981.

From Small Space to Small Width in Resolution

Yuval Filmus[1], Massimo Lauria[2], Mladen Mikša[2], Jakob Nordström[2], and Marc Vinyals[2]

1 **Simons Institute for the Theory of Computing, University of California, Berkeley, USA**
2 **School of Computer Science and Communication, KTH Royal Institute of Technology, Stockholm, Sweden**

Abstract

In 2003, Atserias and Dalmau resolved a major open question about the resolution proof system by establishing that the space complexity of formulas is always an upper bound on the width needed to refute them. Their proof is beautiful but somewhat mysterious in that it relies heavily on tools from finite model theory. We give an alternative, completely elementary, proof that works by simple syntactic manipulations of resolution refutations. As a by-product, we develop a "black-box" technique for proving space lower bounds via a "static" complexity measure that works against any resolution refutation—previous techniques have been inherently adaptive. We conclude by showing that the related question for polynomial calculus (i.e., whether space is an upper bound on degree) seems unlikely to be resolvable by similar methods.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes – Relations among complexity measures, F.4.1 Mathematical Logic – Computational logic, F.2.2 Nonnumerical Algorithms and Problems – Complexity of proof procedures

Keywords and phrases proof complexity, resolution, width, space, polynomial calculus, PCR

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.300

1 Introduction

A *resolution proof* for, or *resolution refutation* of, an unsatisfiable formula F in conjunctive normal form (CNF) is a sequence of disjunctive clauses $(C_1, C_2, \ldots, C_\tau)$, where every clause C_t is either a member of F or is logically implied by two previous clauses, and where the final clause is the contradictory empty clause $\bot$ containing no literals. Resolution is arguably the most well-studied proof system in propositional proof complexity, and has served as a natural starting point in the quest to prove lower bounds for increasingly stronger proof systems on *proof length/size* (which for resolution is the number of clauses in a proof).

Resolution is also intimately connected to SAT solving, in that it lies at the foundation of state-of-the-art SAT solvers using so-called conflict-driven clause learning (CDCL). This connection has motivated the study of *proof space* as a second interesting complexity measure for resolution. The space usage at some step t in a proof is measured as

the number of clauses occurring before C_t that will be used to derive clauses after C_t,

and the space of a proof is obtained by taking the maximum over all steps t.

For both of these complexity measures, it turns out that a key role is played by the auxiliary measure of *width*, i.e., the size of a largest clause in the proof. In a celebrated result, Ben-Sasson and Wigderson [10] showed that there are short resolution refutations of a formula if and only if there are also (reasonably) narrow ones, and almost all known lower bounds on resolution length can be (re)derived using this connection.



© Yuval Filmus, Massimo Lauria, Mladen Mikša, Jakob Nordström, and Marc Vinyals;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 300–311



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



In 2003, Atserias and Dalmau (journal version in [2]) established that width also provides lower bounds on space, resolving a problem that had been open since the space complexity of propositional proofs started being studied in the late 1990s ([1, 13]). This means that for space also, almost all known lower bounds can be rederived by using width lower bounds and appealing to [2]. This is not a two-way connection, however, in that formulas of almost worst-case space complexity may require only constant width as shown in [8].

The starting point of our work is the lower bound on space in terms of width in [2]. This is a very elegant but also magical proof in that it translates the whole problem to Ehrenfeucht–Fraïssé games in finite model theory, and shows that resolution space and width correspond to strategies for two opposite players in such games. Unfortunately, this also means that one obtains essentially no insight into what is happening on the proof complexity side (other than that the bound on space in terms of width is true). It has remained an open problem to give a more explicit, proof complexity theoretic, argument.

In this paper, we give a purely combinatorial proof in terms of simple syntactic manipulations of resolution refutations. To summarize in one sentence, we study the conjunctions of clauses in memory at each time step in a small-space refutation, negate these conjunctions and then expand them to conjunctive normal form again, and finally argue that the new sets of clauses listed in reverse order (essentially) constitute a small-width refutation of the same formula.

This new, simple proof also allows us to obtain a new technique for proving space lower bounds. This approach is reminiscent of [10] in that one defines a static "progress measure" on refutations and argues that when a refutation has made substantial progress it must have high complexity with respect to the proof complexity measure under study. Previous lower bounds on space have been inherently adaptive and in that sense less explicit.

One other important motivation for our work was the hope that a simplified proof of the space-width inequality would serve as a stepping stone to resolving the analogous question for the polynomial calculus proof system, where the width of clauses corresponds to the *degree* of polynomials. While we recently showed in [14] the analogue of [8] that there are formulas of worst-case space complexity that require only constant degree, the question of whether degree lower bounds imply space lower bounds remains open. Unfortunately, as discussed towards the end of this paper we show that it appears unlikely that this question can be resolved by methods similar to our proof of the corresponding inequality for resolution.

The rest of this paper is organized as follows. After some brief preliminaries in Section 2, we present the new proof of the space-width inequality in [2] in Section 3. In Section 4 we showcase the new technique for space lower bounds by studying so-called Tseitin formulas. Section 5 explains why we believe it is unlikely that our methods will extend to polynomial calculus. Some concluding remarks are given in Section 6.

2 Preliminaries

Let us start by a brief review of the preliminaries. The following material is standard and can be found, e.g., in the survey [18].

A *literal* over a Boolean variable x is either the variable x itself (a *positive literal*) or its negation that is denoted either as $\neg x$ or $\overline{x}$ (a *negative literal*). We define $\overline{\overline{x}} = x$. A *clause* $C = a_1 \vee \cdots \vee a_k$ is a disjunction of literals and a *term* $T = a_1 \wedge \cdots \wedge a_k$ is a conjunction of literals. We denote the empty clause by $\bot$ and the empty term by $\emptyset$. The logical negation of a clause $C = a_1 \vee \cdots \vee a_k$ is the term $\overline{a}_1 \wedge \cdots \wedge \overline{a}_k$ that consists of the negations of the literals in the clause. We will sometimes use the notation $\neg C$ or $\overline{C}$ for the term corresponding to the

negation of a clause and $\neg T$ or $\overline{T}$ for the clause negating a term. A clause (term) is *trivial* if it contains both a variable and its negation. For the proof systems we study, trivial clauses and terms can always be eliminated without any loss of generality.

A clause C' *subsumes* clause C if every literal from C' also appears in C. A *k-clause (k-term)* is a clause (term) that contains at most k literals. A *CNF formula* $F = C_1 \wedge \cdots \wedge C_m$ is a conjunction of clauses, and a *DNF formula* $F = T_1 \vee \cdots \vee T_m$ is a disjunction of terms. A *k-CNF formula (k-DNF formula)* is a CNF formula (DNF formula) consisting of k-clauses (k-terms). In this paper we only consider CNF formulas that do not contain the empty clause. We think of clauses, terms, and CNF formulas as sets: the order of elements is irrelevant and there are no repetitions.

Let us next describe a slight generalization of the resolution proof system by Krajíček [16], who introduced the the family of *r-DNF resolution* proof systems, denoted $\mathcal{R}(r)$, as an intermediate step between resolution and depth-2 Frege systems. An *r-DNF resolution configuration* $\mathbb{C}$ is a set of r-DNF formulas. An *r-DNF resolution refutation* of a CNF formula F is a sequence of configurations $(\mathbb{C}_0, \ldots, \mathbb{C}_\tau)$ such that $\mathbb{C}_0 = \emptyset$, $\bot \in \mathbb{C}_\tau$, and for $1 \leq t \leq \tau$ we obtain $\mathbb{C}_t$ from $\mathbb{C}_{t-1}$ by one of the following steps:

Axiom download $\mathbb{C}_t = \mathbb{C}_{t-1} \cup \{A\}$, where A is a clause in F (sometimes referred to as an *axiom clause*).

Inference $\mathbb{C}_t = \mathbb{C}_{t-1} \cup \{D\}$, where D is inferred by one of the following rules (where G, H denote r-DNF formulas, T, T' denote r-terms, and $a_1, \ldots, a_r$ denote literals):

r-cut $\dfrac{(a_1 \wedge \cdots \wedge a_{r'}) \vee G \qquad \overline{a}_1 \vee \cdots \vee \overline{a}_{r'} \vee H}{G \vee H}$, where $r' \leq r$.

$\wedge$-introduction $\dfrac{G \vee T \qquad G \vee T'}{G \vee (T \wedge T')}$, as long as $|T \cup T'| \leq r$.

$\wedge$-elimination $\dfrac{G \vee T}{G \vee T'}$ for any non-empty $T' \subseteq T$.

Weakening $\dfrac{G}{G \vee H}$ for any r-DNF formula H.

Erasure $\mathbb{C}_t = \mathbb{C}_{t-1} \setminus \{C\}$, where C is an r-DNF formula in $\mathbb{C}_{t-1}$.

When setting $r = 1$ we obtain the standard *resolution* proof system. In this case the only nontrivial inference rules are weakening and r-cut, where the former can be eliminated without loss of generality (but is sometimes convenient to have for technical purposes) and the latter simplifies to the *resolution rule* $\frac{C \vee x \quad D \vee \overline{x}}{C \vee D}$. We identify a resolution configuration $\mathbb{C}$ with the CNF formula $\bigwedge_{C \in \mathbb{C}} C$.

The *length* $L(\pi)$ of an r-DNF resolution refutation π is the number of download and inference steps, and the *space* $Sp(\pi)$ is the maximal number of r-DNF formulas in any configuration in π. We define the length $L_{\mathcal{R}(r)}(F \vdash \bot)$ and the space $Sp_{\mathcal{R}(r)}(F \vdash \bot)$ of refuting a formula F in r-DNF resolution by taking the minimum over all refutations F with respect to the relevant measure. We drop the proof system $\mathcal{R}(r)$ from this notation when it is clear from context.

For the resolution proof system, we also define the *width* $W(\pi)$ of a resolution refutation π as the size of a largest clause in π, and taking the minimum over all resolution refutations we obtain the width $W(F \vdash \bot)$ of refuting F. We remark that in the context of resolution the space measure defined above is sometimes referred to as *clause space* to distinguish it from other space measures studied for this proof system.

3 From Space to Width

In this section we present our new combinatorial proof that width is a lower bound for clause space in resolution. The formal statement of the theorem is as follows (where we recall that in this article all CNF formulas are assumed to be non-trivial in that they do not contain the contradictory empty clause).

▶ **Theorem 1** ([2]). *Let F be a k-CNF formula and let $\pi : F \vdash \bot$ be a resolution refutation in space $Sp(\pi) = s$. Then there is a resolution refutation π' of F in width $W(\pi') \leq s + k - 3$.*

The proof idea is to take the refutation π in space s, negate the configurations one by one, rewrite them as equivalent sets of disjunctive clauses, and list these sets of clauses in reverse order. This forms the skeleton of the new refutation, where all clauses have width at most s. To see this, note that each configuration in the original refutation is the conjunction of at most s clauses. Therefore, the negation of such a configuration is a disjunction of at most s terms, which is equivalent (using distributivity) to a conjunction of clauses of width at most s. To obtain a legal resolution refutation, we need to fill in the gaps between adjacent sets of clauses. In this process the width increases slightly from s to $s + k - 3$.

Before presenting the full proof, we need some technical results. We start by giving a formal definition of what a negated configuration is.

▶ **Definition 2.** The *negated configuration* $\operatorname{neg}(\mathbb{C})$ of a configuration $\mathbb{C}$ is defined by induction on the number of clauses in $\mathbb{C}$:

- $\operatorname{neg}(\emptyset) = \{\bot\}$,
- $\operatorname{neg}(\mathbb{C} \cup \{C\}) = \{D \vee \overline{a} \mid D \in \operatorname{neg}(\mathbb{C}) \text{ and } a \in C\}$,

where we remove trivial and subsumed clauses from the final configuration.

In the proof we will use a different characterization of negated configurations that is easier to work with.

▶ **Proposition 3.** *The negated configuration $\operatorname{neg}(\mathbb{C})$ is the set of all minimal (non-trivial) clauses C such that $\neg C$ implies the configuration $\mathbb{C}$. That is*

$$\operatorname{neg}(\mathbb{C}) = \{C \mid \neg C \vDash \mathbb{C} \text{ and for every } C' \subseteq C \text{ it holds that } \neg C' \nvDash \mathbb{C}\} .$$

Proof. Let us fix the configuration $\mathbb{C}$ and let $\mathbb{D}$ denote the set of all minimal clauses implying $\mathbb{C}$. We prove that for each clause $C \in \operatorname{neg}(\mathbb{C})$ there is a clause $C' \in \mathbb{D}$ such that $C' \subseteq C$ and vice versa. The proposition then follows because by definition neither $\mathbb{D}$ nor $\operatorname{neg}(\mathbb{C})$ contains subsumed clauses.

First, let $C \in \operatorname{neg}(\mathbb{C})$. By the definition of $\operatorname{neg}(\mathbb{C})$ we know that for every clause $D \in \mathbb{C}$ the clause C contains the negation of some literal from D. Hence, $\neg C$ implies $\mathbb{C}$ as it is a conjunction of literals from each clause in $\mathbb{C}$. By taking the minimal clause $C' \subseteq C$ such that $\neg C' \vDash \mathbb{C}$ we have that $C' \in \mathbb{D}$.

In the opposite direction, let $C \in \mathbb{D}$ and let us show that C must contain a negation of some literal in D for every clause $D \in \mathbb{C}$. Assume for the sake of contradiction that $D \in \mathbb{C}$ is a clause such that none of its literals has a negation appearing in C. Let α be a total truth value assignment that satisfies $\neg C$ (such an assignment exists because C is non-trivial). By assumption, flipping the variables in α so that they falsify D cannot falsify $\neg C$. Therefore, we can find an assignment that satisfies $\neg C$ but falsifies $D \in \mathbb{C}$, which contradicts the definition of $\mathbb{D}$. Hence, the clause C must contain a negation of some literal in D for every $D \in \mathbb{C}$ and by the definition of $\operatorname{neg}(\mathbb{C})$ there is a $C' \in \operatorname{neg}(\mathbb{C})$ such that $C' \subseteq C$. ◀

The following observation, which formalizes the main idea behind the concept of negated configurations, is an immediate consequence of Proposition 3.

▶ **Observation 4.** *An assignment satisfies a clause configuration* $\mathbb{C}$ *if and only if it falsifies the negated clause configuration* $\mathrm{neg}(\mathbb{C})$*. That is,* $\mathbb{C}$ *is logically equivalent to* $\neg\mathrm{neg}(\mathbb{C})$*.*

Recall that what we want to do is to take a resolution refutation $\pi = (\mathbb{C}_0, \mathbb{C}_1, \ldots, \mathbb{C}_\tau)$ and argue that if π has small space, then the reversed sequence of negated configurations $\pi' = (\mathrm{neg}(\mathbb{C}_\tau), \mathrm{neg}(\mathbb{C}_{\tau-1}), \ldots, \mathrm{neg}(\mathbb{C}_0))$ has small width. However, as noted above π' is not necessarily a legal resolution refutation. Hence, we need to show how to derive the clauses in each configuration of the negated refutation without increasing the width by too much. We do so by a case analysis over the derivation steps in the original refutation, i.e., axiom download, clause inference, or clause erasure. The following lemma show that for inference and erasure steps all that is needed in the reverse direction is to apply weakening.

▶ **Lemma 5.** *If* $\mathbb{C} \vDash \mathbb{C}'$*, then for every clause* $C \in \mathrm{neg}(\mathbb{C})$ *there is a clause* $C' \in \mathrm{neg}(\mathbb{C}')$ *such that* C *is a weakening of* C'*.*

Proof. For any clause C is in $\mathrm{neg}(\mathbb{C})$ it holds by Proposition 3 that $\neg C \vDash \mathbb{C}$. Since $\mathbb{C} \vDash \mathbb{C}'$, this in turns implies that $\neg C \vDash \mathbb{C}'$. Applying Proposition 3 again, we conclude that there exists a clause $C' \subseteq C$ such that $C' \in \mathrm{neg}(\mathbb{C}')$. ◀

The only time in a refutation $\pi = (\mathbb{C}_0, \mathbb{C}_1, \ldots, \mathbb{C}_\tau)$ when it does not hold that $\mathbb{C}_{t-1} \vDash \mathbb{C}_t$ is when an axiom clause is downloaded at time t, and such derivation steps will require a bit more careful analysis. We provide such an analysis in the full proof of Theorem 1, which we are now ready to present.

Proof of Theorem 1. Let $\pi = (\mathbb{C}_0, \mathbb{C}_1, \ldots, \mathbb{C}_\tau)$ be a resolution refutation of F in space s. For every configuration

$\mathbb{C}_t \in \pi$, let $\mathbb{D}_t$ denote the corresponding negated configuration $\mathrm{neg}(\mathbb{C}_t)$. By the discussion preceding Definition 2, it is clear than each clause of $\mathbb{C}_t$ contributes at most one literal to each clause of $\mathbb{D}_t$. Hence, the clauses of $\mathbb{D}_t$ have width at most s. We need to show how to transform the sequence $\pi' = (\mathbb{D}_\tau, \mathbb{D}_{\tau-1}, \ldots, \mathbb{D}_0)$ into a legal resolution refutation of width at most $s + k - 3$.

The initial configuration of the new refutation is $\mathbb{D}_\tau$ itself, which is empty by Definition 2. If $\mathbb{C}_{t+1}$ follows $\mathbb{C}_t$ by inference or erasure, then we can derive any clause of $\mathbb{D}_t$ from a clause of $\mathbb{D}_{t+1}$ by weakening, as proven in Lemma 5. If $\mathbb{C}_{t+1}$ follows $\mathbb{C}_t$ by axiom download, then we can derive $\mathbb{D}_t$ from $\mathbb{D}_{t+1}$ in width at most $s+k-3$, as we show below. The last configuration $\mathbb{D}_0$ includes the empty clause $\bot$ by Definition 2, so the new refutation is complete.

It remains to take care of the case of axiom download. We claim that we can assume without loss of generality that prior to each axiom download step the space of the configuration $\mathbb{C}_t$ is at most $s - 2$. Otherwise, immediately after the axiom download step the proof π needs to erase a clause in order to maintain the space bound s. By reordering the axiom download and clause erasure steps we get a valid refutation of F for which it holds that $Sp(\mathbb{C}_t) \leq s - 2$.

Suppose $\mathbb{C}_{t+1} = \mathbb{C}_t \cup \{A\}$ for some axiom $A = a_1 \vee \cdots \vee a_\ell$, with $\ell \leq k$. Consider now some clause C that is in the negated configuration $\mathbb{D}_t$ and that does not belong to $\mathbb{D}_{t+1}$. Again by Definition 2, the clause C has at most one literal per clause in $\mathbb{C}_t$, so $W(C) \leq s - 2$. To derive C from $\mathbb{D}_{t+1}$ we first download axiom A and then show how to derive C from the clauses in $\mathbb{D}_{t+1} \cup \{A\}$.

First, note that all clauses $C_a = C \vee \overline{a}$ are either contained in or are weakenings of clauses in $\mathbb{D}_{t+1}$. This follows easily from Definition 2 as adding an axiom A to the configuration $\mathbb{C}_t$ results in adding negations of literals from A to all clauses $C \in \mathbb{D}_t$. Hence, we can obtain C by the following derivation:

$$\dfrac{\dfrac{\dfrac{A = a_1 \vee \cdots \vee a_\ell \qquad C_{a_1} = C \vee \overline{a}_1}{C \vee a_2 \vee \cdots \vee a_\ell} \qquad C_{a_2} = C \vee \overline{a}_2}{C \vee a_3 \vee \cdots \vee a_\ell}}{\vdots}$$

$$\dfrac{C \vee a_\ell \qquad\qquad C_{a_\ell} = C \vee \overline{a}_\ell}{C}$$

When C is the empty clause, the width of this derivation is upper-bounded by $W(A) \leq k$. Otherwise, it is upper bounded by $W(C) + W(A) - 1 \leq s + k - 3$. Any resolution refutation has space at least 3 (unless the formula contains the empty clause itself), so the width of π' is upper-bounded by $W(\pi') \leq s + k - 3$. ◀

The proof of Theorem 1 also works for r-DNF resolution, with some loss in parameters. We now define the negated configuration of an r-DNF resolution configuration and sketch a proof that resolution width is a lower bound for r-DNF resolution space.

▶ **Theorem 6.** *Let F be a k-CNF formula and let $\pi : F \vdash \bot$ be an r-DNF resolution refutation of F in space $Sp(\pi) \leq s$. Then there exists a resolution refutation π' of F in width at most $W(\pi') \leq (s-2)r + k - 1$.*

Proof sketch. We define the negated configuration $\mathrm{neg}_{\mathcal{R}(r)}(\mathbb{C})$ of a $\mathcal{R}(r)$ configuration to be

- $\mathrm{neg}_{\mathcal{R}(r)}(\emptyset) = \{\bot\}$,
- $\mathrm{neg}_{\mathcal{R}(r)}(\mathbb{C} \cup \{C\}) = \{D \vee \overline{T} \mid D \in \mathrm{neg}_{\mathcal{R}(r)}(\mathbb{C}) \text{ and } T \in C\}$,

with trivial and subsumed clauses removed. It is easy to see that an s space r-DNF configuration gets transformed into a resolution configuration of width sr. We can prove an analogue of Proposition 3 for this definition of the negated configuration and, hence, the analogue of Lemma 5 easily follows. The case of axiom download is the same as in the proof of Theorem 1 as axioms are clauses. Hence, running the negated refutation backwards we get a resolution refutation of F in width $(s-2)r + k - 1$. ◀

4 A Static Technique for Proving Space Lower Bounds

Looking at the proof complexity literature, the techniques used to prove lower bounds for resolution length and width (e.g., [10, 11, 15, 19]) are essentially different from ones used to prove resolution space lower bounds (e.g., [1, 7, 13], in that the former are *static* or *oblivious* while the latter are *dynamic*.

Lower bounds on resolution length typically have the following general structure: if a refutation is too short, then we obtain a contradiction by applying a suitable random restriction (the length of the proof figures in by way of a union bound); so any refutation must be long. When proving lower bounds on resolution width, one defines a complexity measure, and uses the properties of this measure to show that every refutation must contain a complex clause; in a second step one then argues that such a complex clause must be wide.

In contrast, most lower bound proofs for resolution space use an *adversary argument*. Assuming that the resolution derivation is in small space, one constructs a satisfying assignment for each clause configuration. Such assignments are updated inductively as the derivation

progresses, and one shows that the update is always possible given the assumption that the space is small. This in turn shows that the contradictory empty clause can never be reached, implying a space lower bound on refutations. The essential feature separating this kind of proofs from the ones above is that the satisfying assignments arising during the proof *depend on the history of the derivation*; in contrast, the complexity measures in width lower bounds are defined once and for all, as are the distributions of random restrictions in length lower bounds.

In this section we present a *static* lower bound on resolution space. Our proof combines the ideas of Section 3 and the complexity measure for clauses used in [10]. We define a complexity measure for configurations which can be used to prove space lower bounds along the lines of the width lower bounds mentioned above.

This approach works in general in that complexity measure for clauses can be transformed into a complexity measure for configurations. This turns many width lower bound techniques into space lower bound ones (e.g., width lower bounds for random 3-CNF formulas.) In this section we give a concrete example of this for Tseitin formulas, which are a family of CNFs encoding a specific type of systems of linear equations.

▶ **Definition 7** (Tseitin formula)**.** Let $G = (V, E)$ be an undirected graph and $\chi \colon V \to \{0, 1\}$ be a function. Identify every edge $e \in E$ with a variable x_e, and let $PARITY_{v,\chi}$ denote the CNF encoding of the constraint $\sum_{e \ni v} x_e = \chi(v) \pmod 2$ for any vertex $v \in V$. Then the *Tseitin formula* over G with respect to χ is $Ts(G, \chi) = \bigwedge_{v \in V} PARITY_{v,\chi}$.

When the degree of G is bounded by d, $PARITY_{v,\chi}$ has at most 2^{d-1} clauses, all of width at most d, and hence $Ts(G, \chi)$ is a d-CNF formula with at most $2^{d-1}|V|$ clauses. We say that a set of vertices U has *odd (even) charge* if $\sum_{u \in U} \chi(u)$ is odd (even). A simple counting argument shows that when $V(G)$ has odd charge, $Ts(G, \chi)$ is unsatisfiable. On the other hand, if G is connected then for each $v \in V$ it is always possible to satisfy the constraints $PARITY_{u,\chi}$ for all $u \neq v$. If G is a good expander, then large space is needed to refute $Ts(G, \chi)$.

▶ **Definition 8** (Edge expansion)**.** The graph $G = (V, E)$ is an (s, δ)-*edge expander* if for every set of vertices $U \subseteq V$ such that $|U| \leq s$, the set of edges $E\big(U\big)$ has size at least $\delta|U|$, where $E\big(U\big)$ is the set of edges of G with exactly one vertex in U.

▶ **Theorem 9.** *For a d-degree (s, δ)-edge expander G it holds that $Sp(Ts(G, \chi)) \geq \delta s/d$.*

We remark that Theorem 9 was originally proven in [1, 13] (and with slightly better parameters, as discussed below).

For the rest of this section we fix a particular connected graph G of degree d, a function χ with respect to which $V(G)$ has odd charge, and the corresponding Tseitin formula $Ts(G, \chi)$. The main tool used to prove Theorem 9 is a complexity measure for configurations. We show that if G is a good expander, then every refutation of $Ts(G, \chi)$ must have a configuration with intermediate measure. We conclude the proof by showing that the space of a configuration is at least the value of its measure, if the latter falls within a specific range of values.

We first define our configuration complexity measure for terms (i.e. configurations consisting of unit clauses), and then we extend it to general configurations. In words, the term complexity measure is the smallest number of parity axioms of $Ts(G, \chi)$ that collectively contradict the term, and the configuration complexity measure is the maximum measure over all terms that imply the configuration.

▶ **Definition 10** (Configuration complexity measure)**.** The *term complexity measure* $\nu(T)$ of a term T is $\nu(T) = \min\big\{|V'| : V' \subseteq V \text{ and } T \wedge \bigwedge_{v \in V'} PARITY_{v,\chi} \vDash \bot\big\}$.

The *configuration complexity measure* $\mu(\mathbb{C})$ of a resolution configuration $\mathbb{C}$ is defined as $\mu(\mathbb{C}) = \max\{\nu(T) : T \vDash \mathbb{C}\}$. When only trivial terms T imply $\mathbb{C}$, we have $\mu(\mathbb{C}) = 0$.

We now introduce the convenient concept of *witness* for the measure: a witness for $\nu(T)$ is a set of vertices V^* for which $\nu(T) = |V^*|$ and $T \wedge \bigwedge_{v \in V^*} PARITY_{v,\chi} \vDash \bot$. Similarly, for configurations, a witness for $\mu(\mathbb{C})$ is a term T^* for which $\nu(T^*) = \mu(\mathbb{C})$ and $T^* \vDash \mathbb{C}$.

There is a big gap between the measure of the initial and final configurations of a refutation, and we will see that the measure does not change much at each step. Hence, the refutation must pass through a configuration of intermediate measure. Formally, we have that $\mu(\emptyset) = |V|$, because the empty term implies $\emptyset$ and has measure $|V|$, and $\mu(\mathbb{C}) = 0$ when $\bot \in \mathbb{C}$, as only trivial terms imply contradiction.

To study how the measure changes during the refutation, we look separately at what happens at each type of step. As in the proof of Theorem 1, we can deal with inference and clause erasure steps together.

▶ **Lemma 11.** *If $\mathbb{C} \vDash \mathbb{C}'$ then $\mu(\mathbb{C}) \leq \mu(\mathbb{C}')$.*

Proof. Let T^* be a witness for $\mu(\mathbb{C})$. Then, $T^* \vDash \mathbb{C}$ and, hence, we also have $T^* \vDash \mathbb{C}'$. Therefore, $\mu(\mathbb{C}') \geq \nu(T^*) = \mu(\mathbb{C})$. ◀

Again, as in the proof of Theorem 1, axiom download requires most of the work. We show that if the graph has constant degree d, then the measure decreases slowly.

▶ **Lemma 12.** *For a clause A in $Ts(G, \chi)$ and a graph G of bounded degree d, if $\mathbb{C}' = \mathbb{C} \cup \{A\}$ then $d \cdot \mu(\mathbb{C}') + 1 \geq \mu(\mathbb{C})$.*

Proof. Fix a witness T^* for $\mu(\mathbb{C})$. Since $\mu(\mathbb{C}) = \nu(T^*)$, to prove the lemma we need to upper-bound the value $\nu(T^*)$ by $d \cdot \mu(\mathbb{C}') + 1$.

For any literal a in A, we know that $T^* \wedge a$ implies $\mathbb{C}'$ because T^* implies $\mathbb{C}$ and a implies A. Hence, it holds that $\mu(\mathbb{C}') \geq \nu(T^* \wedge a)$, and so it will be sufficient to relate $\nu(T^*)$ to the values $\nu(T^* \wedge a)$. To this end, we look at the set of vertices $V^* = \bigcup_{a \in A} V_a \cup \{v_A\}$, where each V_a is a witness for the corresponding measure $\nu(T^* \wedge a)$, and v_A is the vertex such that $A \in PARITY_{v_A,\chi}$. Note that by definition we have $|V_a| = \nu(T^* \wedge a)$ for every $a \in A$ and also that $|V^*| \leq \sum_{a \in A} |V_a| + 1$, which can in turn be bounded by $d \cdot \mu(\mathbb{C}') + 1$ because A has at most d literals.

We conclude the proof by showing that $T^* \wedge \bigwedge_{v \in V^*} PARITY_{v,\chi} \vDash \bot$, which shows that $\nu(T^*) \leq |V^*|$. The implication holds because any assignment either falsifies clause A, and so falsifies $PARITY_{v_A,\chi}$, or one of the literals $a \in A$ is satisfied. But then we have as a subformula $T^* \wedge \bigwedge_{v \in V_a} PARITY_{v,\chi}$, which is unsatisfiable by the definition of V_a when a is true. The bound $\nu(T^*) \leq |V^*|$ then follows, and so $\mu(\mathbb{C}) \leq |V^*| \leq d \cdot \mu(\mathbb{C}') + 1$. ◀

The preceding results imply that every resolution refutation of the Tseitin formula has a configuration of intermediate complexity. This holds because every refutation starts with a configuration of measure $|V|$ and needs to reach the configuration of measure 0, while at each step the measure drops at most a factor $1/d$ by previous lemmas.

▶ **Corollary 13.** *For any resolution refutation π of a Tseitin formula $Ts(G, \chi)$ over a connected graph G of bounded degree d and any positive integer $r \leq |V|$ there exists a configuration $\mathbb{C} \in \pi$ such that the configuration complexity measure is bounded by $r/d \leq \mu(\mathbb{C}) \leq r$.*

It remains to show that a configuration having intermediate measure must also have large space. Note that $\nu(T)$ is a monotone decreasing function, since $T \subseteq T'$ implies $\nu(T) \geq \nu(T')$

by definition. Hence, we only need to look at minimal terms T for which $T \vDash \mathbb{C}$ in order to determine $\mu(\mathbb{C})$.

In the case of expander graphs we have a space lower bound from the configuration complexity measure.

▶ **Lemma 14.** *Let G be an (s, δ)-edge expander graph. For every configuration $\mathbb{C}$ satisfying $\mu(\mathbb{C}) \leq s$ it holds that $Sp(\mathbb{C}) \geq \delta \cdot \mu(\mathbb{C})$.*

Proof. To prove the lemma, we lower-bound the size of a minimal witness T^* for $\mu(\mathbb{C})$ and then use the bound $Sp(\mathbb{C}) \geq |T^*|$. If only trivial terms imply $\mathbb{C}$ then the lemma immediately follows because $\mu(\mathbb{C}) = 0$. The latter bound follows by noting that every literal of T^* must imply at least one clause in $\mathbb{C}$. Fix T^* to be a minimal witness for $\mu(\mathbb{C})$ and let V^* be a witness for $\nu(T^*)$. Note that $|V^*| = \mu(\mathbb{C})$. We prove that T^* must contain a variable for every edge in $E(V^*)$.

Towards contradiction, assume that T^* does not contain some x_e for an edge e in $E(V^*)$, and let v_e be the vertex in V^* incident to e. Let α be an assignment that satisfies $T^* \wedge \bigwedge_{v \in V^* \setminus \{v_e\}} PARITY_{v,\chi}$. Such an assignment must exist as otherwise V^* would not be a witness for $\nu(T^*)$. We can modify α by changing the value of x_e so that $PARITY_{v_e,\chi}$ is satisfied. By the assumption, the new assignment α' still satisfies T^* and $\bigwedge_{v \in V^* \setminus \{v_e\}} PARITY_{v,\chi}$ as neither contains the variable x_e. Thus, we have found an assignment satisfying $T^* \wedge \bigwedge_{v \in V^*} PARITY_{v,\chi}$, which is a contradiction.

Hence, the term T^* contains a variable for every edge in $E(V^*)$. Since G is an (s, δ)-edge expander and $|V^*| \leq s$, the term T^* contains at least $\delta \cdot |V^*|$ variables. From $Sp(\mathbb{C}) \geq |T^*|$ and the fact that $|V^*| = \mu(\mathbb{C})$ we prove that $Sp(\mathbb{C}) \geq \delta \cdot \mu(\mathbb{C})$ if $\mu(\mathbb{C}) \leq s$. ◀

The preceding lemma and Corollary 13 together imply Theorem 9, because by Corollary 13 there is a configuration with measure between s/d and s, and this configuration has space at least $\delta s/d$ by the previous lemma.

Theorem 9 gives inferior results compared to a direct application of Theorem 1 to known width lower bounds. The bounds that we get are worse by a multiplicative factor of $1/d$. One might hope to remove this multiplicative factor by improving the bound in Lemma 12, but this is not possible because that bound is tight.

To see this, assume that the graph G consists of a set of vertices V with one vertex v that is a neighbor of d disjoint subgraphs each of size $(|V| - 1)/d$. Also, let A be one of the clauses in $PARITY_{v,\chi}$ such that setting any literal in A to true pushes the odd charge into one of the neighboring subgraphs of v. Taking $\mathbb{C} = \emptyset$ and $\mathbb{C}' = \{A\}$ we have that $\mu(\mathbb{C}) = |V|$ and $\mu(\mathbb{C}') = (|V| - 1)/d$. The latter equality holds because every minimal term T satisfying A contains exactly one literal from A, and so pushes the odd charge into one of the subgraphs neighboring v. This makes the vertices of that subgraph a witness for $\nu(T)$. Hence, we have an example where $d \cdot \mu(\mathbb{C}') + 1 = \mu(\mathbb{C})$, which shows that Lemma 12 is tight.

5 From Small Space to Small Degree in Polynomial Calculus?

An intriguing question is whether an analogue of the bound in Theorem 1 holds also for the stronger algebraic proof system *polynomial calculus* introduced in [12]. In this context, it is more relevant to discuss the variant of this system presented in [1], which is known as *polynomial calculus (with) resolution* or *PCR*, which we briefly describe below.

In a PCR derivation, the configurations are sets of polynomials in $\mathbb{F}[x, \overline{x}, y, \overline{y}, \ldots]$, where x and $\overline{x}$ are different formal variables. By way of example, a clause $x \vee y \vee \overline{z}$ is translated to the polynomial $xy\overline{z}$ consisting of one monomial. In addition to the translations of the

axiom clauses of the CNF formula to be refuted, the proof system also contain axioms $x^2 - x$ and $x + \overline{x} - 1$. These axioms enforce that only assigments to $\{0, 1\}$ are considered (and hence that all polynomials are multilinear without loss of generality) and that $\overline{x}$ always takes the opposite value of x. There are two inference rules, which preserve common roots of the polynomials, namely *linear combination* $\frac{p \quad q}{\alpha p + \beta q}$ and *multiplication* $\frac{p}{xp}$, where p and q are (previously derived) polynomials, the coefficients α, β are elements of $\mathbb{F}$, and x is any variable (with or without bar). The refutation ends when 1 has been derived. The *size*, *degree* and *monomial space* measures are analogues of length, width and clause space in resolution (counting monomials instead of clauses). PCR can simulate resolution refutations efficiently with respect to all of these measures.

Let us now discuss why the method we use to prove Theorem 1 is unlikely to generalize to PCR. An example of formulas that seem hard to deal with in this way are so-called *pebbling contradictions*, which we describe next.

Pebbling contradictions are defined in terms of directed acyclic graphs (DAGs) $G = (V, E)$ with bounded fan-in, where vertices with no incoming edges are called *sources* and vertices without outgoing edges *sinks*. Assume G has a unique sink z and associate a variable V to each vertex $v \in V$. Then the pebbling contradiction over G consists of the following clauses:

- for each source vertex s, a clause s (*source axioms*),
- for each non-source vertex v, a clause $\bigvee_{(u,v)\in E} \overline{u} \vee v$ (*pebbling axioms*),
- for the sink z, a clause $\overline{z}$ (*sink axiom*).

As shown in [6], pebbling contradictions exhibit space-width trade-offs in resolution in that they can always be refuted in constant width as well as in constant space, but there are graphs for which optimizing one of these measures necessary causes essentially worst-case linear behaviour for the other measure.

There are two natural ways to refute pebbling contradictions in resolution. One approach is to go "bottom-up" from sources to sinks in topological order, and derive for each vertex $v \in V(G)$ the clause v using the pebbling axiom for v and the clauses for the predecessors of the vertex v. When the refutation reaches z it derives a contradiction with the sink axiom $\overline{z}$. This can be done in constant width but for some graphs requires large space. The other approach is "top-down" starting from the sink axiom $\overline{z}$ and deriving clauses of the form $\overline{v}_1 \vee \cdots \vee \overline{v}_\ell$. A new clause is derived by replacing any vertex v_i in the old one by all its predecessors, i.e., resolving with the pebbling axiom for v_i. Since G is acyclic we can repeat this process until we get to the sources, for which the negated literals can be resolved away using source axioms. This refutation can be carried out in constant clause space, but such a refutation might require large width.

Now, one can observe that the transformation of configurations in our proof of Theorem 1 maps either of two refutations above into the other one, and this is the main reason why our proof does not seem to generalize to PCR. In PCR, we can represent any conjunction of literals $a_1 \wedge \cdots \wedge a_r$ as the binomial $1 - \prod_i \overline{a}_i$. Using this encoding with the bottom-up approach yields a third refutation, which has constant space but possibly large degree. Hence, there are constant space polynomial calculus refutations of pebbling contradictions in both the bottom-up and the top-down direction. This in turn means that if our proof method were to work for PCR, we would need to find constant degree refutations in both directions. For the top-down case it seems unlikely that such a refutation exists.

6 Concluding Remarks

In this work, we present an alternative, completely elementary, proof of the result by Atserias and Dalmau [2] that space is an upper bound on width in resolution. Our construction

gives a syntactic way to convert a small-space resolution refutation into a refutation in small width. We also exhibit a new "black-box" approach for proving space lower bounds that works by defining a progress measure à la Ben-Sasson and Wigderson [10] and showing that when a refutation has made medium progress towards a contradiction it must be using a lot of space. We believe that these techniques shed interesting new light on resolution space complexity, and hope that they will serve to increase our understanding of this notoriously tricky complexity measure.

As an example of a question about resolution space that still remains open, suppose we are given a k-CNF formula that is guaranteed to be refutable in constant space. By [2] it is also refutable in constant width, and a simple counting argument then shows that exhaustive search in small width will find a polynomial-length resolution refutation. But is there any way of obtaining such a short refutation from a refutation in small space that is more explicit that doing exhaustive search? And can we obtain a short refutation without blowing up the space by more than, say, a constant factor?

Known length-space trade-off results for resolution in [4, 5, 9, 17] do not answer this question as they do not apply to this range of parameters. Unfortunately, our new proof of the space-width inequality cannot be used to resolve this question either, since in the worst case the resolution refutation we obtain might be as bad as the one found by exhaustive search of small-width refutations (or even worse, due to repetition of clauses). This would seem to be inherent—a recent result [3] shows that there are formulas refutable in space and width s where the shortest refutation has length $n^{\Omega(s)}$, i.e., matching the exhaustive search upper bound up to a (small) constant factor in the exponent.

An even more intriguing question is how the space and degree measures are related in polynomial calculus, as discussed in Section 5. For most relations between length, space, and width in resolution, it turns out that they carry over with little or no modification to size, space, and degree, respectively, in polynomial calculus. So can it be that it also holds that space yields upper bounds on degree in polynomial calculus? Or could perhaps even the stronger claim hold that polynomial calculus space is an upper bound on resolution width? These questions remain wide open, but in the recent paper [14] we made some limited progress by showing that if a formula requires large resolution width, then the "XORified version" of the formula requires large polynomial calculus space. We refer to the introductory section of [14] for a more detailed discussion of these issues.

Acknowledgments. The authors wish to thank Albert Atserias, Ilario Bonacina, Nicola Galesi, and Li-Yang Tan for stimulating discussions on topics related to this work.

The research of the first author has received funding from the European Union's Seventh Framework Programme (FP7/2007–2013) under grant agreement no. 238381. Part of the work of the first author was performed while at the University of Toronto and while visiting KTH Royal Institute of Technology. The other authors were funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 279611. The fourth author was also supported by Swedish Research Council grants 621-2010-4797 and 621-2012-5645.

References

1 Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal on Computing*, 31(4):1184–1211, 2002. Preliminary version appeared in *STOC '00*.

2 Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, May 2008. Preliminary version appeared in *CCC '03*.

3 Albert Atserias, Massimo Lauria, and Jakob Nordström. Narrow proofs may be maximally long. In *Proceedings of the 29th Annual IEEE Conference on Computational Complexity (CCC'14)*, to appear, 2014.

4 Paul Beame, Chris Beck, and Russell Impagliazzo. Time-space tradeoffs in resolution: Superpolynomial lower bounds for superlinear space. In *Proc. of the 44th Annual ACM Symp. on Theory of Computing (STOC '12)*, pages 213–232, May 2012.

5 Chris Beck, Jakob Nordström, and Bangsheng Tang. Some trade-off results for polynomial calculus. In *Proc. of the 45th Annual ACM Symp. on Theory of Computing (STOC '13)*, pages 813–822, May 2013.

6 Eli Ben-Sasson. Size space tradeoffs for resolution. *SIAM Journal on Computing*, 38(6):2511–2525, May 2009. Preliminary version appeared in *STOC '02*.

7 Eli Ben-Sasson and Nicola Galesi. Space complexity of random formulae in resolution. *Random Structures and Algorithms*, 23(1):92–109, August 2003. Preliminary version appeared in *CCC '01*.

8 Eli Ben-Sasson and Jakob Nordström. Short proofs may be spacious: An optimal separation of space and length in resolution. In *Proc. of the 49th Annual IEEE Symp. on Foundations of Computer Science (FOCS '08)*, pages 709–718, October 2008.

9 Eli Ben-Sasson and Jakob Nordström. Understanding space in proof complexity: Separations and trade-offs via substitutions. In *Proc. of the 2nd Symp. on Innovations in Computer Science (ICS '11)*, pages 401–416, January 2011. Full-length version available at `http://eccc.hpi-web.de/report/2010/125/`.

10 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, March 2001. Preliminary version appeared in *STOC '99*.

11 Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, October 1988.

12 Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proc. of the 28th Annual ACM Symp. on Theory of Computing (STOC '96)*, pages 174–183, May 1996.

13 Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171(1):84–97, 2001. Preliminary versions of these results appeared in *STACS '99* and *CSL '99*.

14 Yuval Filmus, Massimo Lauria, Mladen Mikša, Jakob Nordström, and Marc Vinyals. Towards an understanding of polynomial calculus: New separations and lower bounds (extended abstract). In *Proc. of the 40th Int'l Colloquium on Automata, Languages and Programming (ICALP '13)*, volume 7965 of *LNCS*, pages 437–448. Springer, July 2013.

15 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2–3):297–308, August 1985.

16 Jan Krajíček. On the weak pigeonhole principle. *Fundamenta Mathematicae*, 170(1–3):123–140, 2001.

17 Jakob Nordström. A simplified way of proving trade-off results for resolution. *Information Processing Letters*, 109(18):1030–1035, August 2009. Preliminary version appeared in ECCC report TR07-114, 2007.

18 Jakob Nordström. Pebble games, proof complexity and time-space trade-offs. *Logical Methods in Computer Science*, 9:15:1–15:63, September 2013.

19 Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.

Explicit Linear Kernels via Dynamic Programming*

Valentin Garnero[1], Christophe Paul[1], Ignasi Sau[1], and Dimitrios M. Thilikos[1,2]

1 **AlGCo project-team, CNRS and Université de Montpellier 2, LIRMM, Montpellier, France**
`FirstName.FamilyName@lirmm.fr`

2 **Department of Mathematics, National and Kapodistrian University of Athens, Athens, Greece**

Abstract

Several algorithmic meta-theorems on kernelization have appeared in the last years, starting with the result of Bodlaender *et al.* [FOCS 2009] on graphs of bounded genus, then generalized by Fomin *et al.* [SODA 2010] to graphs excluding a fixed minor, and by Kim *et al.* [ICALP 2013] to graphs excluding a fixed topological minor. Typically, these results guarantee the existence of linear or polynomial kernels on sparse graph classes for problems satisfying some generic conditions but, mainly due to their generality, it is not clear how to derive from them constructive kernels with explicit constants.

In this paper we make a step toward a fully constructive meta-kernelization theory on sparse graphs. Our approach is based on a more explicit protrusion replacement machinery that, instead of expressibility in CMSO logic, uses dynamic programming, which allows us to find an explicit upper bound on the size of the derived kernels. We demonstrate the usefulness of our techniques by providing the first explicit linear kernels for r-Dominating Set and r-Scattered Set on apex-minor-free graphs, and for Planar-$\mathcal{F}$-Deletion on graphs excluding a fixed (topological) minor in the case where all the graphs in $\mathcal{F}$ are connected.

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

Keywords and phrases parameterized complexity, linear kernels, dynamic programming, protrusion replacement, graph minors

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.312

1 Introduction

Motivation. Parameterized complexity deals with problems whose instances I come equipped with an additional integer parameter k, and the objective is to obtain algorithms whose running time is of the form $f(k) \cdot \text{poly}(|I|)$, where f is some computable function (see [6, 7] for an introduction to the field). We will be only concerned with problems defined on graphs. A fundamental notion in parameterized complexity is that of *kernelization*, which asks for the existence of polynomial-time preprocessing algorithms that produce equivalent instances whose size depends exclusively (preferably polynomially or event linearly) on k. Finding

* This work was supported by the ANR project AGAPE (ANR-09-BLAN-0159) and the Languedoc-Roussillon Project "Chercheur d'avenir" KERNEL. The fourth author was co-financed by the E.U. (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: "Thales. Investing in knowledge society through the European Social Fund".



© Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 312–324



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



kernels of size polynomial or linear in k (called *linear kernels*) is one of the major goals of this area.

An influential work in this direction was the linear kernel of Alber *et al.* [2] for Dominating Set on planar graphs, which was generalized by Guo and Niedermeier [11] to a family of problems on planar graphs. Several algorithmic meta-theorems on kernelization have appeared in the last years, starting with the result of Bodlaender *et al.* [3] on graphs of bounded genus. After that, similar results have been obtained on larger sparse graph classes, such as graphs excluding a minor [9] or a topological minor [14].

Typically, the above results guarantee the *existence* of linear or polynomial kernels on sparse graph classes for a number of problems satisfying some generic conditions but, mainly due to their generality, it is hard to derive from them *constructive* kernels with *explicit* constants. The main reason behind this non-constructibility is that the proofs rely on a property of problems called *Finite Integer Index* (FII) that, roughly speaking, allows to replace large "protrusions" (i.e., large subgraphs with small boundary to the rest of the graph) with "equivalent" subgraphs of constant size. This substitution procedure is known as *protrusion replacer*, and while its *existence* has been proved, so far, there is no generic way to *construct* it. Using the technology developed in [3], there are cases where protrusion replacements can become constructive given the expressibility of the problem in Counting Monadic Second Order (CMSO) logic. This approach is essentially based on extensions of Courcelle's theorem [4] that, even when they offer constructibility, it is hard to extract from them any *explicit constant* that upper-bounds the size of the derived kernel.

Results and techniques. In this article we tackle the above issues and make a step toward a fully constructive meta-kernelization theory on sparse graphs with explicit constants. For this, we essentially substitute the algorithmic power of CMSO logic with that of dynamic programming on graphs of bounded decomposability (i.e., bounded treewidth). Our approach provides a dynamic programming framework able to construct a protrusion replacer for a wide variety of problems.

Loosely speaking, the framework that we present can be summarized as follows. First of all, we propose a general definition of a problem encoding for the tables of dynamic programming when solving parameterized problems on graphs of bounded treewidth. Under this setting, we provide general conditions on whether such an encoding can yield a protrusion replacer. While our framework can also be seen as a possible formalization of dynamic programming, our purpose is to use it for constructing protrusion replacement algorithms and linear kernels whose size is explicitly determined.

In order to obtain an explicit linear kernel for a problem Π, the main ingredient is to prove that when solving Π on graphs of bounded treewidth via dynamic programming, we can use tables such that the maximum difference between all the values that need to be stored is bounded by a function of the treewidth. For this, we prove in Theorem 13 that when the input graph excludes a fixed graph H as a (topological) minor, this condition is sufficient for constructing an explicit protrusion replacer algorithm, i.e., a polynomial-time algorithm that replaces a large protrusion with an equivalent one whose size can be bounded by an *explicit* constant. Such a protrusion replacer can then be used, for instance, whenever it is possible to compute a linear protrusion decomposition of the input graph (that is, an algorithm that partitions the graph into a part of size linear in $O(k)$ and a set of $O(k)$ protrusions). As there is a wealth of results for constructing such decompositions [3, 8, 9, 14], we can use them as a starting point and, by applying dynamic programming, obtain an explicit linear kernel for Π.

We demonstrate the usefulness of this general strategy by providing the first explicit linear kernels for three distinct families of problems on sparse graph classes. On the one hand, for each integer $r \geqslant 1$, we provide a linear kernel for r-Dominating Set and r-Scattered Set on graphs excluding a fixed apex graph H as a minor. Moreover, for each finite family $\mathcal{F}$ of connected graphs containing at least one planar graph, we provide a linear kernel for Planar-$\mathcal{F}$-Deletion on graphs excluding a fixed graph H as a (topological) minor[1].

We chose these families of problems as they are all tuned by a secondary parameter that is either the constant r or the size of the graphs in the family $\mathcal{F}$. That way, we not only capture a wealth of parameterized problems, but we also make explicit the contribution of the secondary parameter in the size of the derived kernels.

Organization of the paper. Due to space limitations, the proofs of the results marked with '[⋆]' can be found in the full version of this paper, which is permanently available at [10]. For the reader not familiar with the background used in previous work on this topic [3, 9, 14], some preliminaries can be found in [10], including graph minors, parameterized problems, (rooted) tree-decompositions, boundaried graphs, the canonical equivalence relation $\equiv_{\Pi,t}$ for a problem Π and an integer t, FII, protrusions, and protrusion decompositions. In Section 2 we introduce the basic definitions of our framework and present an explicit protrusion replacer. In Section 3 we show how to apply our methodology to various problems, and we conclude with some directions for further research in Section 4.

2 An explicit protrusion replacer

In this section we present our strategy to construct an explicit protrusion replacer via dynamic programming. For a positive integer t, we define $\mathcal{F}_t$ as the class of all t-boundaried graphs of treewidth at most $t-1$ that have a rooted tree-decomposition with all boundary vertices contained in the root-bag. We will restrict ourselves to parameterized graph problems such that a solution can be certified by a subset of vertices.

▶ **Definition 1** (Vertex-certifiable problem)**.** A parameterized graph problem Π is called *vertex-certifiable* if there exists a language L_Π (called *certifying language for* Π) defined on pairs (G, S), where G is a graph and $S \subseteq V(G)$, such that (G, k) is a Yes-instance of Π if and only if there exists a subset $S \subseteq V(G)$ with $|S| \leqslant k$ (or $|S| \geqslant k$, depending on the problem) such that $(G, S) \in L_\Pi$.

Many graph problems are vertex-certifiable, like r-Dominating Set, Feedback Vertex Set, or Treewidth-t Vertex Deletion. This section is structured as follows. In Subsection 2.1 we define the notion of encoder, the main object that will allow us to formalize in an abstract way the tables of dynamic programming. In Subsection 2.2 we use encoders to define an equivalence relation on graphs in $\mathcal{F}_t$ that, under some natural technical conditions, will be a *refinement* of the canonical equivalence relation defined by a problem Π (see [10]). This refined equivalence relation allows us to provide an explicit upper bound on the size of its representatives (Lemma 11), as well as a linear-time algorithm to find them (Lemma 12). In Subsection 2.3 we use the previous ingredients to present an explicit

[1] In an earlier version of this paper, we also described a linear kernel for Planar-$\mathcal{F}$-Packing on graphs excluding a fixed graph H as a minor. Nevertheless, as this problem is not directly vertex-certifiable (see Definition 1), for presenting it we should restate and extend many of the definitions and results given in Section 2 in order to deal with more general families of problems. Therefore, we decided not to include this family of problems in this article.

protrusion replacement rule (Theorem 13), which replaces a large enough protrusion with a bounded-size representative from its equivalence class, in such a way that the parameter does not increase.

2.1 Encoders

The Dominating Set problem, as a vertex-certifiable problem, will be used hereafter as a running example to illustrate our general framework and definitions. Let us start with a description of dynamic programming tables for Dominating Set on graphs of bounded treewidth.

Running example: Let B be a bag of a rooted tree-decomposition $(T, \mathcal{X})$ of width $t-1$ of a graph $G \in \mathcal{F}_t$. The dynamic programming (DP) tables for Dominating Set can be defined as follows. The entries of the DP-table for B are indexed by the set of tuples $R \in \{0, \uparrow 1, \downarrow 1\}^{|B|}$, so-called *encodings*. As detailed below, the symbol 0 stands for vertices in the (partial) dominating set, the symbol $\downarrow 1$ for vertices that are already dominated, and $\uparrow 1$ for vertices with no constraints. More precisely, the coordinates of each $|B|$-tuple are in one-to-one correspondence with the vertices of B. For a vertex $v \in B$, we denote by $R(v)$ its corresponding coordinate in the encoding R. A subset $S \subseteq V(G_B)$ is a *partial dominating set satisfying* R if the following conditions are satisfied:

- $\forall v \in V(G_B) \setminus B$, $d_{G_B}(v, S) \leqslant 1$; and
- $\forall v \in B$: $R(v) = 0 \Rightarrow v \in S$, and $R(v) = \downarrow 1 \Rightarrow d_{G_B}(v, S) \leqslant 1$.

Observe that if S is a partial dominating set satisfying R, then $\{v \in B \mid R(v) = 0\} \subseteq S$, but S may also contain vertices with $R(v) \neq 0$. Likewise, the vertices that are not (yet) dominated by S are contained in the set $\{v \in B \mid R(v) = \uparrow 1\}$. ◀

The following definition considers the tables of dynamic programming in an abstract way.

▶ **Definition 2** (Encoder). An *encoder* $\mathcal{E}$ is a pair $(\mathcal{C}, L_{\mathcal{C}})$ where

(i) $\mathcal{C}$ is a function that, for each (possibly empty) finite subset $I \subseteq \mathbb{N}^+$, outputs a (possibly empty) finite set $\mathcal{C}(I)$ of strings over some alphabet. Each $R \in \mathcal{C}(I)$ is called a *$\mathcal{C}$-encoding* of I; and

(ii) $L_{\mathcal{C}}$ is a computable language whose strings encode triples (G, S, R), where G is a boundaried graph, $S \subseteq V(G)$, and $R \in \mathcal{C}(\Lambda(G))$. If $(G, S, R) \in L_{\mathcal{C}}$, we say that S *satisfies* the $\mathcal{C}$-encoding R.

As it will become clear with the running example, the set I represents the labels from a bag, $\mathcal{C}(I)$ represents the possible configurations of the vertices in the bag, and $L_{\mathcal{C}}$ contains triples that correspond to solutions to these configurations.

Running example: Each rooted graph G_B can be naturally viewed as a $|B|$-boundaried graph such that $B = \partial(G_B)$ with $I = \Lambda(G_B)$. Let $\mathcal{E}_{\mathrm{DS}} = (\mathcal{C}_{\mathrm{DS}}, L_{\mathcal{C}_{\mathrm{DS}}})$ be the encoder described above for Dominating Set. The tables of the dynamic programming algorithm to solve Dominating Set are obtained by assigning to every $\mathcal{C}_{\mathrm{DS}}$-encoding (that is, DP-table entry) $R \in \mathcal{C}_{\mathrm{DS}}(I)$, the size of a minimum partial dominating set satisfying R, or $+\infty$ if such a set of vertices does not exist. This defines a function $f_G^{\mathcal{E}_{\mathrm{DS}}} : \mathcal{C}_{\mathrm{DS}}(I) \to \mathbb{N} \cup \{+\infty\}$. Observe that if $B = \partial(G_B) = \emptyset$, then the value assigned to the encodings in $\mathcal{C}_{\mathrm{DS}}(\emptyset)$ is indeed the size of a minimum dominating set of G_B. ◇

For a general minimization problem Π, we will only be interested in encoders that permit to solve Π via dynamic programming. More formally, we define a Π-encoder and the values assigned to the encodings as follows. (Maximization problems are treated similarly, see [10]

for the corresponding definitions of the functions $f_G^{\mathcal{E}}$ and $f_G^{\mathcal{E},g}$ defined below. The other definitions of this section remain unchanged.)

▶ **Definition 3** (Π-encoder and its associated function)**.** Let Π be a vertex-certifiable minimization problem.

(i) An encoder $\mathcal{E} = (\mathcal{C}, L_{\mathcal{C}})$ is a Π-*encoder* if $\mathcal{C}(\emptyset)$ consists of a single $\mathcal{C}$-encoding, namely $R_\emptyset$, such that for every 0-boundaried graph G and every $S \subseteq V(G)$, $(G, S, R_\emptyset) \in L_{\mathcal{C}}$ if and only if $(G, S) \in L_\Pi$.

(ii) Let G be a t-boundaried graph with $\Lambda(G) = I$. We define the function $f_G^{\mathcal{E}} : \mathcal{C}(I) \to \mathbb{N} \cup \{+\infty\}$ as

$$f_G^{\mathcal{E}}(R) \;=\; \min\{k \;:\; \exists S \subseteq V(G), |S| \leqslant k, (G, S, R) \in L_{\mathcal{C}}\}. \tag{1}$$

In Equation (1), if such a set S does not exist, we set $f_G^{\mathcal{E}}(R) := +\infty$. We define $\mathcal{C}_G^*(I) := \{R \in \mathcal{C}(I) \mid f_G^{\mathcal{E}}(R) \neq +\infty\}$.

Condition (i) in Definition 3 guarantees that, when the considered graph G has no boundary, the language of the encoder is able to *certify* a solution of problem Π. In other words, we ask that the set $\{(G, S) \mid (G, S, R_\emptyset) \in L_{\mathcal{C}})\}$ is a *certifying language* for Π. Observe that for a 0-boundaried graph G, the function $f_G^{\mathcal{E}}(R_\emptyset)$ outputs the minimum size of a set S such that $(G, S) \in L_\Pi$.

The following definition provides a way to control the number of possible distinct values assigned to encodings. This property will play a similar role to FII or *monotonicity* in previous work [3, 9, 14].

▶ **Definition 4** (Confined encoding)**.** An encoder $\mathcal{E}$ is *g-confined* if there exists a function $g : \mathbb{N} \to \mathbb{N}$ such that for any t-boundaried graph G with $\Lambda(G) = I$ it holds that either $\mathcal{C}_G^*(I) = \emptyset$ or

$$\max_{R \in \mathcal{C}_G^*(I)} f_G^{\mathcal{E}}(R) \;-\; \min_{R \in \mathcal{C}_G^*(I)} f_G^{\mathcal{E}}(R) \;\leqslant\; g(t). \tag{2}$$

See the figure in [10] for a schematic illustration of a confined encoder. In this figure, each column of the table corresponds to a $\mathcal{C}$-encoder R, which is filled with the value $f_G^{\mathcal{E}}(R)$. *Running example:* It is easy to observe that the encoder $\mathcal{E}_{\text{DS}}$ described above is g-confined for $g(t) = t$. Indeed, let G be a t-boundaried graph (corresponding to the graph G_B considered before) with $\Lambda(G) = I$. Consider an arbitrary encoding $R \in \mathcal{C}(I)$ and the encoding $R_0 \in \mathcal{C}(I)$ satisfying $R_0(v) = 0$ for every $v \in \partial(G)$. Let $S_0 \subseteq V(G)$ be a minimum-sized partial dominating set satisfying R_0, i.e., such that $(G, S_0, R_0) \in L_{\mathcal{C}_{\text{DS}}}$. Observe that S_0 also satisfies R, i.e., $(G, S_0, R) \in L_{\mathcal{C}_{\text{DS}}}$. It then follows that $f_G^{\mathcal{E}_{\text{DS}}}(R_0) = \max_R f_G^{\mathcal{E}_{\text{DS}}}(R)$. Moreover, let $S \subseteq V(G)$ be a minimum-sized partial dominating set satisfying R, i.e., such that $(G, S, R) \in L_{\mathcal{C}_{DS}}$. Then note that R_0 is satisfied by the set $S \cup \partial(G)$, so we have that for every encoding R, $f_G^{\mathcal{E}_{\text{DS}}}(R) + |\partial(G)| \geqslant f_G^{\mathcal{E}_{\text{DS}}}(R_0)$. It follows that $f_G^{\mathcal{E}_{\text{DS}}}(R_0) - \min_R f_G^{\mathcal{E}_{\text{DS}}}(R) \;\leqslant\; |\partial(G)| \leqslant t$, proving that the encoder is indeed g-confined. ◇

For some problems and encoders, we may need to "force" the confinement of an encoder $\mathcal{E}$ that may not be confined according to Definition 4, while still preserving its usefulness for dynamic programming, in the sense that no relevant information is removed from the tables (for example, see the encoder for r-Scattered Set in [10]). To this end, given a function $g : \mathbb{N} \to \mathbb{N}$, we define the function $f_G^{\mathcal{E},g} : \mathcal{C}(I) \to \mathbb{N} \cup \{+\infty\}$ as

$$f_G^{\mathcal{E},g}(R) \;=\; \begin{cases} +\infty, & \text{if } f_G^{\mathcal{E}}(R) - g(t) > \min_{R \in \mathcal{C}(I)} f_G^{\mathcal{E}}(R) \\ f_G^{\mathcal{E}}(R), & \text{otherwise.} \end{cases} . \tag{3}$$

Intuitively, one shall think as the function $f_G^{\mathcal{E},g}$ as a "compressed" version of the function $f_G^{\mathcal{E}}$, which stores only the values that are useful for performing dynamic programming.

2.2 Equivalence relations and representatives

An encoder $\mathcal{E}$ together with a function $g : \mathbb{N} \to \mathbb{N}$ define an equivalence relation $\sim_{\mathcal{E},g,t}$ on graphs in $\mathcal{F}_t$ as follows.

▶ **Definition 5** (Equivalence relation $\sim_{\mathcal{E},g,t}$)**.** Let $\mathcal{E}$ be an encoder, let $g : \mathbb{N} \to \mathbb{N}$, and let $G_1, G_2 \in \mathcal{F}_t$. We say that $G_1 \sim_{\mathcal{E},g,t} G_2$ if and only if $\Lambda(G_1) = \Lambda(G_2) =: I$ and there exists an integer c, depending only on G_1 and G_2, such that for every $\mathcal{C}$-encoding $R \in \mathcal{C}(I)$ it holds that

$$f_{G_1}^{\mathcal{E},g}(R) \;=\; f_{G_2}^{\mathcal{E},g}(R) + c\,. \tag{4}$$

Note that if there exists $R \in \mathcal{C}(I)$ such that $f_{G_1}^{\mathcal{E},g}(R) \neq \infty$, then the integer c satisfying Equation (4) is unique, otherwise every integer c satisfies Equation (4). We define the following function $\Delta_{\mathcal{E},g,t} : \mathcal{F}_t \times \mathcal{F}_t \to \mathbb{Z}$, which is called, following the terminology from Bodlaender *et al.* [3], the *transposition function* for the equivalence relation $\sim_{\mathcal{E},g,t}$.

$$\Delta_{\mathcal{E},g,t}(G_1, G_2) \;=\; \begin{cases} c, & \text{if } G_1 \sim_{\mathcal{E},g,t} G_2 \text{ and Eq. (4) holds for a unique integer } c; \\ 0, & \text{if } G_1 \sim_{\mathcal{E},g,t} G_2 \text{ and Eq. (4) holds for every integer; and} \\ & \text{undefined otherwise.} \end{cases} \tag{5}$$

If we are dealing with a problem defined on a graph class $\mathcal{G}$, the protrusion replacement rule has to preserve the class $\mathcal{G}$, as otherwise we would obtain a *bikernel* instead of a kernel. That is, we need to make sure that, when replacing a graph in $\mathcal{F}_t \cap \mathcal{G}$ with one of its representatives, we do not produce a graph that does not belong to $\mathcal{G}$ anymore. To this end, we define an equivalence relation $\sim_{\mathcal{E},g,t,\mathcal{G}}$ on graphs in $\mathcal{F}_t \cap \mathcal{G}$, which refines the equivalence relation $\sim_{\mathcal{E},g,t}$ of Definition 5.

▶ **Definition 6** (Equivalence relation $\sim_{\mathcal{E},g,t,\mathcal{G}}$)**.** Let $\mathcal{G}$ be a class of graphs and let $G_1, G_2 \in F_t \cap \mathcal{G}$.

(i) $G_1 \sim_{\mathcal{G},t} G_2$ if and only if for any t-boundaried graph H, $G_1 \oplus H \in \mathcal{G}$ if and only if $G_2 \oplus H \in \mathcal{G}$.

(ii)] $G_1 \sim_{\mathcal{E},g,t,\mathcal{G}} G_2$ if and only if $G_1 \sim_{\mathcal{E},g,t} G_2$ and $G_1 \sim_{\mathcal{G},t} G_2$.

It is well-known by Büchi's theorem that regular languages are precisely those definable in Monadic Second Order logic (MSO logic). By Myhill-Nerode's theorem, it follows that if the membership in a graph class $\mathcal{G}$ can be expressed in MSO logic, then the equivalence relation $\sim_{\mathcal{G},t}$ has a finite number of equivalence classes (see for instance [6, 7]). However, we do not have in general an explicit upper bound on the number of equivalence classes of $\sim_{\mathcal{G},t}$, henceforth denoted by $r_{\mathcal{G},t}$. Fortunately, in the context of our applications in Section 3, where $\mathcal{G}$ will be a class of graphs that exclude some fixed graph as a (topological) minor[2], this will always be possible, and in this case it holds that $r_{\mathcal{G},t} \leqslant 2^{t \log t} \cdot h^t \cdot 2^{h^2}$.

For an encoder $\mathcal{E} = (\mathcal{C}, L_{\mathcal{C}})$, we let $s_{\mathcal{E}}(t) := \max_{I \subseteq \{1,\ldots,t\}} |\mathcal{C}(I)|$, where $|\mathcal{C}(I)|$ denotes the number of $\mathcal{C}$-encodings in $\mathcal{C}(I)$. The following lemma gives an upper bound on the number of equivalence classes of $\sim_{\mathcal{E},g,t,\mathcal{G}}$, which depends also on $r_{\mathcal{G},t}$.

[2] A particular case of the classes of graphs whose membership can be expressed in MSO logic. We would like to stress here that we rely on the expressibility of the *graph class* $\mathcal{G}$ in MSO logic, whereas in previous work [3, 9, 14] what is used in the expressibility in CMSO logic of the *problems* defined on a graph class.

▶ **Lemma 7.** *Let $\mathcal{G}$ be a graph class whose membership can be expressed in MSO logic. For any encoder $\mathcal{E}$, any function $g : \mathbb{N} \to \mathbb{N}$, and any positive integer t, the equivalence relation $\sim_{\mathcal{E},g,t,\mathcal{G}}$ has finite index. More precisely, the number of equivalence classes of $\sim_{\mathcal{E},g,t,\mathcal{G}}$ is at most $r(\mathcal{E}, g, t, \mathcal{G}) := (g(t) + 2)^{s\varepsilon(t)} \cdot 2^t \cdot r_{\mathcal{G},t}$.*

Proof. Let us first show that the equivalence relation $\sim_{\mathcal{E},g,t}$ has finite index. Indeed, let $I \subseteq \{1, \ldots, t\}$. By definition, we have that for any graph $G \in \mathcal{F}_t$ with $\Lambda(G) = I$, the function $f_G^{\mathcal{E},g}$ can take at most $g(t) + 2$ distinct values ($g(t) + 1$ finite values and possibly the value $+\infty$). Therefore, it follows that the number of equivalence classes of $\sim_{\mathcal{E},g,t}$ containing all graphs G in $\mathcal{F}_t$ with $\Lambda(G) = I$ is at most $(g(t) + 2)^{|\mathcal{C}(I)|}$. As the number of subsets of $\{1, \ldots, t\}$ is 2^t, we deduce that the overall number of equivalence classes of $\sim_{\mathcal{E},g,t}$ is at most $(g(t) + 2)^{s\varepsilon(t)} \cdot 2^t$. Finally, since the equivalence relation $\sim_{\mathcal{E},t,\mathcal{G}}$ is the Cartesian product of the equivalence relations $\sim_{\mathcal{E},g,t}$ and $\sim_{\mathcal{G},t}$, the result follows from the fact that $\mathcal{G}$ can be expressed in MSO logic. ◀

In order for an encoding $\mathcal{E}$ and a function g to be useful for performing dynamic programming on graphs in $\mathcal{F}_t$ that belong to a graph class $\mathcal{G}$, we introduce the following definition, which captures the natural fact that the tables of a dynamic programming algorithm should depend exclusively on the tables of the descendants in a rooted tree-decomposition. Before moving to the definition, we note that given a graph $G \in \mathcal{F}_t$ and a rooted tree-decomposition $(T, \mathcal{X})$ of G such that $\partial(G)$ is contained in the root-bag of $(T, \mathcal{X})$, the labels of $\partial(G)$ can be propagated in a natural way to all bags of $(T, \mathcal{X})$ by introducing, removing, and shifting labels appropriately. Therefore, for any node x of T, the graph G_x can be naturally seen as a graph in $\mathcal{F}_t$. (A brief discussion can be found in [10], and we refer to [3] for more details.)

▶ **Definition 8** (DP-friendly equivalence relation)**.** An equivalence relation $\sim_{\mathcal{E},g,t,\mathcal{G}}$ is *DP-friendly* if for any graph $G \in \mathcal{F}_t$ and any rooted tree-decomposition $(T, \mathcal{X})$ of G such that $\partial(G)$ is contained in the root-bag of $(T, \mathcal{X})$, and for any descendant x of the root r of T, if G' is the graph obtained from G by replacing the graph $G_x \in \mathcal{F}_t$ with a graph $G'_x \in \mathcal{F}_t$ such that $G_x \sim_{\mathcal{E},g,t,\mathcal{G}} G'_x$, then G' satisfies the following conditions:

(i) $G \sim_{\mathcal{E},g,t,\mathcal{G}} G'$; and

(ii) $\Delta_{\mathcal{E},g,t}(G, G') = \Delta_{\mathcal{E},g,t}(G_x, G'_x)$.

In Definition 8, as well as in the remainder of the article, when we *replace* the graph G_x with the graph G'_x, we do *not* remove from G any of the edges with both endvertices in the boundary of G_x. That is, $G' = (G - (V(G_x) - \partial(V(G_x)))) \oplus G'_x$.

Recall that for the protrusion replacement to be valid for a problem Π, the equivalence relation $\sim_{\mathcal{E},g,t,\mathcal{G}}$ needs to be a refinement of the canonical equivalence relation $\equiv_{\Pi,t}$ (note that this implies, in particular, that if $\sim_{\mathcal{E},g,t,\mathcal{G}}$ has finite index, then Π has FII). The next lemma states a sufficient condition for this property, and furthermore it gives the value of the transposition constant $\Delta_{\Pi,t}(G_1, G_2)$, which will be needed in order to update the parameter after the replacement.

▶ **Lemma 9.** [$\star$] *Let Π be a vertex-certifiable problem. If $\mathcal{E}$ is a Π-encoder and $\sim_{\mathcal{E},g,t,\mathcal{G}}$ is a DP-friendly equivalence relation, then for any two graphs $G_1, G_2 \in \mathcal{F}_t$ such that $G_1 \sim_{\mathcal{E},g,t,\mathcal{G}} G_2$, it holds that $G_1 \equiv_{\Pi,t} G_2$ and $\Delta_{\Pi,t}(G_1, G_2) = \Delta_{\mathcal{E},g,t}(G_1, G_2)$.*

The following definition will be important to guarantee that, when applying our protrusion replacement rule, the parameter of the problem under consideration does not increase.

▶ **Definition 10** (Progressive representatives of $\sim_{\mathcal{E},g,t,\mathcal{G}}$)**.** Let $\mathfrak{C}$ be some equivalence class of $\sim_{\mathcal{E},g,t,\mathcal{G}}$ and let $G \in \mathfrak{C}$. We say that G is a *progressive representative* of $\mathfrak{C}$ if for any graph $G' \in \mathfrak{C}$ it holds that $\Delta_{\mathcal{E},g,t}(G, G') \leqslant 0$.

In the next lemma we provide an upper bound on the size of a smallest *progressive* representative of any equivalence class of $\sim_{\mathcal{E},g,t,\mathcal{G}}$.

▶ **Lemma 11.** [⋆] *Let $\mathcal{G}$ be a graph class whose membership can be expressed in MSO logic. For any encoder $\mathcal{E}$, any function $g : \mathbb{N} \to \mathbb{N}$, and any $t \in \mathbb{N}$ such that $\sim_{\mathcal{E},g,t,\mathcal{G}}$ is DP-friendly, every equivalence class of $\sim_{\mathcal{E},g,t,\mathcal{G}}$ has a progressive representative of size at most $b(\mathcal{E}, g, t, \mathcal{G}) := 2^{r(\mathcal{E},g,t,\mathcal{G})+1} \cdot t$, where $r(\mathcal{E}, g, t, \mathcal{G})$ is the function defined in Lemma 7.*

The next lemma states that if one is given an upper bound on the size of the progressive representatives of an equivalence relation defined on t-protrusions (that is, on graphs in $\mathcal{F}_t$)[3], then a *small* progressive representative of a t-protrusion can be explicitly calculated in linear time. In other words, it provides a generic and constructive way to perform a dynamic programming procedure to replace protrusions, without needing to deal with the particularities of each encoder in order to compute the tables. Its proof uses some ideas taken from [3, 9].

▶ **Lemma 12.** [⋆] *Let $\mathcal{G}$ be a graph class, let $\mathcal{E}$ be an encoder, let $g : \mathbb{N} \to \mathbb{N}$, and let $t \in \mathbb{N}$ such that $\sim_{\mathcal{E},g,t,\mathcal{G}}$ is DP-friendly. Assume that we are given an upper bound b on the size of a smallest progressive representative of any equivalence class of $\sim_{\mathcal{E},g,t,\mathcal{G}}$. Then, given an n-vertex t-protrusion G, we can output in time $O(n)$ a t-protrusion H of size at most b such that $G \sim_{\mathcal{E},g,t,\mathcal{G}} H$ and the corresponding transposition constant $\Delta_{\mathcal{E},g,t}(H, G)$ with $\Delta_{\mathcal{E},g,t}(H, G) \leqslant 0$, where the constant in the "O" notation depends only on $\mathcal{E}, g, b, \mathcal{G}$, and t.*

2.3 Explicit protrusion replacer

We are now ready to piece everything together and state our main technical result, which can be interpreted as a generic *constructive* way of performing protrusion replacement with *explicit* size bounds. For our algorithms to be fully constructive, we restrict $\mathcal{G}$ to be the class of graphs that exclude some fixed graph H as a (topological) minor.

▶ **Theorem 13.** *Let H be a fixed graph and let $\mathcal{G}$ be the class of graphs that exclude H as a (topological) minor. Let Π be a vertex-certifiable parameterized graph problem defined on $\mathcal{G}$, and suppose that we are given a Π-encoder $\mathcal{E}$, a function $g : \mathbb{N} \to \mathbb{N}$, and an integer $t \in \mathbb{N}$ such that $\sim_{\mathcal{E},g,t,\mathcal{G}}$ is DP-friendly. Then, given an input graph (G, k) and a t-protrusion Y in G, we can compute in time $O(|Y|)$ an equivalent instance $((G - (Y - \partial(Y))) \oplus Y', k')$, where $k' \leqslant k$ and Y' is a t-protrusion with $|Y'| \leqslant b(\mathcal{E}, g, t, \mathcal{G})$, where $b(\mathcal{E}, g, t, \mathcal{G})$ is the function defined in Lemma 11.*

Proof. By Lemma 7, the number of equivalence classes of the equivalence relation $\sim_{\mathcal{E},g,t,\mathcal{G}}$ is finite, and by Lemma 11 the size of a smallest progressive representative of any equivalence class of $\sim_{\mathcal{E},g,t,\mathcal{G}}$ is at most $b(\mathcal{E}, g, t, \mathcal{G})$. Therefore, we can apply Lemma 12 and deduce that, in time $O(|Y|)$, we can find a t-protrusion Y' of size at most $b(\mathcal{E}, g, t, \mathcal{G})$ such that $Y \sim_{\mathcal{E},g,t,\mathcal{G}} Y'$, and the corresponding transposition constant $\Delta_{\mathcal{E},g,t}(Y', Y)$ with $\Delta_{\mathcal{E},g,t}(Y', Y) \leqslant 0$. Since $\mathcal{E}$ is a Π-encoder and $\sim_{\mathcal{E},g,t,\mathcal{G}}$ is DP-friendly, it follows from Lemma 9 that $Y \equiv_{\Pi,t} Y'$ and that

[3] Note that we slightly abuse notation when identifying t-protrusions and graphs in $\mathcal{F}_t$, as protrusions are defined as subsets of vertices of a graph. Nevertheless, this will not cause any confusion.

$\Delta_{\Pi,t}(Y',Y) = \Delta_{\mathcal{E},g,t}(Y',Y) \leqslant 0$. Therefore, if we set $k' := k + \Delta_{\Pi,t}(Y',Y)$, it follows that (G,k) and $((G-(Y-\partial(Y)))\ oplus Y', k')$ are indeed equivalent instances of Π with $k' \leqslant k$ and $|Y'| \leqslant b(\mathcal{E},g,t,\mathcal{G})$. ◀

The general recipe to use our framework on a parameterized problem Π defined on a class of graphs $\mathcal{G}$ is as follows: one has just to define the tables to solve Π via dynamic programming on graphs of bounded treewidth (that is, the encoder $\mathcal{E}$ and the function g), check that $\mathcal{E}$ is a Π-encoder and that $\sim_{\mathcal{E},g,t,\mathcal{G}}$ is DP-friendly, and then Theorem 13 provides a linear-time algorithm that replaces large protrusions with graphs whose size is bounded by an explicit constant, and that updates the parameter of Π accordingly. This protrusion replacer can then be used, for instance, whenever one is able to find a linear protrusion decomposition of the input graphs of Π on some sparse graph class $\mathcal{G}$. In particular, Theorem 13 yields the following corollary.

▶ **Corollary 14.** *Let H be a fixed graph, and let $\mathcal{G}$ be the class of graphs that exclude H as a (topological) minor. Let Π be a vertex-certifiable parameterized graph problem on $\mathcal{G}$, and suppose that we are given a Π-encoder $\mathcal{E}$, a function $g : \mathbb{N} \to \mathbb{N}$, and an integer $t \in \mathbb{N}$ such that $\sim_{\mathcal{E},g,t,\mathcal{G}}$ is DP-friendly. Then, given an instance (G,k) of Π together with an $(\alpha \cdot k, t)$-protrusion decomposition of G, we can construct a linear kernel for Π of size at most $(1 + b(\mathcal{E},g,t,\mathcal{G})) \cdot \alpha \cdot k$, where $b(\mathcal{E},g,t,\mathcal{G})$ is the function defined in Lemma 11.*

Proof. For $1 \leqslant i \leqslant \ell$, we apply the polynomial-time algorithm given by Theorem 13 to replace each t-protrusion Y_i with a graph Y_i' of size at most $b(\mathcal{E},g,t,\mathcal{G})$, and to update the parameter accordingly. In this way we obtain an equivalent instance (G',k') such that $G' \in \mathcal{G}$, $k' \leqslant k$, and $|V(G')| \leqslant |Y_0| + \ell \cdot b(\mathcal{E},g,t,\mathcal{G}) \leqslant (1 + b(\mathcal{E},g,t,\mathcal{G}))\alpha \cdot k$. ◀

Notice that once we fix the problem Π and the class of graphs $\mathcal{G}$ where Corollary 14 is applied, a kernel of size $c \cdot k$ can be derived with a concrete upper bound for the value of c. Notice that such a bound depends on the problem Π and the excluded (topological) minor H. In general, the bound can be quite big as it depends on the bound of Lemma 11, and this, in turn, depends on the bound of Lemma 7. However, as we see in the next section, more moderate estimations can be extracted for particular families of parameterized problems.

3 Application to concrete problems

In this section we demonstrate the applicability of our framework by providing linear kernels for several problems on graphs excluding a fixed graph as a (topological) minor. Due to space limitations, we focus here on r-Dominating Set and Planar-$\mathcal{F}$-Deletion. The linear kernel for r-Scattered Set can be found in [10].

The following result will be fundamental in order to find linear protrusion decompositions when a treewidth-modulator X of the input graph G is given, with $|X| = O(k)$. It is a consequence of [14, Lemma 3, Proposition 1, and Theorem 1].

▶ **Theorem 15** (Kim *et al.* [14])**.** *Let c, t be two positive integers, let H be an h-vertex graph, let G be an n-vertex H-topological-minor-free graph, and let k be a positive integer (typically corresponding to the parameter of a parameterized problem). If we are given a set $X \subseteq V(G)$ with $|X| \leqslant c \cdot k$ such that $\mathbf{tw}(G - X) \leqslant t$, then we can compute in time $O(n)$ an $((\alpha_H \cdot t \cdot c) \cdot k, 2t + h)$-protrusion decomposition of G, where α_H is a constant depending only on H, which is upper-bounded by $40h^2 2^{5h \log h}$.*

As mentioned in Subsection 2.2, if $\mathcal{G}$ is a graph class whose membership can be expressed in MSO logic, then $\sim_{\mathcal{G},t}$ has a finite number of equivalence classes, namely $r_{\mathcal{G},t}$. In our applications, we will be only concerned with families of graphs $\mathcal{G}$ that exclude some fixed h-vertex graph H as a (topological) minor. In this case, using standard dynamic programming techniques, it can be shown that $r_{\mathcal{G},t} \leqslant 2^{t \log t} \cdot h^t \cdot 2^{h^2}$. The details can be found in [10].

An explicit linear kernel for r-Dominating Set. Let $r \geqslant 1$ be a fixed integer. We define the r-Dominating Set problem as follows.

r-Dominating Set	
Instance:	A graph $G = (V, E)$ and a non-negative integer k.
Parameter:	The integer k.
Question:	Does G have a set $S \subseteq V$ with $\lvert S\rvert \leqslant k$ and such that every vertex in $V \setminus S$ is within distance at most r from some vertex in S?

For $r = 1$, the r-Dominating Set problem corresponds to Dominating Set. Our encoder for r-Dominating Set is strongly inspired by the work of Demaine *et al.* [5], and it generalizes to one given for Dominating Set in the running example of Section 2. It can be found in [10], and we call it $\mathcal{E}_{r\mathrm{DS}} = (\mathcal{C}_{r\mathrm{DS}}, L_{\mathcal{C}_{r\mathrm{DS}}})$.

▶ **Lemma 16.** [⋆] *The encoder $\mathcal{E}_{r\mathrm{DS}}$ is a rDS-encoder. Furthermore, if $\mathcal{G}$ is an arbitrary class of graphs and $g(t) = t$, then the equivalence relation $\sim_{\mathcal{E}_{r\mathrm{DS}},g,t,\mathcal{G}}$ is DP-friendly.*

We now proceed to construct a linear kernel for r-Dominating Set when the input graph excludes a fixed apex graph H as a minor. Toward this end, the following theorem will play an important role. It follows mainly from the results of Fomin *et al.* [9], but also uses the explicit combinatorial bound of Kawarabayashi and Kobayashi [12] on the relation between the treewidth and the largest grid minor on H-minor-free graphs, and the algorithmic results of Kawarabayashi and Reed [13] in order to obtain the claimed set X.

▶ **Theorem 17** (Fomin *et al.* [9])**.** *Let $r \geqslant 1$ be an integer, let H be an h-vertex apex graph, and let $r\mathrm{DS}_H$ be the restriction of the r-Dominating Set problem to input graphs which exclude H as a minor. If $(G, k) \in r\mathrm{DS}_H$, then there exists a set $X \subseteq V(G)$ such that $|X| = r \cdot 2^{O(h \log h)} \cdot k$ and $\mathbf{tw}(G - X) = r \cdot 2^{O(h \log h)}$. Moreover, given an instance (G, k) of $r\mathrm{DS}_H$ with $|V(G)| = n$, there is an algorithm running in time $O(n^3)$ that either finds such a set X or correctly reports that (G, k) is a* No*-instance.*

We are now ready to present the linear kernel for r-Dominating Set.

▶ **Theorem 18.** *Let $r \geqslant 1$ be an integer, let H be an h-vertex apex graph, and let $r\mathrm{DS}_H$ be the restriction of the r-Dominating Set problem to input graphs which exclude H as a minor. Then $r\mathrm{DS}_H$ admits a constructive linear kernel of size at most $f(r, h) \cdot k$, where f is an explicit function depending only on r and h, defined in Equation (6) below.*

Proof. Given an instance (G, k) of $r\mathrm{DS}_H$, we run the cubic algorithm given by Theorem 17 to either conclude that (G, k) is a No-instance or to find a set $X \subseteq V(G)$ such that $|X| = r \cdot 2^{O(h \log h)} \cdot k$ and $\mathbf{tw}(G - X) = r \cdot 2^{O(h \log h)}$. In the latter case, we use the set X as input to the algorithm given by Theorem 15, which outputs in linear time a $(r^2 \cdot 2^{O(h \log h)} \cdot k, r \cdot 2^{O(h \log h)})$-protrusion decomposition of G. We now consider the encoder $\mathcal{E}_{r\mathrm{DS}} = (\mathcal{C}_{r\mathrm{DS}}, L_{\mathcal{C}_{r\mathrm{DS}}})$ defined in [10]. By Lemma 16, $\mathcal{E}_{r\mathrm{DS}}$ is an rDS-encoder and $\sim_{\mathcal{E}_{r\mathrm{DS}},g,t,\mathcal{G}}$ is DP-friendly, where $\mathcal{G}$ is the class of H-minor-free graphs and $g(t) = t$. It can be proved

that $s_{\mathcal{E}_{r\mathrm{DS}}}(t) \leqslant (2r+1)^t$ (see [10]). Therefore, we are in position to apply Corollary 14 and obtain a linear kernel for $r\mathrm{DS}_H$ of size at most

$$r^2 \cdot 2^{O(h \log h)} \cdot b\left(\mathcal{E}_{r\mathrm{DS}}, g, r \cdot 2^{O(h \log h)}, \mathcal{G}\right) \cdot k \ , \tag{6}$$

where $b\left(\mathcal{E}_{r\mathrm{DS}}, g, r \cdot 2^{O(h \log h)}, \mathcal{G}\right)$ is the function defined in Lemma 11. ◀

It can be easily checked that the multiplicative constant involved in the upper bound of Equation (6) is $2^{2^{2^{r \cdot \log r \cdot 2^{O(h \cdot \log h)}}}}$, that is, it depends triple-exponentially on the integer r.

An explicit linear kernel for Planar-$\mathcal{F}$-Deletion. Let $\mathcal{F}$ be a finite set of graphs. We define the $\mathcal{F}$-Deletion problem as follows.

$\mathcal{F}$-Deletion	
Instance:	A graph G and a non-negative integer k.
Parameter:	The integer k.
Question:	Does G have a set $S \subseteq V(G)$ such that $\lvert S\rvert \leqslant k$ and $G - S$ is H-minor-free for every $H \in \mathcal{F}$?

When all the graphs in $\mathcal{F}$ are connected, the corresponding problem is called Connected-$\mathcal{F}$-Deletion, and when $\mathcal{F}$ contains at least one planar graph, we call it Planar-$\mathcal{F}$-Deletion. When both conditions are satisfied, the problem is called Connected-Planar-$\mathcal{F}$-Deletion. Note that Connected-Planar-$\mathcal{F}$-Deletion encompasses, in particular, Vertex Cover and Feedback Vertex Set. We obtain a linear kernel for the problem using two different approaches. The first one follows the same scheme as the one used so far, that is, we first find a treewidth-modulator X in polynomial time, and then we use this set X as input to the algorithm of Theorem 15 to find a linear protrusion decomposition of the input graph. In order to find the treewidth-modulator X, we need that the input graph G excludes a fixed graph H as a minor. With our second approach, that can be found in [10], we obtain a linear kernel on the larger class of graphs that exclude a fixed graph H as a *topological* minor. We provide two variants of this second approach. One possibility is to use the randomized constant-factor approximation for Planar-$\mathcal{F}$-Deletion by Fomin *et al.* [8] as treewidth-modulator, which yields a randomized linear kernel that can be found in uniform polynomial time. The second possibility consists in arguing just about the *existence* of a linear protrusion decomposition in Yes-instances, and then greedily finding large protrusions to be reduced by the protrusion replacer of Theorem 13. This yields a deterministic linear kernel that can be found in time $n^{f(H,\mathcal{F})}$, where f is a function depending on H and $\mathcal{F}$.

Our encoder for the $\mathcal{F}$-Deletion problem (see [10]) uses the dynamic programming machinery developed by Adler *et al.* [1]. We prove that this encoder is indeed an $\mathcal{F}$-Deletion-encoder and that the corresponding equivalence relation is DP-friendly, under the constraint that all the graphs in $\mathcal{F}$ are *connected*. Interestingly, this phenomenon concerning the connectivity seems to be in strong connection with the fact that the $\mathcal{F}$-Deletion problem has FII if all the graphs in $\mathcal{F}$ are connected [3, 8], but for some families $\mathcal{F}$ containing disconnected graphs, $\mathcal{F}$-Deletion has not FII (see [14] for an example of such family).

▶ **Theorem 19.** [⋆] *Let $\mathcal{F}$ be a finite set of connected graphs containing at least one r-vertex planar graph F, let H be an h-vertex graph, and let $\mathrm{CP}\mathcal{F}\mathrm{D}_H$ be the restriction of the* Connected-Planar-$\mathcal{F}$-Deletion *problem to input graphs which exclude H as a minor. Then $\mathrm{CP}\mathcal{F}\mathrm{D}_H$ admits a constructive linear kernel of size at most $f(r,h) \cdot k$, where f is an explicit function depending only on r and h, which can be found in [10].*

4 Further research

The methodology for performing explicit protrusion replacement via dynamic programming that we have presented is quite general, and it could also be used to obtain polynomial kernels (not necessarily linear). We have restricted ourselves to vertex-certifiable problems, but is seems plausible that our approach could be also extended to edge-certifiable problems or to problems on directed graphs.

The linear kernel for Planar-$\mathcal{F}$-Deletion requires that all graphs in the family $\mathcal{F}$ are *connected*. It would be interesting to get rid of this assumption. All the applications examined in this paper concerned parameterized problems tuned by a secondary parameter, i.e., r for the case of r-Dominating Set and r-Scattered Set and the size of the graphs in $\mathcal{F}$ for the case of $\mathcal{F}$-Deletion. In all kernels derived for these problems, the dependency on this secondary parameter is triple-exponential, while the dependency on the choice of the excluded graph H is one exponent higher. Improving these dependencies on the "meta-parameters" is worth being investigated, as well as examining to what extent this exponential dependency is unavoidable under some assumptions based on automata theory or (parameterized) complexity theory.

References

1 Isolde Adler, Frederic Dorn, Fedor V. Fomin, Ignasi Sau, and Dimitrios M. Thilikos. Faster parameterized algorithms for minor containment. *Theoretical Comput. Science*, 412(50):7018–7028, 2011.

2 J. Alber, M.R. Fellows, and R. Niedermeier. Polynomial-Time Data Reduction for Dominating Set. *Journal of the ACM*, 51(3):363–384, 2004.

3 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. In *Proc. of the 50th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 629–638. IEEE Computer Society, 2009.

4 Bruno Courcelle. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1):12–75, 1990.

5 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Transactions on Algorithms*, 1(1):33–47, 2005.

6 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

7 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer Verlag, 2006.

8 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *Proc. of the 53rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 470–479, 2012.

9 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *Proc. of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 503–510. SIAM, 2010.

10 Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Explicit linear lernels via dynamic programming. *CoRR*, abs/1312.6585, 2013.

11 Jiong Guo and Rolf Niedermeier. Linear problem kernels for NP-hard problems on planar graphs. In *Proc. of the 34th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4596 of *LNCS*, pages 375–386, 2007.

12 Ken ichi Kawarabayashi and Yusuke Kobayashi. Linear min-max relation between the treewidth of H-minor-free graphs and its largest grid. In *Proc. of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 14 of *LIPIcs*, pages 278–289, 2012.

13 Ken ichi Kawarabayashi and Bruce Reed. A Separator Theorem in Minor-Closed Classes. In *Proc. of the 51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 153–162. IEEE Computer Society, 2010.

14 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 *Proc. of the 40th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 7965 of *LNCS*, pages 613–624, 2013.

Partition Expanders*

Dmitry Gavinsky and Pavel Pudlák

Institute of Mathematics, Academy of Sciences, Prague, Czech Republic
dmitry.gavinsky@gmail.com, pudlak@math.cas.cz

Abstract

We introduce a new concept, which we call *partition expanders*. The basic idea is to study quantitative properties of graphs in a slightly different way than it is in the standard definition of expanders. While in the definition of expanders it is required that the number of edges between any pair of sufficiently large sets is close to the expected number, we consider *partitions* and require this condition only for *most of the pairs* of blocks. As a result, the blocks can be substantially smaller.

We show that for some range of parameters, to be a partition expander a random graph needs *exponentially smaller* degree than any expander would require in order to achieve similar expanding properties.

We apply the concept of partition expanders in communication complexity. First, we give a PRG for the SMP model of the optimal seed length, $n + O\log k$. Second, we compare the model of SMP to that of Simultaneous Two-Way Communication, and give a new separation that is stronger both qualitatively and quantitatively than the previously known ones.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases partitions, expanders, communication, complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.325

1 Introduction

Expanders are a very interesting and useful concept and appear in many applications in computer science. Therefore several related concepts have been introduced; e.g., lossless expanders [6], monotone expanders and dimension expanders [7], superexpanders [13].

In this paper we introduce yet another concept that we call *partition expanders*. The definition is motivated by the following observation. The well-known Expander-Mixing Lemma says, roughly speaking, that for every two sufficiently big sets of vertices A and B the number of edges of the expander between A and B is close to $\frac{d}{n} \cdot |A| \cdot |B|$, where n is the number of vertices and d is the degree. If we want to apply this lemma to smaller sets, we have to increase the degree of expanders appropriately.

Now suppose we have a partition of the vertices of the graph and we only want to satisfy the density condition for most of the pairs of sets. It turns out that a random graph with relatively small degree is able to satisfy this condition for partitions with many blocks, although the Expander-Mixing Lemma is not able to give any interesting estimate. So while expanders are graphs with "typical connectivity" with respect to *subsets of vertices*, partition expanders have "typical connectivity" with respect to *partitions of vertices*. Informally speaking, in the context of expanders, partitions are "more structured" objects than subsets, and therefore demanding the same "expanding performance" with respect to partitions can be

* Partially funded by the grant P202/12/G061 of GA ČR and by RVO: 67985840.



© Dmitry Gavinsky and Pavel Pudlák;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 325–336



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



viewed as a relaxation of usual expanders. In return, we expect partition expanders to have considerably smaller degree than usual expanders with the same expanding performance.

There are several possible ways to formally define a partition expander. We choose the following definition as "canonical" due to its brevity and robustness. We will give alternative definitions shortly.

▶ **Definition 1.** Partition expanders Let $G = (V, E)$ be an (undirected) graph. Let μ be the uniform distribution over $V \times V$, and let μ_G be the uniform distribution over E. For any coloring $c : V \to [K]$, let ν^c and ν^c_G be the distributions of the pair $(c(v_1), c(v_2))$ when (v_1, v_2) is chosen according to μ or μ_G, respectively.

For $K \in \mathbb{N}$ and $\delta \in (0, 1)$, we say that G is a (K, δ)-*partition expander* if for every coloring $c : V \to [K]$ the statistical distance between ν^c and ν^c_G is at most δ.

It should be noted that this concept is interesting in the situations where the number K of partitions is increasing with the number of vertices and the graphs are d-regular with d increasing. We are mainly interested in the question of how small d can be for a given K, assuming $0 < \delta < 1$ is a fixed constant.

1.1 Our results

We start by giving several equivalent definitions of partition expanders, which emphasize the fact that they are a natural modification of usual expanders.

In Section 3 we analyze the behavior of random graphs as partition expanders. We prove that random d-regular graphs almost always are good partition expanders – the dependence of K on d is the best possible, namely exponential.

In Section 4 the notion of partition expanders is advocated through comparing it to expanders. We show that the gap between the absolute values of the first two eigenvalues does not ensure that the graph is a good partition expander. Namely, if only the spectral gap is taken into account when a partition expander is constructed, then the degree has to be *exponentially larger* than an optimal partition expander requires. Since the spectral gap characterizes almost tightly the expander properties of a graph, this demonstrates exponential advantage of partition expanders (in those scenarios when they are suitable) over expanders. In other words, if "partition expansion" is the desired behavior, then using an expander instead of an optimal partition expander would incur exponential loss in terms of the required degree.

Based on the spectral properties only, we use the Hoffman-Wielandt inequality and get a slightly better bound than what would follow from a direct application of the Expander-Mixing Lemma.[1] The fact that the spectral gap is incapable to characterize good partition expanders partially explains why new methods are required for their construction.

In Section 5 we present another equivalent definition of partition expanders. We show that a graph $G = (V, E)$ is a partition expander if and only if the uniform distribution over E is a *Pseudo-Random Generator (PRG)* in the setting of *Simultaneous Message Passing (SMP)* in communication complexity. We use this fact to give a lower bound on the degree of partition expanders, thus showing *optimality* of the randomized construction given in Section 3.

In the second part of Section 5 we show two applications of our randomized construction of a partition expander. First, we construct a PRG against SMP protocols of communication

[1] We get quadratic improvement in terms of the partition size, and show that it is essentially optimal general bound in terms of the spectral gap alone.

cost k that requires seed length $n + O(\log k)$ (see Theorem 15 and the comment thereafter).[2] Second, we compare the model of SMP to that of Simultaneous Two-Way Communication, and give a new separation that is stronger both qualitatively and quantitatively than the previously known ones (see Theorem 19).

2 Notation and more

Unless stated otherwise, all sets are assumed to be finite, and all graphs are *undirected* and *simple* (having no self loops or multiple edges).[3] For two subsets $S_1, S_2 \subseteq V$, we denote by $E(S_1, S_2)$ the set of ordered pairs (v_1, v_2) such that (v_1, v_2) is an edge in E, $v_1 \in S_1$ and $v_2 \in S_2$, and write $E(v_1, v_2)$ for $E(\{v_1\}, \{v_2\})$.[4] We will say that a set family $\sigma = \{C_1, \ldots, C_K\}$ is a K-partition of a set X if $\cup_{i=1}^{K} C_i = X$ and $C_1, \ldots, C_K$ are pairwise disjoint and nonempty.

The *statistical distance* between two distributions μ_1 and μ_2 defined over a set X is

$$\mathrm{d}_{\mathrm{st}}(\mu_1, \mu_2) \stackrel{\mathrm{def}}{=} \frac{1}{2} \sum_{x \in X} |\mu_1(x) - \mu_2(x)| .$$

▶ **Lemma 2.** *Let $K \in \mathbb{N}$ and $\delta \in \mathbb{R}$. The following statements are equivalent:*

1. *$G = (V, E)$ is a (K, δ)-partition expander.*
2. *For every K-partition $\sigma = \{C_1, \ldots, C_K\}$ of V,*

$$\delta \geq \frac{1}{2} \sum_{i,j \in [K]} \left| \frac{|E(C_i, C_j)|}{|E|} - \frac{|C_i| \cdot |C_j|}{|V|^2} \right| . \tag{1}$$

3. *For every K-partition σ and $S \subseteq [K] \times [K]$,*

$$\delta \geq \sum_{(i,j) \in S} \left(\frac{|E(C_i, C_j)|}{|E|} - \frac{|C_i| \cdot |C_j|}{|V|^2} \right) = \frac{\sum_S |E(C_i, C_j)|}{|E|} - \frac{\sum_S |C_i| \cdot |C_j|}{|V|^2} . \tag{2}$$

4. *Like 0c, but only over symmetric S (i.e., $(i, j) \in S \Leftrightarrow (j, i) \in S$).*

Proof. Equivalence between 0a and 0b is immediate from Definition 1. Equivalence between 0c and 0d follows from the fact that G is undirected. To see that 0b is equivalent to 0c, note that

$$\sum_{i,j \in [K]} \left(\frac{|E(C_i, C_j)|}{|E|} - \frac{|C_i| \cdot |C_j|}{|V|^2} \right) = \frac{\sum_{[K] \times [K]} |E(C_i, C_j)|}{|E|} - \frac{\sum_{[K] \times [K]} |C_i| \cdot |C_j|}{|V|^2} = 0.$$

◀

There are many possible ways to define expanders. The standard definition is based on the second largest absolute value of an eigenvalue of a graph G, which we will denote by $\lambda(G)$.

[2] All previously known PRGs in communication complexity were given against stronger models, thus requiring exponentially larger "overhead" over n in terms of seed length – for details, see Section 5.

[3] In those cases when we explicitly allow multiple edges, the edges of a graph will be viewed as a collection with repetitions.

[4] Note that if $v_1, v_2 \in S_1 \cap S_2$, then the edge (v_1, v_2) appears in $E(S_1, S_2)$ twice: as ordered pairs (v_1, v_2) and (v_2, v_1).

▶ **Definition 3** (Expanders). A regular graph G is an ℓ-expander if $\lambda(G) \leq \ell$.

We will denote the degree of a regular graph G by $d(G)$, or simply by d when G is clear from the context.

The most natural relation between expanders and partition expanders comes from the following well-known fact (e.g., see [2]).

▶ **Lemma 4** (Expander-Mixing Lemma). *Let (V, E) be an ℓ-expander. Then for every $S_1, S_2 \subseteq V$,*

$$\left| \frac{|E(S_1,S_2)|}{|E|} - \frac{|S_1| \cdot |S_2|}{|V|^2} \right| \leq \ell \cdot \frac{\sqrt{|S_1| \cdot |S_2|}}{|E|} = \frac{\ell}{d} \cdot \frac{\sqrt{|S_1| \cdot |S_2|}}{|V|}.$$

One can show using this lemma that an ℓ-expander is a (K, δ)-partition expander for constant $\delta > 0$ and certain $K \in \Theta(d/\ell)$ – however, this trivial arguments fails for $K \geq d/\ell$. In Section 4 we will use the Hoffman-Wielandt inequality to show that an ℓ-expander is a $(K, \Omega(1))$-partition expander for certain $K \in \Theta\big((d/\ell)^2\big)$, and that will be shown to be optimal up to the factor of $\log n$.

▶ **Theorem 5** (Hoffman-Wielandt inequality [9]). *If A and B are normal matrices with respective eigenvalues $\lambda_1(A), \ldots, \lambda_n(A)$ and $\lambda_1(B), \ldots, \lambda_n(B)$, then*

$$\min_{\pi} \left\{ \sum_{i=1}^{n} \left| \lambda_i(A) - \lambda_{\pi(i)}(B) \right|^2 \right\} \leq \|A - B\|_F^2 ,$$

where π runs over all permutations over $[n]$ and $\| \ldots \|_F^2$ denotes the square of the Frobenius norm (the sum of squares of the absolute values of the elements).

If A and B are symmetric real matrices, we can drop the absolute value and write the terms as $\lambda_i(A)^2 + \lambda_{\pi(i)}(B)^2 - 2\lambda_i(A)\lambda_{\pi(i)}(B)$. Since the sum of the squares of eigenvalues of a matrix is the square of its Frobenius norm, the inequality is equivalent to

$$\sum_{i,j} a_{ij} b_{ij} \leq \max_{\pi} \left\{ \sum_{i=1}^{n} \lambda_i(A) \lambda_{\pi(i)}(B) \right\}. \tag{3}$$

Let $d, n \in \mathbb{N}$ be such that $2|dn$, denote by $\mathcal{G}_{n,d}$ the uniform distribution on d-regular (simple undirected) graphs on n vertices. In our analysis we will use the *pairing method* for generating $G \sim \mathcal{G}_{n,d}$, due to Bollobás [5] (also see [14]).

▶ **Lemma 6** (Pairing method [5]). *The following procedure generates $E \subseteq [n] \times [n]$ such that $G = ([n], E) \sim \mathcal{G}_{n,d}$.*

1. *Let $\pi \subset [nd] \times [nd]$ be a uniformly random perfect matching on $[nd]$ (viewed as a symmetric set of directed edges). For $i \in [n]$, let $cell_i \stackrel{def}{=} \{x \mid id - d < x \leq id\}$ and $d_\pi(v_1, v_2) \stackrel{def}{=} |\pi(cell_{v_1}, cell_{v_2})|$.*
2. *For every $(v_1, v_2) \in [n] \times [n]$, let (v_1, v_2) be $d_\pi(v_1, v_2)$ times an element of E.*
3. *Return to Step 0a if $G = ([n], E)$ is not simple.*

In the analysis we will consider the distribution of $([n], E)$ resulting from dropping Step 0c off the above procedure; let us denote it by $\mathcal{G}'_{n,d}$. Observe that a graph $G \sim \mathcal{G}'_{n,d}$ is always undirected, but doesn't have to be simple.[5]

We will use the following estimate, due to McKay and Wormald [12]:

[5] Note also that the distribution $\mathcal{G}'_{n,d}$ is not uniform over its support - e.g., $\mathcal{G}'_{2,2}$ produces the graph with two parallel edges with probability 2/3.

▶ **Lemma 7** ([12]). *For $d \in o(\sqrt{n})$,*

$$\Pr_{G\sim\mathcal{G}'_{n,d}}[G \text{ is simple}] \in \exp\left(\frac{1-d^2}{4} - \frac{d^3}{12n} + O\left(\frac{d^2}{n}\right)\right) \subseteq \exp(o(n)).$$

3 Random d-regular graphs as partition expanders

Let us see that a random regular graph is likely to form a partition expander.

▶ **Theorem 8.** *For $d \in O\left(n^{1/3}\right)$, a random d-regular simple undirected graph on n vertices is a (K,δ)-partition expander with probability at least $1 - \exp\left(n\log K + K^2 - \Omega\left(\delta^2 nd\right)\right)$.*

The proof can be found in the full version of the paper.

▶ **Corollary 9.** *For any $\varepsilon > 0$ and $B \in \mathbb{N}$ there exists $C \in \mathbb{N}$, such that the following holds: A random d-regular graph on n vertices is a (K,δ)-partition expander with probability at least $1-\varepsilon$, as long as $K \le B\cdot\sqrt{n}$ and $d \ge \frac{C\cdot\log K}{\delta^2}$.*

4 Partition expanders vs. expanders

Let us compare the notions of expanders and partition expanders in more detail.

▶ **Theorem 10.** *Let G be a d-regular ℓ-expander on n vertices. Then it is a $(K, \sqrt{K}\ell/d)$-partition expander for every $K < d^2/\ell^2$.*

The proof is based on the Hoffman-Wielandt inequality and can be found in the full version of the paper.

Note that the Expander-Mixing Lemma (Lemma 4) only gives that G is a (K,δ) partition expander for $\delta = O(K\ell/d)$, which is meaningful only for $K < d/\ell$. The statement of the above theorem is essentially tight (cf. Theorem 12), and this means that only small (quadratic, in terms of K vs. d) improvement can result from using partition expanders instead of expanders, as long as the construction of a partition expanders relies on the spectral gap. On the other hand, we will see soon that good partition expanders offer *exponential* improvement in terms of the dependence of K on d.

Now we will show that the above bound is essentially optimal, and therefore, in general expanders are not good partition expanders. We will use the following result of Alon and Roichman [1]. (For a simpler proof, and an explicit and better bound, see [11].)

▶ **Theorem 11** ([1, 11]). *There exists an absolute constant c such that for every finite group Γ and any $d \le |\Gamma|$, the following is true. If we pick uniformly at random the elements $g_1,\ldots,g_d \in \Gamma$, then the resulting Cayley-graph has the second largest eigenvalue λ satisfying*

$$\lambda \le c\cdot\sqrt{d\log|\Gamma|}$$

with probability going to 1 as $|\Gamma| \to \infty$.

This theorem is not stated explicitly in those papers, but it is an immediate corollary of Theorem 2 of [11]. (One can take any constant c such that $c > 2\ln 2$.)

Let $m > 0$ be a natural number and let Γ be the symmetric group on m elements represented by permutations of $[m]$. Let $\pi_1,\ldots,\pi_d$ be some permutations for which the bound on the eigenvalue is satisfied. W.l.o.g. we will assume that for every $i \in [d]$ there is a $j \in [d]$ such that $\pi_j = \pi_i^{-1}$. Let G be the Cayley graph determined by Γ and $\pi_1,\ldots,\pi_d$.

Let $1 \leq t \leq m$. We will consider the partition $\{C_1, \ldots, C_K\}$ defined by the following equivalence relation on G

$$\rho|_{[t]} = \sigma|_{[t]},$$

where $\rho, \sigma \in G$ are permutations and $|_{[t]}$ denote their restriction to the first t elements. Thus the number of blocks is $K = m(m-1)\ldots(m-t+1)$. Consider the symmetric set S defined by

$$(i,j) \in S \;\equiv\; \exists \rho \in C_i, \sigma \in C_j \; \exists s \in [d] \; \rho|_{[t]} = \pi_s \sigma|_{[t]} \,. \tag{4}$$

Note that if for some i and j the condition is satisfied by some $s = s_0$, then for all $\rho \in C_i, \sigma \in C_j$, we have $\rho|_{[t]} = \pi_{s_0}\sigma|_{[t]}$.

Consider the equation (2) that defines partition expanders. The first term is in our case equal to 1. To bound the second term, note that for a given $s \in [d]$ the number of pairs ρ, σ satisfying the condition $\rho|_{[t]} = \pi_\ell \sigma|_{[t]}$ is $m!(m-t)!$. Hence the second term is bounded by

$$\frac{d \cdot m!(m-t)!}{(m!)^2} = \frac{d}{m(m-1)\ldots(m-t+1)} = \frac{d}{K}.$$

This proves that if $d/K < 1-\delta$, then G is not a (K,δ)-partition expander.

Thus we have proved:

▶ **Theorem 12.** *There exist a constant c such that for infinitely many n and every $d \leq n$, there are d-regular $c\sqrt{d \log n}$-expanders on n vertices which are not $(K, 1 - \frac{d+1}{K})$-partition expanders.*

Comparing this statement to the bound given by Theorem 10 in the most natural regime when a $(K, 1-\Omega(1))$-partition expander is required, we can see that the upper and the lower bounds match up to the factor of $\log n$ in the spectral gap. In particular, since the second eigenvalue of a graph is always $\Omega\left(\sqrt{d}\right)$, K can be at most linear in d, as long as our only assumption about G is the absolute value of its second eigenvalue. In contrast to this, according to Corollary 9, there exist $(K, 1-\Omega(1))$ partition expanders whose degree is $O(\log K)$. Thus any construction of such partition expanders must rely on some properties of G, other than the spectral gap.

5 Partition expanders as PRGs in communication complexity

Let us turn to the realm of communication complexity, where we give an equivalent formulation of partition expanders. First, we use this equivalence to give a nearly-tight lower bound on the degree of good partition expanders, thus arguing near-optimality of the randomized construction given in Section 3. Second, we use the same construction to obtain a new separation between two models of communication complexity, which is qualitatively stronger than the previously known one.

We will use the following models of two-party communication complexity.

▶ **Definition 13** (Models of communication complexity)**.** Two players whose names are Alice and Bob each receive a binary string of length n, respectively denoted by x and y. Players' goal is to compute the value of $f(x,y)$, where $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is fixed. The players obey the following scenario:

- In the model of *Simultaneous Message Passing* (SMP), denoted by $\mathcal{R}^{\|}$, both Alice and Bob send a message to the third participant, the referee. The referee does not know the values of x and y, so his only input are the messages received from the players, and he has to produce the answer using the information received from the players. All three participants are allowed to use private randomness.
- The model of *SMP with shared randomness,* denoted by $\mathcal{R}^{\|,pub}$, is similar to $\mathcal{R}^{\|}$ but the players are allowed to use public randomness.[6]
- In the model of *One-Way Communication,* denoted by $\mathcal{R}^{1}$, Alice sends her message to Bob, who has to produce the answer using his part of the input and the information received from Alice.
- In the model of *Simultaneous Two-Way Communication,* denoted by $\mathcal{R}^{\leftrightarrow}$, Alice and Bob send their messages simultaneously, similarly to the case of SMP. But here the recipient of Alice's message is Bob, and the recipient of Bob's message is Alice. Upon receiving the partner's message, each player must produce an answer.

We say that a communication protocol solves the problem represented by f if it produces the correct answer(s) with probability at least $2/3$ for every possible input. The communication cost of a protocol is the maximal total number of bits sent by the players, and the communication cost of a function f is the minimal communication cost of a protocol that solves it in the given model.

The models $\mathcal{R}^{\|}$, $\mathcal{R}^{\|,pub}$ and $\mathcal{R}^{1}$ have been studied widely and the corresponding notation is commonly used; the Simultaneous Two-Way model has been considered in several works (see below), but no specific name was assigned to it. Note that when we say that an $\mathcal{R}^{\leftrightarrow}$-protocol has produced the answer "a", we refer to the situation when both the players have produced the same answer.

▶ **Definition 14** (Pseudo-randomness in communication complexity)**.** Let $\mathcal{M}$ be a communication complexity model, and let μ be a distribution defined over $\{0,1\}^n \times \{0,1\}^n$. We say that μ is k-pseudo-random for $\mathcal{M}$ if for any protocol $\mathcal{P}$ of communication cost at most k it holds that

$$\mathop{\mathbf{Pr}}_{(X,Y)\sim\mu}[\mathcal{P}(X,Y) \text{ outputs "1"}] - \mathop{\mathbf{Pr}}_{(X,Y)\sim\mathcal{U}_{\{0,1\}^n\times\{0,1\}^n}}[\mathcal{P}(X,Y) \text{ outputs "1"}] < \frac{1}{3}.$$

We say that $g : \{0,1\}^s \to \{0,1\}^n \times \{0,1\}^n$ is a k-Pseudo-Random Generator (k-PRG) of seed length s against $\mathcal{M}$ if the distribution of $g(X)$ when $X \sim \mathcal{U}_{\{0,1\}^s}$ is k-pseudo-random for $\mathcal{M}$.

Pseudo-randomness in the context of communication complexity has been introduced in [10]. Intuitively, both pseudo-randomness and lower bounds on communication cost can be viewed as claims that certain problem is hard for the model under consideration.

Given a d-regular graph $G = (\{0,1\}^n, E)$, let μ_G be the uniformly random distribution of the edges from E. Note that in order to choose $(v_1, v_2) \sim \mu_G$, a "seed" of length $n + \log d$ is both necessary and sufficient.

▶ **Theorem 15.** *Let $k, n \in \mathbb{N}$ and $G = (\{0,1\}^n, E)$. The following statements are equivalent:*

1. *G is a $(2^{\Theta(k)}, \delta)$-partition expander for some $\delta < 1/3$.*
2. *μ_G is $\Theta(k)$-pseudo-random for $\mathcal{R}^{\|}$.*

[6] Note that in the communication complexity setting Alice and Bob collaborate, and therefore availability of public randomness is equivalent to players' ability to use mixed strategies.

In particular, our construction in Section 3 corresponds to a k-PRG against $\mathcal{R}^{\|}$ of seed length $n+O(\log k)$. Note that due to the fact that in the context of communication complexity the players are computationally unlimited, a randomized construction of a PRG is neither meaningless nor trivial.[7]

Proof. Let C be a constant. First, suppose that G is a $(2^{Ck}, \delta)$-partition expander. Let $\mathcal{P}$ be an $\mathcal{R}^{\|}$-protocol of cost at most Ck, and let us show that it cannot distinguish with high confidence μ_G from $\mathcal{U}_{\{0,1\}^n \times \{0,1\}^n}$. Without loss of generality assume that $\mathcal{P}$ is deterministic, and let $\alpha : \{0,1\}^n \to \{0,1\}^{Ck}$ be the mapping from x to the concatenation of Alice's and Bob's messages in response to the input (x, x). Let $\nu_{\mathcal{U}}$ and ν_G be the distributions of $(\alpha(X), \alpha(Y))$ when, respectively, $(X,Y) \sim \mathcal{U}_{\{0,1\}^n \times \{0,1\}^n}$ and $(X,Y) \sim \mu_G$. Clearly,

$$\Pr_{(X,Y)\sim\mu_G}[\mathcal{P}(X,Y) \text{ outputs "1"}] - \Pr_{(X,Y)\sim\mathcal{U}_{\{0,1\}^n\times\{0,1\}^n}}[\mathcal{P}(X,Y) \text{ outputs "1"}] \leq \mathrm{d}_{\mathrm{st}}(\nu_G, \nu_{\mathcal{U}}).$$

Note that α defines a partition of $\{0,1\}^n$ into at most 2^{Ck} blocks, and by the definition of partition expanders,

$$\mathrm{d}_{\mathrm{st}}(\nu_G, \nu_{\mathcal{U}}) \leq \delta < 1/3.$$

Therefore, μ_G "fools" $\mathcal{P}$ and thus it is Ck-pseudo-random for $\mathcal{R}^{\|}$.

Now assume that μ_G is $2Ck$-pseudo-random for $\mathcal{R}^{\|}$, and let us show that G is a partition expander. Let $\sigma = \{S_1, \ldots, S_{2^{Ck}}\}$ be a partition of $\{0,1\}^n$, and for $x \in \{0,1\}^n$, define $\sigma(x) \stackrel{\text{def}}{=} i$ for i such that $x \in S_i$. Let $\mathcal{P}_\sigma$ be an $\mathcal{R}^{\|}$-protocol, where upon receiving input (X,Y), Alice sends $\sigma(X)$ and Bob sends $\sigma(Y)$. Let $\tau_{\mathcal{U}}$ and τ_G be the distributions of $(\sigma(X), \sigma(Y))$ when, respectively, $(X,Y) \sim \mathcal{U}_{\{0,1\}^n \times \{0,1\}^n}$ and $(X,Y) \sim \mu_G$. Note that $\mathcal{P}_\sigma$ is of cost $2Ck$, and therefore

$$\mathrm{d}_{\mathrm{st}}(\tau_{\mathcal{U}}, \tau_G) < 1/3,$$

since otherwise the referee would be able to distinguish the two cases with confidence high enough to contradict pseudo-randomness of μ_G. Let δ be the maximum value of $\mathrm{d}_{\mathrm{st}}(\tau_{\mathcal{U}}, \tau_G)$ possible under any choice of 2^{Ck}-partition σ, then $\delta < 1/3$ and G is a $(2^{Ck}, \delta)$-partition expander, as required. ◀

5.1 Lower bound on the degree of partition expanders

Let us use the correspondence between partition expanders and pseudo-random generators given by Theorem 15 in order to get a lower bound on the degree of partition expanders.

▶ **Theorem 16.** *For any $\delta < 1/3$, if a d-regular graph G is a (K, δ)-partition expander then $d \in \Omega\left(\frac{\log K}{\log\log K}\right)$.*

In particular, the randomized construction given in Section 3 is optimal, up to the multiplicative $\log\log K$ factor.

[7] For example, the models $\mathcal{R}^{1}$ and $\mathcal{R}^{\leftrightarrow}$ (and more generally, any two-party model where a k-bit message from one player reaches the other player, who also receives his own n bits of input) require seed length at least $n + k - O(1)$ even with a non-uniform PRG, as witnessed by the protocol where the sender sends the first k bits of his input and the computationally-unlimited recipient outputs "1" only if the message together with his own n bits of input have Kolmogorov complexity $n + k - O(1)$.

Proof. For convenience, let n and d be powers of 2. Let $G = ([n], E)$, and assume it is a (K, δ)-partition expander. On the one hand, according to Theorem 15, μ_G is $\Omega(\log K)$-pseudo-random for the SMP model. On the other hand, we will see below that an SMP protocol of cost $O(d \log d)$ can distinguish μ_G from the uniform distribution with error at most $1/4$, and therefore $d \in \Omega\left(\frac{\log K}{\log \log K}\right)$, as required.

The distinguishing protocol is as follows. When her input is $v \in V$, Alice sends to the referee the first $\log d + 2$ bits of the indices of the d neighbors of v. On input $u \in V$, Bob sends to the referee the first $\log d + 2$ bits of the index of u. The referee guesses that the input pair (v, u) has been drawn from the distribution μ_G if the index-prefix received from Bob appears in the list of d index-prefixes received from Alice. This protocol is always correct if the input comes from the support of μ_G, and errs with probability at most $1/4$ when the input comes from the uniform distribution. ◀

5.2 Model separations based on PRGs

Model separation in computational complexity usually means demonstrating existence of a computational problem that can be solved efficiently in one model, but not in the other. If several classes of problems can be handled by the models under consideration, one can define the corresponding *types* of model separations. When one problem class is a special case of another, separation via an element of the smaller class can be viewed as a stronger indication that the compared models have different computational power than separation via an element of the bigger class. These ideas can be pushed further, resulting in various "hierarchies" of model separations.

In the case of communication complexity, there are at least four natural classes of computational problems[8], namely:

- Total functions $f : A \times B \to Z$
- Partial functions $f : C \to Z$, $C \subseteq A \times B$
- Relations $\mathcal{P} \subseteq A \times B \times Z$
- Distinguishing some distribution μ defined on $A \times B$ from the uniform (cf. Definition 14)

Consider the four types of model separations corresponding to these four classes. We will call the fourth type *separation via a PRG*. Obviously, if two communication models are separable via a total function they are also separated via a partial function, and separability via a partial function implies separability via a relation. On the other hand, there are pairs of communication models that can be separated via a relation but not via a partial function (e.g., see [8]), and there are many pairs of models that have been separated via partial functions, but are *conjectured* not to be separable via total functions (e.g., most of quantum communication models form such pairs with their natural classical counterparts). Therefore, in communication complexity it is always desirable to separate models via the "most limited" possible type of separation, as that gives the "strongest" possible indication of difference in the computational power of those models.

To the best of our knowledge, separation via a PRG has not been studied in the context of communication complexity. It is probably incomparable to the first three types of separation: On the one hand, it is straightforward to get a separation via a PRG by modifying slightly one of the known separations via a partial function between quantum and classical one-way

[8] The same applies to many other fields of complexity, where also most of the following discussion remains valid – e.g., in the field of circuit complexity.

models, but it is *conjectured* that those two models cannot be separated via a total function. Therefore, modulo that conjecture, separation via a PRG cannot, in general, be as limited as separation via a total function. On the other hand, the models $\mathcal{R}^{\|}$ and $\mathcal{R}^{\|,pub}$ cannot be separated via a PRG (in general, it is easy to see that for any distribution-distinguishing task there exists an optimal protocol that does not need any randomness), but they can be separated via a total function – e.g., the equality function. Therefore, separation via a total function cannot, in general, be as limited as separation via a PRG.

Is there a type of model separation that would be the most limited, and therefore separations demonstrated through it would be the most "convincing" indication of difference in the computational power of the compared models?

Take a total Boolean function $f : A \times B \to \{0,1\}$, let $\mathcal{M}$ be a communication complexity model, and consider the following two claims:

- No protocol in $\mathcal{M}$ of cost less than k can compute f.
- The distributions $\mathcal{U}_{f^{-1}(0)}$ and $\mathcal{U}_{f^{-1}(1)}$ are k-PRGs for $\mathcal{M}$.

We will say that f is *k-hard for $\mathcal{M}$* in the first case, and that f is *k-pseudo-random for $\mathcal{M}$* in the second.[9] If f is k-pseudo-random for $\mathcal{M}$, then it is also k-hard for $\mathcal{M}$; the converse is not necessarily true, as follows from the same example of the equality function in $\mathcal{R}^{\|}$.

As usual in communication complexity, we will say that a communication problem is *easy* for a given model if it can be solved by a protocol of cost $(\log n)^{O(1)}$.

▶ **Definition 17** (Ultra-separation)**.** Complexity models $\mathcal{M}_1$ and $\mathcal{M}_2$ are ultra-separated if there is a total Boolean function f that is easy for $\mathcal{M}_1$ and $n^{\Omega(1)}$-pseudo-random for $\mathcal{M}_2$.

Ultra-separation is a very limited type of model separation – in fact, the most limited "reasonable" one we came up with.

▶ Claim 18. For any two models that allow efficient error reduction for total functions, ultra-separability implies separability both via a total function and via a PRG.

Here by efficient error reduction we mean that if f can be solved efficiently, then for any constant ε there exists an efficient protocol that solves f with error at most ε. Probably all studied communication complexity models satisfy this very natural property.

Proof. If f is $n^{\Omega(1)}$-pseudo-random for $\mathcal{M}_2$, then it is also $n^{\Omega(1)}$-hard for $\mathcal{M}_2$, and therefore ultra-separability implies separability via a total function.

If f is easy for $\mathcal{M}_1$, then the elements of $f^{-1}(1)$ can be distinguished from the elements of $f^{-1}(0)$ with worst-case error at most $1/10$ by a protocol of cost $(\log n)^{O(1)}$. Without loss of generality, let $\mathbf{Pr}\left[f(X,Y)=1\right] \leq 1/2$ when (X,Y) is uniformly random. Then there exist an efficient protocol in $\mathcal{M}_1$ that outputs "1" with probability at least $9/10$ when $(X,Y) \sim \mathcal{U}_{f^{-1}(1)}$, and with probability at most $11/20$ when (X,Y) is uniformly random. So, $\mathcal{U}_{f^{-1}(1)}$ can be distinguished from the uniform with "bias" more than $1/3$ by an efficient protocol in $\mathcal{M}_1$, and thus it is not a PRG. Therefore, ultra-separability implies separability via a PRG. ◀

5.3 Ultra-separation of $\mathcal{R}^{\|,pub}$ and $\mathcal{R}^{\leftrightarrow}$

We have seen that *ultra-separability of two models is a stronger evidence of difference in their computational power than separability via a function (total or partial), via a relation, or via a PRG*. We are not aware of any type of model separation that would not be subsumed by

[9] Note that we required both $\mathcal{U}_{f^{-1}(0)}$ and $\mathcal{U}_{f^{-1}(1)}$ to be k-PRGs for $\mathcal{M}$ when f is k-pseudo-random in order not to require f to be balanced; if it is balanced, either condition implies the other.

ultra-separation. Therefore, it is interesting to demonstrate ultra-separations even for those pairs of models that have been separated previously via some "less convincing" methods.

For long time, it had been believed that the models $\mathcal{R}^{\|,pub}$ and $\mathcal{R}^{\leftrightarrow}$ were equivalent. In 2002 Bar-Yossef, Jayram, Kumar and Sivakumar [4] demonstrated a separation between these models via a cleverly constructed total function g, for which $\mathcal{R}^{\leftrightarrow}(g) \in O(\log n)$ and $\mathcal{R}^{\|,pub}(g) \in \Omega(\sqrt{n})$. The ideas used in their construction seem to be insufficient to yield separation via a PRG.

▶ **Theorem 19.** *The models $\mathcal{R}^{\|,pub}$ and $\mathcal{R}^{\leftrightarrow}$ can be ultra-separated. Namely, there exists a total Boolean function f, such that $\mathcal{R}^{\leftrightarrow}(f) \in O(\log n)$ and $\mathcal{U}_{f^{-1}(1)}$ cannot be distinguished from $\mathcal{U}_{\{0,1\}^n \times \{0,1\}^n}$ by any $\mathcal{R}^{\|,pub}$-protocol of cost $o(n)$.*

The new separation is stronger not only qualitatively, but quantitatively as well – the improvement results from the (optimal) lower bound of $\Omega(n)$ on the $\mathcal{R}^{\|,pub}$-complexity of f.

Proof. From Corollary 9 it follows that for any constant δ there exists a graph G on 2^n vertices of degree $d \in \Theta(n)$, which is a $(2^{n/2}, \delta)$-partition expander. According to Theorem 15, the corresponding μ_G is $\Theta(n)$-pseudo-random for $\mathcal{R}^{\|}$. Clearly, the same is true for $\mu_{\bar{G}}$, where $\bar{G}$ is the complement graph. If we define $f_G : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ to be the "edge function" of G, then it is $\Theta(n)$-pseudo random for $\mathcal{R}^{\|}$.

Let us see that $\mathcal{R}^{\leftrightarrow}(f_G) \in O(\log d) = O(\log n)$. Consider a protocol where the players use shared randomness[10] to choose a hash function from $\{0,1\}^n$ to $\{0,1\}^{2\log d}$, then Alice sends the hash-value of x and Bob answers "1" if the received value equals the hash-value of one of the neighbors of y in G, and "0" otherwise (if Bob is the sender, they act symmetrically). This protocol has communication cost $O(d)$ and computes f_G with error $o(1)$. The result follows. ◀

6 Discussion

The most interesting open problem is to find an explicit construction of a good partition expander; more precisely, to construct a family of (K, δ)-partition expanders in which $\delta < 1$ is constant, K goes to infinity, and the degrees are $d = O(\log K)$. We will call informally such families of graphs *good partition expanders*. As we have shown in this paper, expanders are, in general, not good partition expanders and it seems unlikely that the property would be implied by a property of the spectrum of a graph. One possible way of constructing good partition expanders could be by using zig-zag product or a similar kind of product. Indeed, in a recent paper [13] Mendel and Naor have shown that zig-zag product can be used for constructing various types of generalizations of expanders. These constructions start with a small object, which can be found by brute force, and which are enlarged by applying products repeatedly. They work well when one needs constant degree, but in our case we need increasing degree and to satisfy a certain property for partitions with exponentially increasing number of blocks. It is not totally excluded that some kind of product will work, but it will require a new kind of argument to prove it.

We demonstrated some applications of partition expanders in communication complexity. In particular, we defined the notion of ultra-separation and argued that it is one of the weakest model-separating methods, thus applying it provides a very strong (probably, the strongest known) evidence that the two separated models have different computational power.

[10] The power of the model $\mathcal{R}^{\leftrightarrow}$ is not affected by allowing public randomness.

We gave an example of such separation. It would be interesting to find more examples of ultra-separations, not only in communication complexity.

We believe that partition expanders will be useful in many other areas of complexity theory, especially when explicit constructions are found. For example, one could use good partition expanders instead of expanders in the pseudorandom generators of Impagliazzo, Nisan and Wigderson [10], provided that an explicit construction of good partition expanders is found. Since the number of partitions corresponds to the exponential of space complexity, they would certainly have better parameters. This, however, requires further research, because the direct application of partition expanders in INW generators does not seem to give substantially better results than the use of expanders.

Acknowledgments. We thank Hartmut Klauck and anonymous reviewers for helpful comments.

References

1 N. Alon and Y. Roichman. Random Cayley Graphs and Expanders. *Random Structures and Algorithms 5*, pages 271–284, 1994.

2 N. Alon and J. Spencer. The Probabilistic Method. *John Wiley*, 2008.

3 K. Azuma. Weighted Sums of Certain Dependent Random Variables. *Tohoku Mathematical Journal 68*, pages 357–367, 1967.

4 Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. Information Theory Methods in Communication Complexity. *Proceedings of 17th IEEE Conference on Computational Complexity*, pages 93–102, 2002.

5 B. Bollobás. A Probabilistic Proof of an Asymptotic Formula for the Number of Labelled Regular Graphs. *European Journal of Combinatorics 1*, pages 311–316, 1980.

6 M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson. Randomness Conductors and Constant-Degree Lossless Expanders. *Proceedings of the 34th Symposium on Theory of Computing*, pages 659–668, 2002.

7 Z. Dvir and A. Wigderson. Monotone Expanders: Constructions and Applications. *Theory of Computing 6(1)*, pages 291–308, 2010.

8 D. Gavinsky, O. Regev, and R. de Wolf. Simultaneous Communication Protocols with Quantum and Classical Messages. *Chicago Journal of Theoretical Computer Science*, 7, 2008.

9 A. J. Hoffman and H. W. Wielandt. The Variation of the Spectrum of a Normal Matrix. *Duke Mathematical Journal 20*, pages 37–39, 1953.

10 R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for Network Algorithms. *Proceedings of the 26th Symposium on Theory of Computing*, pages 356–364, 1994.

11 Z. Landau and A. Russell. Random Cayley Graphs are Expanders: A Simple Proof of the Alon-Roichman Theorem. *Electronic Journal of Combinatorics 11*, 2004.

12 B. D. McKay and N. C. Wormald. Asymptotic Enumeration by Degree Sequence of Graphs with Degrees $o(n^{1/2})$. *Combinatorica 11(4)*, pages 369–382, 1991.

13 M. Mendel and A. Naor. Nonlinear Spectral Calculus and Super-Expanders. *Publications mathématiques de l'IHÉS*, 2013.

14 N. C. Wormald. Models of Random Regular Graphs. *Surveys in Combinatorics. Lecture Note Series 276*, pages 239–298, 1999.

Testing Generalised Freeness of Words*

Paweł Gawrychowski[1], Florin Manea[2], and Dirk Nowotka[2]

1 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
gawry@cs.uni.wroc.pl
2 **Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany**
{flm,dn}@informatik.uni-kiel.de

Abstract

Pseudo-repetitions are a natural generalisation of the classical notion of repetitions in sequences: they are the repeated concatenation of a word and its encoding under a certain morphism or antimorphism (anti-/morphism, for short). We approach the problem of deciding efficiently, for a word w and a literal anti-/morphism f, whether w contains an instance of a given pattern involving a variable x and its image under f, i.e., $f(x)$. Our results generalise both the problem of finding fixed repetitive structures (e.g., squares, cubes) inside a word and the problem of finding palindromic structures inside a word. For instance, we can detect efficiently a factor of the form xx^Rxxx^R, or any other pattern of such type. We also address the problem of testing efficiently, in the same setting, whether the word w contains an arbitrary pseudo-repetition of a given exponent.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Stringology, Pattern matching, Repetition, Pseudo-repetition

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.337

1 Introduction

A word is a *repetition* if it can be written as a repeated concatenation of one of its prefixes to itself. A word w is a *pseudo-repetition* if it can be written as a repeated concatenation of one of its prefixes t and its image $f(t)$ under some morphic or antimorphic function f (for short, an "anti-/morphism" f), thus $w \in t\{t, f(t)\}^+$.

The concept of pseudo-repetitions (introduced in [4]) draws its original motivations from two important biological concepts: tandem repeat, i.e., the consecutive repetition of the same sequence of nucleotides, and the inverted repeat, i.e., a sequence of nucleotides whose reversed complement (or, Watson-Crick complement) occurred already in the longer DNA sequence we analyse, both occurrences (the original one and the complemented one) encoding, essentially, the same genetic information. Noting that the Watson-Crick complement can be abstracted as an antimorphic involution on the DNA-alphabet, pseudo-repetitions formalise generalised tandem repeats, in which one sequence is followed by several consecutive occurrences of either its copy or of its reversed complement.

Other situation in which one encounters pseudo-repetitions appears in musical theory. The repetition of some fragment, in its initial form but also slightly modified (like the initial fragment on a higher or lower pitch), are used to provide unity to a musical piece. For instance, the *ternary (song) form* appears frequently: three consecutive musical fragments

* P. Gawrychowski is supported by the *NCN* grant 2011/01/D/ST6/07164, F. Manea by the *DFG* grant 596676, D. Nowotka by the *DFG Heisenberg* grant 590179.



© Paweł Gawrychowski, Florin Manea, and Dirk Nowotka;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 337–349



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



such that the first and third ones are identical, while the second one is constructed in order to provide a contrast to the other two (and, sometimes, this can be seen as the image of the other two parts under some simple anti-/morphism).

Note that both in biology and music, the function applied to obtain the pseudo-repetition is structurally simple: it usually rewrites each letter (nucleotide or note) into another one (i.e., it is literal), and usually does not rewrite more letters into the same one (i.e., it is bijective). Besides the two examples above, in which pseudo-repetitions occur, the concept seems to be of intrinsic theoretical interest, as it generalises a combinatorics on words concept (that is, repetitions) that is central both in theory and applications. If we consider palindromes as a natural modification of squares, repetitive structures containing both normal occurrences of some factor and mirrored occurrences of the same factor seem to be one of the most natural, hence, interesting, concepts derived from the classical repetitions.

The results obtained so far on pseudo-repetitions were both of combinatorial [4, 12, 13] and of algorithmic [6, 7, 14] nature. We continue here the study of algorithmic problems related to this concept. More precisely, we study how efficiently can one decide, given an input word w, a literal (in some cases, bijective) anti-/morphism f, and a pattern involving both a variable x and its image under f, namely $f(x)$, whether an input word contains as a factor an instance of that pattern. The problem seems natural to us, both in the light of the combinatorial questions on the avoidability of patterns under anti-/morphisms, raised in [13], as well as in the respect that it generalises both the well-studied problems of efficiently deciding whether a word contains k-repetitions or, alternatively, various palindromic structures.

The paper is structured as follows. We begin with a series of basic definitions, then we overview our results and compare them to the existing literature, and conclude with two technical sections, containing the proofs of our results.

Some Basic Concepts. For more detailed definitions we refer to [2].

Let V be a finite alphabet; V^* is the set of all words over V, and the *empty word* is denoted by λ. The *length* of a word $w \in V^*$ is denoted by $|w|$, while $\mathrm{alph}(w)$ denotes the set of all letters that occur in w. In our problems, we assume that the letters of any word w of length n are in fact integers from $\{1, \ldots, n\}$; accordingly, w is a sequence of integers. This is a common assumption in stringology (see, e.g., [8]).

If $w = xuy$, for $w, x, u, y \in V^*$, then u is a *factor* of w, x is a *prefix* of w, and y is a *suffix* of w. For $1 \leq i \leq j \leq |w|$, we denote by $w[i]$ the symbol at position i in w and by $w[i..j]$ the factor $w[i]w[i+1]\cdots w[j]$ of the word w. A word u occurs in w at position i if u is a prefix of $w[i..|w|]$. The powers of a word w are defined recursively by $w^0 = \lambda$ and $w^n = ww^{n-1}$ for $n \geq 1$. A word w that is not a power of some other word is called *primitive*. If w is not primitive, then there exists a unique primitive word u, called the primitive root of w, such that $w = u^n$ for some $n \geq 2$. A *period* of a word w over V is a positive integer p such that $w[i] = w[j]$ for all i and j with $i \equiv j \pmod{p}$; $per(w)$ denotes the smallest period of w.

A function $f : V^* \to V^*$ is a morphism if $f(xy) = f(x)f(y)$ for all $x, y \in V^*$; f is an antimorphism if $f(xy) = f(y)f(x)$ for all $x, y \in V^*$. To define an anti-/morphism it is enough to define $f(a)$, for all $a \in V$. We call f *literal* if $|f(a)| = 1$ for all $a \in V$. Let $f(V) = \{f(a) \mid a \in V\}$. We assume that any literal anti-/morphism f is given as the image of the letters of V under f (in order, from 1 to $|V|$).

We say that a word w is an *f-repetition* with root t or, alternatively, an f-power with root t, if w is in $\{t, f(t)\}^+$, for a factor t of w. If w is not an f-power, then w is *f-primitive*. For example, the word *abcaab* is primitive from the classical point of view (i.e., **1**-primitive, where **1** is the identical morphism) as well as g-primitive for the morphism g defined by

$g(a) = b$, $g(b) = a$, $g(c) = c$. However, when considering the morphism $f(a) = c$, $f(b) = a$, $f(c) = b$, we get that $abcaab = ab\, f(ab)\, ab$, thus, being an f-repetition with root ab.

For an anti-/morphism $f : V^* \to V^*$, a unary f-pattern p is an element of the set $\{x, f(x)\}^*$ (i.e., a word over the alphabet having the letters x and $f(x)$); here x is called variable and, if $p \in \{x, f(x)\}^k$, we say that k is the length of p. An instance of the f-pattern p is a word obtained by replacing in it the variable x by a word $t \in V^+$ and evaluating the resulting expression in V^*. For instance, if f is the identity antimorphism (i.e., f is $(\cdot)^R$, the mirror image) then $xf(x) = xx^R$ is a pattern whose instances are all palindromes of even length; if f is the identity morphism then $xf(x) = x^2$ is a pattern whose instances are all squares. We will only discuss the case of unary f-patterns, called patterns for brevity.

Finally, the computational model we use is the standard unit-cost RAM with logarithmic word size. Also, all logarithms appearing here are in base 2.

2 The problems

In [6], the problem of identifying, given a word w and an anti-/morphism f, all the f-repetitions (either of arbitrary or of given exponent) contained in w was efficiently solved. The solutions and their time complexities depended on the properties of f, but, in all cases, they were influenced by the fact that we needed to output *all* f-repetitions.

Here we approach two related problems. We are given a word w, an anti-/morphism f, just like before, but also a unary f-pattern p. We want to test whether w has as factor

an instance of the given pattern p (i.e., whether w is p-free). Further, in the same setting, but given a number k instead of the pattern, we are interested in testing whether the word w contains an f-repetition of exponent k, that is, a word of the form $\{t, f(t)\}^k$. This task was called in [1] the problem of testing pseudo-kth-power freeness. Compared to the first problem, this one requires deciding whether w contains an instance of *any* pattern of length k.

The results we present deal with the cases when f is a literal morphism or antimorphism; moreover, in part of the results we restrict f to being bijective. These cases seem to be interesting as they cover exactly the classes of morphisms and antimorphisms that play an important role in the literature: the classical mirror image of words, the antimorphic involutions used in the initial papers on pseudo-repetitions [1, 4, 14] to model the DNA Watson-Crick complementarity, or the anti-/morphic permutations used in [13] in the context of avoiding pseudo-repetitions or in [12] to show generalised periodicity lemmas.

The two problems are defined formally in the following.

▶ Problem 1. Given $w \in V^+$, $|w| = n$, $f : V^* \to V^*$ a literal anti-/morphism, and an f-pattern p of length k, decide whether there exists an instance of p occurring in w.

▶ Problem 2. Given $w \in V^+$, $|w| = n$, $f : V^* \to V^*$ a literal anti-/morphism, and an integer $k > 0$, decide whether there exists a factor v of w with $v \in \{t, f(t)\}^k$ for some $t \in V^+$.

Our results are the following. We first give, in Section 4, solutions to Problem 1 and 2 that run in $\mathcal{O}(nk^2)$ time when f is both a literal morphism or a literal antimorphism. This is especially interesting as it shows that Problem 1 can be solved in linear time for f-patterns of fixed length (or Problem 2 for constant k). For instance, the occurrence of patterns having length 2 (generalised squares) or 3 (cubes under literal anti-/morphisms), inside a word can be tested in linear time. The solutions we present here rely both on several combinatorics on words results and on the possibility of constructing efficient data structures for word processing. It is worth noting that our approach begins similarly to that used in [11] to detect classical repetitions; however, the arguments used further in our algorithms and proofs are more general and required both a deeper analysis and novel techniques. Moreover, the

tools we developed may have a broader range of applications. For instance, Lemmas 3, 4, or 7, provide novel insight in the combinatorics of pseudo-repetitions, while Lemma 2 shows how an extended Lempel-Ziv-like factorisation of a word can be efficiently computed, which might be useful in, e.g., efficiently detecting inverted repeats in DNA sequences.

In Section 5, we consider the problems for f bijective. As already described, this case played an important role in the existing theory and its motivations. In this setting we propose new solutions for Problems 1 and 2, running in $\mathcal{O}(n \log n)$ time. While being based on a different approach than the previous ones, these solution have also the nice feature that they are efficient even for larger values of k. Moreover, the solution of Problem 2 can be used to compute the maximum k such that w contains an instance of a pattern of length k.

Before concluding this section, let us note that our results improve significantly the results reported in [14], [1], and [6]. In [14] it was shown that factors of the form $t^{k-1}f(t)$ or $(tf(t))^k$ (and symmetrical) can be detected in $\mathcal{O}(kn)$ time; in [1] these results were extended to show that that pseudo-cube freeness can be tested in $\mathcal{O}(n^2)$ time. Both these results were developed for f an antimorphic involution and an alphabet of constant size. In [1], within the same setting, an algorithm outputting all the pseudo-k-powers contained in a word in $\mathcal{O}(n^2 \log n)$ time was reported, and used to test pseudo-kth-power freeness; a faster algorithm finding all the pseudo-k-powers occurring in a word in time $\Theta(n^2)$ was given in [6] (for f literal, and working over integer alphabets, as here), but it was still slower than our algorithms when used to test pseudo-kth-power freeness. For patterns without functional dependencies, Problems 1 and 2 coincide, and can be solved in linear time (see, e.g., [11]).

3 Prerequisites

For a word u, $|u| = n$, over $V \subseteq \{1, \ldots, n\}$ we can build in $\mathcal{O}(n)$ time the suffix tree and suffix array structures, as well as data structures allowing us to retrieve in constant time the length of the longest common prefix of any two suffixes $u[i..n]$ and $u[j..n]$ of u, denoted $LCP_u(i, j)$ (the subscript u is omitted when there is no danger of confusion). Such structures are called LCP data structures in the following. For details, see, e.g., [8], and the references therein. Similarly, we can build structures allowing us to retrieve in constant time the length of the longest common suffix of any two prefixes $u[1..i]$ and $u[1..j]$ of u, denoted $LCS_u(i, j)$.

▶ Remark 1. Given a word w of length n and a divisor ℓ of n we can use one LCP query to check in constant time whether $w = x^k$, where $|x| = \ell$ and $k = \frac{n}{\ell}$. Indeed, $w = x^k$ if and only if $LCP(1, \ell + 1) = n - \ell$. The longest prefix of w that is a power of $u[1..\ell]$ is obtained similarly, as the longest prefix w' of w whose length is divisible by ℓ and $|w'| \leq LCP(1, \ell + 1)$.

When solving Problems 1 and 2 we construct LCP data structures for the words w, w^R (the mirror image of w), and $v = wf(w)$; this takes $\mathcal{O}(|w|)$ time. Note that, when f is a literal morphism, checking whether $f(w[i..j])$ appears at position ℓ in w is equivalent to checking whether $\ell + j - i \leq n$ and $LCP_v(\ell, |w| + i) \geq j - i + 1$. When f is a literal antimorphism, checking whether $f(w[i..j])$ appears at position ℓ in w is, in this case, equivalent to checking whether $\ell + j - i \leq n$ and $LCP_v(\ell, 2|w| - j + 1)) \geq j - i + 1$.

In some of the proofs we will need an efficient solution for the *interval union-find* problem, which asks to maintain a partition of the universe $U = [1, n]$ into a number of disjoint intervals, so that given any element we can locate its current interval, and we can merge two currently adjacent intervals into one. Both operations can be implemented in amortised constant time [5] in our model of computation.

The following classical combinatorial result is used in this paper; for proofs and details see [10], the handbook [2, Chapter 9], and the references therein.

▶ **Lemma 1.** *Let $w \in V^*$, with $|w| = n$, and $PS_w = \{u \text{ primitive} \mid u^2 \text{ prefix of } w\}$. Then $|PS_w| \leq 2 \log n$ and one can compute all the sets $PS_{w[i..n]}$, for $1 \leq i \leq n$, in $\mathcal{O}(n \log n)$ time.*

4 General solutions

The main result we show in this section is that both our problems can be solved in $\mathcal{O}(nk^2)$ time, so in linear time for constant k. That is, testing freeness with respect to a fixed f-pattern or testing pseudo-kth-power freeness for constant k can be done in linear time. These results seem highly interesting to us as they show that the efficiency of testing square, cube, or palindrome freeness is preserved for a class of more general patterns.

We begin this section with a slightly modified version of the s-factorisation defined in [11] (see also [2]), and series of lemmas on the newly defined concept. Let $g : V^* \to V^*$ be a literal anti-/morphism and $w \in V^*$ a word. The g-factorisation of w is defined as follows. We factor $w = u_1 \cdots u_r$ if the following hold for all $i \geq 1$:

- If letter a occurs in w immediately after $u_1 \cdots u_{i-1}$ and neither a nor $g(a)$ appeared in $u_1 \cdots u_{i-1}$, then $u_i = a$.
- Otherwise, u_i is the longest word such that $u_1 \cdots u_{i-1}u_i$ is a prefix of w and u_i or $g(u_i)$ occurs at least once as a factor in $u_1 \cdots u_{i-1}$.

The **1**-factorisation of w (the not-self-referential variant of the s-factorisation from [11]) is obtained by just taking g to be **1**, the identity morphism.

By arguments similar to those used in [3], it follows that g-factorisations of words can be computed in linear time, for g literal anti-/morphism.

▶ **Lemma 2.** *If g is a literal anti-/morphism we can compute the g-factorisation of a word w of length n in time $\mathcal{O}(n)$.*

The following combinatorial lemma shows the relation between possible occurrences of an f-pattern p in a word and its f-factorisation, for a morphism f.

▶ **Lemma 3.** *Let f be a literal morphism, w a word, and p a pattern of length $k \geq 2$, such that $p \neq x^{k-1}f(x)$. Let $w = u_1 \cdots u_r$ be the f-factorisation of w and consider all instances of p. Then for any instance $w[i..j]$ with $|u_1 \cdots u_{h-1}| < j \leq |u_1 \cdots u_h|$ we have two mutually exclusive possibilities:*
1. *$i > |u_1 \cdots u_{h-1}|$, and we call $w[i..j]$ a* secondary *instance, completely contained in u_h,*
2. *$j - i + 1 \leq k(|u_{h-1}| + |u_h|)$, and we call $w[i..j]$ a* crossing *instance.*
Furthermore, the leftmost instance of the pattern is crossing.

A related result can be shown also for f antimorphic, but in a slightly more particular setting: the word w is **1**-factored and the pattern p has at least length 3.

▶ **Lemma 4.** *Let f be a literal anti-/morphism, w a word, and p a pattern of length $k \geq 3$, such that $p \notin \{x^{k-1}f(x), f(x)^{k-1}x\}$. Let $w = u_1 \cdots u_r$ be the **1**-factorisation of w and consider all instances of the pattern p. Then for any such instance $w[i..j]$ with $|u_1 \cdots u_{h-1}| < j \leq |u_1 \cdots u_h|$ we have two mutually exclusive possibilities:*
1. *$i > |u_1 \cdots u_{h-1}|$, and $w[i..j]$ is a secondary instance, completely contained in u_h,*
2. *$j - i + 1 \leq k(|u_{h-1}| + |u_h|)$, and $w[i..j]$ is a crossing instance.*
Furthermore, the leftmost instance of the pattern is crossing.

We only present here the solutions of Problems 1 and 2 for f antimorphic. The morphic case can be solved in a similar manner with less technicalities. For Problem 1, for instance, we compute the f-factorisation of the input word w, and then we try to identify (like in

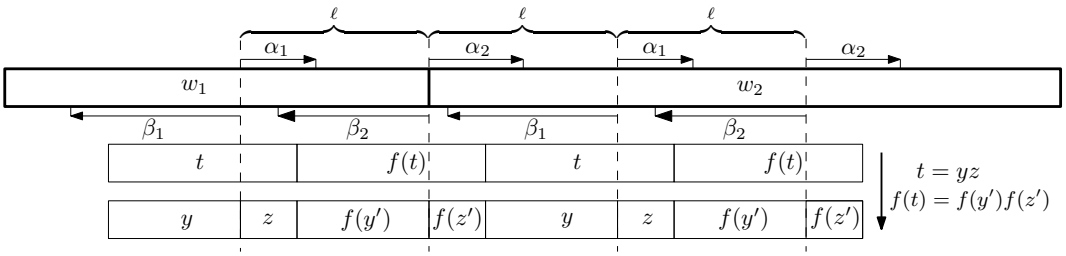


Figure 1 Finding $tf(t)tf(t)$ in the catenation of two words.

the antimorphic case, described below) the leftmost occurrence, if any, of an instance of the pattern p, which is crossing by Lemma 3. The usage of f-factorisations is needed here so that the case of patterns of length 2 (or f-squares) is also covered. The time complexity of this approach is $\mathcal{O}(nk^2)$.

The case of Problem 1, for antimorphisms. When f is an antimorphism we use **1**-factorisations of the input words, instead of f-factorisations. The major points of the algorithm detecting instances of the pattern f are given in the following.

In the following, a pseudopalindrome of length ℓ is a word of the form $uf(u)$ with $|u| = \ell$. Its occurrence is centred at position i if the first character of $f(u)$ is aligned there. The pseudopalindromic radius at i is simply the length of the longest pseudopalindrome centred at i, which can be computed in constant time using LCS queries on $wf(w)$.

To begin with, patterns of the form $xf(x)$ or $f(x)x$ are detected in $\mathcal{O}(n)$ time by checking the existence of positions with strictly positive pseudopalindromic radius in w and, respectively, w^R. Moreover, if the pattern is x^k, the problem reduces to detecting the usual repetitions, hence can be done in $\mathcal{O}(n)$ time. If the pattern is $f(x)^k$, we just need to check which letters can be an image under f, so which factors of w can contain an instance of the pattern, and again we can reduce the problem to the standard case of detecting repetitions.

▶ **Lemma 5.** *Let f be a literal antimorphism, w be a* **1***-factorised word, and p a pattern of length $k \geq 3$, $p \notin \{x^{k-1}f(x), f(x)x^{k-1}, f(x)^{k-1}x, xf(x)^{k-1}, x^k, f(x)^k\}$. All M crossing instances of p (if any) can be detected (and output as pairs of indices) in $\mathcal{O}(nk^2 + M)$ time.*

Proof. The proof is based on Lemma 4. Assume that the **1**-factorisation of w is $u_1 \cdots u_r$ and for each $h \leq r$ we look for an instance of p ending inside u_h, shorter than $k(|u_{h-1}| + |u_h|)$.

We describe how to find an instance v of a given pattern p of length k, in the concatenation of two words w_1 and w_2, such that this occurrence crosses the boundary; then we can apply the strategy for $w_2 = u_h$ and w_1 a suffix of $u_1 \cdots u_{h-1}$, $|w_1| = \min\{|u_1 \cdots u_{h-1}|, k(|u_{h-1}u_h|) - 1\}$. For each such w_1 and w_2 we construct a list of simple conditions which guarantee the existence of an occurrence. Then we consider all such conditions together, and verify them in a specific order. For simplicity, we only explain the case of $v = tf(t)tf(t)$ (an instance of $p = xf(x)xf(x)$), with $k = 4$, and then show how to generalise. Let us also assume that the first letter of w_2 belongs to the first $f(t)$, see Figure 1; the other cases are analogous. Let ℓ denote $|t|$ and assume that we built LCP and LCS data structures for $w' = w_1w_2f(w_1)f(w_2)$.

Now, let α_1 be the length of the longest factor starting at positions $|w_1| - \ell + 1$ in w_1 and ℓ in w_2; let α_2 be the length of the longest factor starting at positions 1 and $2\ell + 1$ in w_2. Similarly, let β_1 be the length of the longest factor ending at positions $|w_1| - \ell$ in w_1 and $\ell - 1$ in w_2, and β_2 be the length of the longest factor ending at positions $|w_1|$ in w_1 and 2ℓ in w_2. All $\alpha_1, \alpha_2, \beta_1, \beta_2$ can be computed with $LCP_{w'}$ and $LCS_{w'}$ queries. Now,

if $\min\{\alpha_1, \alpha_2\} + \min\{\beta_1, \beta_2\} < \ell$, clearly no instance exists. Otherwise, the only possible instances have $|z| \leq \min\{\alpha_1, \alpha_2\}$ and $|z| \geq \ell - \min\{\beta_1, \beta_2\}$ (where z is defined as in Figure 1). Moreover, those two conditions guarantee that all fragments corresponding to t are the same, and all fragments corresponding to $f(t)$ are equal, too. They do not guarantee, though, that the fragment which is supposed to be $f(t)$ is indeed an image of the fragment corresponding to t. Hence we also need to check if the pseudopalindromic radius at $|w_1| - \ell + |z|$ in w_1w_2 is at least ℓ. That is, an instance of the pattern corresponds exactly to a pseudopalindrome of length ℓ centred at $i \in [|w_1| - \min\{\beta_1, \beta_2\}, |w_1| - \ell + \min\{\alpha_1, \alpha_2\}]$ in w_1w_2, if any.

We build a list of all such conditions corresponding to different values of $\ell \leq \frac{|w_1|+|w_2|}{k}$ (and different pairs of concatenated words w_1 and $w_2 = u_h$, as described in the beginning) and then process them all at once in the order of increasing ℓ. This is done efficiently by maintaining in a structure S all positions i such that the pseudopalindromic radius at i is at least the current ℓ. Then, reporting all instances corresponding to a single condition requires iterating through all $i \in S$ such that i is in the corresponding range. Note now that S can be implemented as the interval union-find structure. Then, if the range contains H numbers, we output them in $\mathcal{O}(1 + H)$ time. We check similarly all the other possibilities for a factor t or $f(t)$ to fall on the border, so the claimed complexity (for $p = xf(x)xf(x)$) follows.

Other patterns of length 3 and 4 are analysed similarly. If the pattern is longer (of length $k > 4$), we check k possibilities for the factor falling on the border. For each of them we need to execute $\mathcal{O}(k)$ constant time queries to generate the conditions on $|y|$ or detect that no such instance is possible. Hence the total number of conditions is $\sum_{2 \leq h \leq r} \sum_{\ell \leq |u_{h-1}|+|u_h|} k = \mathcal{O}(nk)$, and the total time to generate all of them is $\mathcal{O}(nk^2)$, plus additional $\mathcal{O}(n)$ to maintain the interval union-find structure. Then each instance is reported in constant time. ◀

Assume first that $p \notin \{x^{k-1}f(x), f(x)x^{k-1}, f(x)^{k-1}x, xf(x)^{k-1}, x^k, f^k(x)\}$. Lemma 4 shows that if p occurs in w, then there exists a crossing instance of p in w. The previous lemma shows that we can locate in $\mathcal{O}(nk^2)$ time such a crossing instance of p, if any (just output the first instance we meet and stop searching for others). If we found one, we conclude that w contains an instance of the pattern; otherwise, w does not contain any instance of p.

Further, we only have to consider now the cases when p is $x^{k-1}f(x)$ or $xf(x)^{k-1}$; the other cases can be reduced to these two by looking for the occurrences of p^R in w^R. In the first (respectively, second) remaining case, we only need to check if there is a position i such that the pseudopalindromic radius at i is at least as long as the length of the shortest word whose k-th power is a suffix (respectively, prefix) of $w[1..i-1]$ (respectively, $w[i..n]$). Thus, to conclude, we apply the following lemma for w (respectively, w^R).

▶ **Lemma 6.** *Given a word w of length n and $k \geq 2$, we can compute for each position i the smallest $\ell \geq 1$ such that $w[i - k\ell + 1..i]$ is a power of $w[i - \ell + 1..i]$, in $\mathcal{O}(n)$ total time.*

This technical lemma and its main consequence, that we can detect instances of the patterns $x^{k-1}f(x)$ and $f(x)^{k-1}x$ in $\mathcal{O}(n)$ time, improves significantly the results reported in [14]: we decreased the complexity from $\mathcal{O}(nk)$, and our algorithm works for integer alphabets.

The case of Problem 2, for antimorphisms. Let us first give the following lemma:

▶ **Lemma 7.** *If w contains $\max(k, 3)$ pseudopalindromes of length ℓ starting at positions $s, s + \delta_1, s + \delta_2, \ldots$ with all $\delta_i \leq \frac{\ell}{4}$, then w has a factor r^k with $r = f(r)$. Accordingly, w contains an instance of any pattern of length k.*

To solve Problem 2, we begin with checking if there is an instance of x^k, $f^k(x)$, $f(x)^{k-1}x$, $xf(x)^{k-1}$, $x^{k-1}f(x)$, or $f(x)x^{k-1}$ using the method from the previous section. If there is

no such instance, we apply the reasoning described in Lemma 5. Of course, now we do not know the exact structure of the pattern. Nevertheless, we can look at the **1**-factorisation $w = u_1 \cdots u_r$, and for each two adjacent factors u_{h-1} and u_h we consider all possibilities for the length ℓ of t, the image of the variable x. Assume that for each such possibility we get that an instance of the pattern corresponds to the existence of a pseudopalindrome of length ℓ centred in a range of at most $2\ell k$ positions, these positions being the suffix of w_1 of length ℓk and the prefix of w_2 of length ℓk, with w_1 and $w_2 = u_h$ defined as in the proof of Lemma 5. We generate all pseudopalindromes in such ranges using the same approach as in that lemma, i.e., by considering the entire set of conditions at once and maintaining an interval union-find structure. If we can generate sufficiently many results, we terminate, as Lemma 7 shows that w contains instances of all patterns of length k. More concretely, for each range of length $2\ell k$ there can be at most $8k^2$ pseudopalindromes of length ℓ centred there, provided that no word r^k with $r = f(r)$ exists in w. Also, $\ell \leq |u_{h-1}| + |u_h|$ (by Lemma 4). So, summing up over all h and ℓ we get that the total number of generated pseudopalindromes should not exceed $\sum_{2 \leq h \leq r} \sum_{\ell \leq |u_{h-1}|+|u_h|} 8k^2 = 16nk^2$. Thus, assume that we generated at most $16nk^2$ pseudopalindromes. Each of them corresponds to certain values of ℓ and i such that the instance of $xf(x)$ is centred at i and the image of x has length ℓ. So, we can check in $\mathcal{O}(k)$ time whether the image of $xf(x)$ can be extended to an instance of a pattern of length k. For this we do not need to know the exact structure of this pattern, we just check that how many fragments of length ℓ are the same as either the left or the right half of the found pseudopalindrome with LCP queries. This gives an $\mathcal{O}(nk^3)$ time solution.

We can shave off one factor of k by using dynamic programming. We consider all generated pseudopalindromes with the same value of ℓ at once. For each such pseudopalindrome $vf(v)$, we compute the largest k' such that there is a k-repetition with $x = v$ starting with this $vf(v)$. The idea is that after any $vf(v)$ we must have a power of v or $f(v)$ followed by another $vf(v)$ (or by the end of the instance of the pattern). Hence with a constant number of LCP queries we can compute the highest power of both v and $f(v)$ following this $vf(v)$, and then check if the next fragment of length 2ℓ is $vf(v)$ as well. Then the total running time becomes proportional to the number of generated pseudopalindromes, which is $\mathcal{O}(nk^2)$.

5 The bijective case

An $\mathcal{O}(n \log n)$ solution for Problem 1. The second solution we propose for Problem 1 is based on a careful analysis of the length 3 factors that may occur in the pattern, and is valid, in this case, both for f literal bijective morphism and antimorphism.

We first check whether the pattern p contains any of the factors $xxf(x)$, $f(x)f(x)x$, $f(x)xx$, or $xf(x)f(x)$. This takes $\mathcal{O}(k) \subset \mathcal{O}(n)$ time. If not, then the pattern is a prefix of x^∞, $f(x)^\infty$, $(xf(x))^\infty$, or of $(f(x)x)^\infty$. We treat separately each of these cases.

First, let us note that if p is a prefix of x^∞ then Problem 1 is equivalent to testing whether w contains k-repetitions. This can be done in $\mathcal{O}(n)$ time, using the algorithm given by Main [11]. When p is a prefix of $f(x)^\infty$ the Problem 1 can be solved similarly, in linear time, by detecting repetitions in the factors of w containing only letters from $f(V)$.

▶ **Lemma 8.** *Testing whether a word w, $|w| = n$, has as factor an instance of a pattern p, $|p| = k$, that contains $xxf(x)$, $f(x)f(x)x$, $f(x)xx$, or $xf(x)f(x)$, can be done in $\mathcal{O}(kn \log n)$ time.*

Proof. We analyse now the cases when p contains $xxf(x)$ or $f(x)f(x)x$ (the cases when p contains $f(x)xx$ or $xf(x)f(x)$ are solved by considering the reversed word and reversed pattern). Hence, any instance of p contains a square. We analyse the first occurrence of a

factor $xxf(x)$ or $f(x)f(x)x$ in p and check whether it is possible that such a factor occurs at position i in the word w, for all $i \in \{1, \ldots, n\}$. To this end, we consider such a number i; by Lemma 1, there are at most $2 \log n$ primitive squares occurring at position i in w. Clearly, when $vvf(v)$ is a word that occurs at position i then v is a power of some $y \in PS_{w[i..n]}$. Thus, we iterate through all the elements of $y \in PS_{w[i..n]}$ and see whether we can construct a word v such that $vvf(v)$ or $f(v)f(v)v$ occurs at position i in w and v is a power of y.

Let us assume that $xxf(x)$ occurs before any occurrence of $f(x)f(x)x$ in p (the other case can be treated similarly). We have two subcases to analyse. The first subcase is when $y = f(y)$ (and this can be checked in constant time). Then we compute the maximum q such that y^q is a prefix of $w[i..n]$; this takes $\mathcal{O}(1)$ time, as $q = \frac{LCP_w(i,i+|y|)}{|y|} + 1$. Similarly, using an LCP_{w^R} query, we compute the maximum ℓ such that y^ℓ is a suffix of $w[1..i]$. If $q \geq 3$ we take $v = y$ and, clearly, an instance of the pattern p occurs in w whenever $\ell + q \geq k$, where $k = |p|$. The second subcase is when $y \neq f(y)$. Just like before, we compute the maximum q such that y^q is a prefix of $w[i..n]$. Now, if q is even and $f(y)$ occurs at position $1 + q|y|$, we have $v = y^{\frac{q}{2}}$; if q is odd then v cannot be a power of y. Once we know v we check whether an instance of p occurs in w such that its first factor $xxf(x)$ is mapped to the factor $vvf(v)$ occurring at position i in w. This check is done in $\mathcal{O}(k)$ time, using $LCP_{wf(w)}$ queries.

If $f(x)f(x)x$ occurs first in p (before $xxf(x)$) then one should also analyse, for each y, two subcases just like before. If $f(v)f(v)v$ occurs at position i then $f(v)$ is a power of some $y \in PS_{w[i..n]}$. The first subcase is when $y = f(y)$, and can be treated just as the case $y = f(y)$ above. The second subcase is when we assume that $y = f(z)$ for some $z \neq y$. As f is injective, we have $y \neq f(y)$. Thus, z is the first factor of length $|y|$ that occurs after the maximal prefix y^p of $w[i..n]$. We then check by an $LCP_{wf(w)}$ query whether $f(z) = y$ and p is even, and follow the same procedure as before, for v starting with z and of length $\frac{p|y|}{2}$. ◀

In conclusion, the case when p contains an occurrence of the factor $xxf(x)$ or of $f(x)f(x)x$ can be solved in $\mathcal{O}(kn \log n)$ time. By similar arguments, the case when p is a prefix of $(xf(x))^\infty$ is analysed faster, in $\mathcal{O}(n \log n)$ time.

Altogether, the analysis of the above cases takes $\mathcal{O}(kn \log n)$, where the most time-consuming step is the case when p contains at least one factor $xxf(x)$, $f(x)f(x)x$, $f(x)xx$, or $xf(x)f(x)$, and the word v, to which x is mapped, is different from $f(v)$. In the following, we show how we can reduce the time needed to analyse this case to $\mathcal{O}(n \log n)$ (that is, shave off the k factor), and, consequently, obtain that Problem 1 can be solved in $\mathcal{O}(n \log n)$ time.

▶ **Lemma 9.** *Testing whether a word w, $|w| = n$, has as factor an instance of a pattern p, $|p| = k$, that contains $xxf(x)$, $f(x)f(x)x$, $f(x)xx$, or $xf(x)f(x)$, can be done in $\mathcal{O}(n \log n)$ time.*

Proof. The approach in Lemma 8 can be optimised as follows. Instead of iterating through all positions where a factor of the form $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ or $vf(v)f(v)$ occurs, and then verifying in $\mathcal{O}(k)$ time if a given occurrence can be extended to form an occurrence of the whole pattern, we look at entire groups of such factors at once, and adapt the Knuth-Morris-Pratt algorithm (see [9]) to verify all of them at once. To make this verification efficient, we will run the pattern matching algorithm not on the original word, but on its suitably constructed compressed representation.

Using the same strategy as in the initial analysis of this case, we identify all the factors $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$ with $v \neq f(v)$ in w. We saw above that there are $\mathcal{O}(n \log n)$ occurrences of such factors. Note that a factor $vvf(v)$ (or a factor $vf(v)f(v)$) can be also identified as a factor $f(v')f(v')v'$ (respectively, as a factor $f(v')v'v'$), but only when $f(f(a)) = a$ for all the letters a occurring in v.

Then we group together the occurrences of $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$ factors having the same length. This can be done in $\mathcal{O}(n \log n)$ time by describing each such factor as a pair containing the position where it starts and the length of $|v|$, and then putting together in a set (ordered with respect to the starting position) all the factors with v of a certain length. As $|v| \leq n$ we will have $\mathcal{O}(n)$ such sets. Furthermore, we additionally partition each set into smaller sets (again, ordered with respect to the starting position) according to the remainder of the starting position modulo the length of v, which takes linear time in the size of the set. Finally, each such set is split into even smaller groups: we put together all the consecutive elements in the (ordered) set, that have the form $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$ for a certain (the same for all factors) word v. As we have already mentioned, a factor $vvf(v)$ (or a factor $vf(v)f(v)$) can be also identified as a factor $f(v')f(v')v'$ (respectively, as a factor $f(v')v'v'$), whenever $v' = f(v) \neq v$; however, the group corresponding to v is exactly the same one as the one constructed for v', so we do not need to consider them separately. Note that, in the end, there may be more than one group corresponding to the same v; however, one occurrence of $vvf(v)$ (as well as one occurrence of $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$) belongs to exactly one group.

The idea of this splitting into groups is that an instance of the pattern p, with x mapped to v, should have all the occurrences of factors $xxf(x)$, $f(x)f(x)x$, $xf(x)f(x)$, or $f(x)xx$ mapped to elements of the same group, that corresponds to v. Hence, in what follows we fix a single group (and, consequently, a word v) and show how to detect a corresponding instance of the pattern, with x mapped to v, in linear time in the size of the group. In particular, if f acts as an involution on the letters of v, we should also try to detect a corresponding instance of p, with x mapped to $f(v)$; but this is done analogously, and takes the same time.

We additionally partition the fixed group into subgroups. Each subgroup is a maximal set of consecutive elements of the group such that between any two of them we either have a power of either v or $f(v)$ (note that two consecutive elements, say $vvf(v)$ and $vf(v)f(v)$, might overlap, and in such case there is nothing between them). The partitioning requires just a single left-to-right scan of the elements of the group, with a constant number of LCP queries when moving from one element to the other. Now a single subgroup corresponds to a factor of w of the form $\{v, f(v)\}^+$, and we have an ordered list of all occurrences of $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$ in this factor. Furthermore, the factor starts and ends with factors of this form, so we call it v-delimited. We can represent an v-delimited word in a unique compressed form, called v-representation, as follows.

Sweep through all occurrences of $vvf(v)$, $f(v)f(v)v$, $f(v)vv$ and $vf(v)f(v)$. For each of them append $xxf(x)$, $f(x)f(x)x$, $f(x)xx$, or $xf(x)f(x)$, respectively, to the current v-representation. Then, for each such two adjacent occurrences additionally insert one of the following between the corresponding elements of the current representation: -2 if the occurrences overlap by $2|v|$; -1 if the occurrences overlap by $|v|$; 0 if the occurrences are one after another; (x, h) if the fragment between the occurrences is v^h with $h > 0$; $(f(x), h)$ if the fragment between the occurrences is $f(v)^h$ with $h > 0$.

▶ Example 1. If $v = a$, $f(v) = b$, and the word is $aabbaaaaabbbba$, its v-representation is $[xxf(x), -1, f(x)f(x)x, (x, 2), xxf(x), (f(x), 1), f(x)f(x)x]$.

Finally, treat each element of the representation as a single character (over a new alphabet, whose size is $\mathcal{O}(n^c)$ for some constant c, so the characters can be compared in constant time), and the whole representation as a single word. It is clear that above definition guarantees that the v-representation is unique. Furthermore, given a subgroup we can construct its v-representation in linear time (in the size of the subgroup). Similarly, we can locate the first and the last occurrence of $xxf(x)$, $f(x)f(x)x$, $f(x)xx$ or $xf(x)f(x)$ in the pattern and

construct in linear time (in the length of the pattern) the representation of its x-delimited factor starting at the first and ending at the last of such occurrences. Finally, observe that an occurrence of the whole pattern with x mapped to v corresponds to an occurrence of such maximal v-delimited factor, and furthermore given the latter we can verify in constant time if it corresponds to the former using a few LCP queries (i.e., the maximal v-delimited factor can be padded with powers of v or $f(v)$, to get an actual instance of the pattern p).

Hence, we can focus on generating all instances of the maximal x-delimited factor of the pattern, and this is what we designed the x-representation for. We simply apply the usual Knuth-Morris-Pratt algorithm to locate all occurrences of the x-representation of the maximal x-delimited factor of the pattern in the v-representation of the factor corresponding to the subgroup. This takes linear time in the size of the subgroup, excluding the preprocessing, and the number of occurrences is linear, too. Finally, note that the preprocessing of the pattern is done for all subgroups just once. ◀

This approach clearly identifies an instance, if any, of a pattern p that contains one of the factors $xxf(x)$, $f(x)xx$, $xf(x)f(x)$, or $f(x)f(x)x$, in time $\mathcal{O}(n \log n)$. In conclusion, by this last argument and the previous remarks, we obtained an $\mathcal{O}(n \log n)$ solution for Problem 1.

An $\mathcal{O}(n \log n)$ solution for Problem 2. In this case we want to test whether w contains the image of any pattern p from $\{x, f(x)\}^k$, for a given k. However, we will determine the maximum ℓ such that w contains an instance of some pattern $p \in \{x, f(x)\}^\ell$. Let us note that for $\ell \leq 3$ we have a linear time solution for testing whether w contains an f-repetition of exponent ℓ, as described in Section 4. Hence we focus on the remaining case $\ell \geq 4$.

The key remark in our approach is that if w contains an instance of a pattern $p \in \{x, f(x)\}^{k_1}$ then it also contains an instance of a pattern $p' \in \{x, f(x)\}^{k_2}$ where x is replaced by a primitive word v and $k_2 \geq k_1$. Hence, we identify first the maximal factors of w that have the form $\{t, f(t)\}^*$, for some primitive word t (thus, with $f(t)$ primitive, as well), and contain either tt or $f(t)f(t)$, and, then, the ones that do not contain such squares.

This is done similarly to the subgroup splitting of the previous section. We first locate the occurrences of such factors, in $\mathcal{O}(n \log n)$ time, and then refine this set to obtain the subgroups containing the occurrences of factors defined by the same t which have between them only elements from $\{t, f(t)\}^*$. Like before, this takes $\mathcal{O}(n \log n)$ time.

So far, we obtained the maximal factors of w that have the form $\{t, f(t)\}^*$ and start either with tt or $f(t)f(t)$. For such a factor y that starts with tt we compute in constant time the maximal factor z that ends just before y and has the form $(t\{f(t)t\}^* \cup \{f(t)t\}^*)f(t)$, following Remark 1. Then zy is a maximal factor that contains tt. Similarly, for a factor $y \in f(t)f(t)\{t, f(t)\}^*$ we compute the maximal factor z that ends just before y and has the form $(f(t)\{tf(t)\}^* \cup \{tf(t)\}^*)t$. Then zy is a maximal factor that contains $f(t)f(t)$. Consequently, we computed the maximal factors of w that have the form $\{t, f(t)\}^*$ and contain either tt or $f(t)f(t)$. Clearly, the time complexity of this procedure is $\mathcal{O}(n \log n)$.

Now, if at least one of the maximal factors of w of the form $\{t, f(t)\}^*$ and containing tt or $f(t)f(t)$ is an f-power of t having the exponent at least k, then we can answer Problem 2 positively. Otherwise, we need to check whether w contains a repetition of the form $\{tf(t)\}^\ell$, $\{tf(t)\}^\ell t$, $\{f(t)t\}^\ell$, or $\{f(t)t\}^\ell f(t)$, for a large enough exponent $\ell \geq 2$. But, when looking at maximal f-repetitions of this type, we get that their prefix of length $2|t|$ (that is, $tf(t)$ or $f(t)t$) should be primitive; otherwise, longer repetitions were detected in the previous step. Hence, such repetitions start with a primitively rooted square. Now, analysing all the primitively rooted squares occurring in w, and mapping them to $t(f)tf(t)$ or $f(t)tf(t)t$, depending on

the form pattern we look for (from the ones above), we can detect all these repetitions in $\mathcal{O}(n \log n)$ time, too. In conclusion, we solve completely Problem 2 in $\mathcal{O}(n \log n)$ time.

Using this strategy, we also identify, in $\mathcal{O}(n \log n)$ time, all the factors t and v of w such that t is primitive and $v \in \{t, f(t)\}^*$, and v cannot be extended, in any direction, by neither t nor $f(t)$. It is not hard to see that one of these f-repetitions is the one that has the maximum exponent among all the f-repetitions contained in w. Thus, we can also find the maximum power of an f-repetition contained in w, within $\mathcal{O}(n \log n)$ time.

6 Conclusions

In this paper we showed the following theorems.

▶ **Theorem 10.** *Given a word $w \in V^*$, with $|w| = n$, a literal anti-/morphism $f : V^* \to V^*$, and an f-pattern p of length k, we can decide whether w contains an instance of p in $\mathcal{O}(nk^2)$ time; for a fixed pattern p, the problem can be solved in linear time. If f is bijective, then the problem can be solved in $\mathcal{O}(n \log n)$ time.*

▶ **Theorem 11.** *Given a word $w \in V^*$, with $|w| = n$, a literal anti-/morphism $f : V^* \to V^*$, and a positive integer k, we can decide whether w contains a factor of the form $\{t, f(t)\}^k$, for some word t, in $\mathcal{O}(nk^2)$ time; for a constant k, the problem can be solved in linear time. If f is bijective, then we can compute the maximum k such that w contains a factor of the form $\{t, f(t)\}^k$, for some word t, in $\mathcal{O}(n \log n)$ time.*

We conjecture that results in the line of the second parts of our theorems (and similar proofs) hold also for general literal morphisms. In this case, however, one has to overcome the difficulty that when f is not bijective, then $f(t)$ is no longer primitive for all primitive t; accordingly, the technicalities are expected to be far more involved. Consequently, the time bounds are expected to be larger (although still close to linear time).

The main question left open is whether the results reported here can be improved to find (if there exist) algorithmic solutions for the approached problems running in $\mathcal{O}(n)$ time.

References

1. E. Chiniforooshan, L. Kari, and Z. Xu. Pseudopower avoidance. *Fundamenta Informaticae*, 114(1):55–72, 2012.
2. M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.
3. Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Wojciech Rytter, and Tomasz Walen. Efficient algorithms for two extensions of lpf table: The power of suffix arrays. In *Proc. SOFSEM*, volume 5901 of *LNCS*, pages 296–307, 2010.
4. E. Czeizler, L. Kari, and S. Seki. On a special class of primitive words. *Theoretical Computer Science*, 411:617–630, 2010.
5. H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proc. STOC*, pages 246–251. ACM, 1983.
6. P. Gawrychowski, F. Manea, R. Mercas, D. Nowotka, and C. Tiseanu. Finding Pseudo-repetitions. In *Proc. STACS*, volume 20 of *LIPIcs vol. 20*, pages 257–268, 2013.
7. P. Gawrychowski, F. Manea, and D. Nowotka. Discovering hidden repetitions in words. In *Proc. CiE*, volume 7921 of *LNCS*, pages 210–219. Springer, 2013.
8. J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *J. ACM*, 53:918–936, 2006.

9 D. E. Knuth, J. H. Morris Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

10 R. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. In *Proc. FOCS*, pages 596–604. IEEE Computer Society, 1999.

11 M. G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25(1-2):145–153, 1989.

12 F. Manea, R. Mercas, and D. Nowotka. Fine and Wilf's theorem and pseudo-repetitions. In *Proc. MFCS*, volume 7464 of *LNCS*, pages 668–680. Springer, 2012.

13 F. Manea, M. Müller, and D. Nowotka. The avoidability of cubes under permutations. In *Proc. DLT*, volume 7410 of *LNCS*, pages 416–427. Springer, 2012.

14 Zhi Xu. A minimal periods algorithm with applications. In *Proc. CPM*, volume 6129 of *LNCS*, pages 51–62. Springer, 2010.

Counting Homomorphisms to Cactus Graphs Modulo 2*

Andreas Göbel, Leslie Ann Goldberg, and David Richerby

Department of Computer Science, University of Oxford, Oxford, UK

Abstract

A homomorphism from a graph G to a graph H is a function from $V(G)$ to $V(H)$ that preserves edges. Many combinatorial structures that arise in mathematics and computer science can be represented naturally as graph homomorphisms and as weighted sums of graph homomorphisms. In this paper, we study the complexity of counting homomorphisms modulo 2. The complexity of modular counting was introduced by Papadimitriou and Zachos and it has been pioneered by Valiant who famously introduced a problem for which counting modulo 7 is easy but counting modulo 2 is intractable. Modular counting provides a rich setting in which to study the structure of homomorphism problems. In this case, the structure of the graph H has a big influence on the complexity of the problem. Thus, our approach is graph-theoretic. We give a complete solution for the class of cactus graphs, which are connected graphs in which every edge belongs to at most one cycle. Cactus graphs arise in many applications such as the modelling of wireless sensor networks and the comparison of genomes. We show that, for some cactus graphs H, counting homomorphisms to H modulo 2 can be done in polynomial time. For every other fixed cactus graph H, the problem is complete for the complexity class $\oplus$P which is a wide complexity class to which every problem in the polynomial hierarchy can be reduced (using randomised reductions). Determining which H lead to tractable problems can be done in polynomial time. Our result builds upon the work of Faben and Jerrum, who gave a dichotomy for the case in which H is a tree.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases modular counting, homomorphisms, cactus graph, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.350

1 Introduction

A homomorphism from a graph G to a graph H is a function from $V(G)$ to $V(H)$ that preserves edges (i.e., maps every edge of G to some edge of H). Many combinatorial structures arising in mathematics and computer science can be represented naturally as graph homomorphisms. For example, proper q-colourings of a graph G correspond to homomorphisms from G to the q-clique, and independent sets of G correspond to homomorphisms from G the 2-vertex connected graph with one self-loop (the set of vertices of G mapped to the unlooped vertex is independent). Partition functions in statistical physics such as the Ising, Potts, and hard-core models arise naturally as weighted sums of homomorphisms. See, e.g., [3, 11].

* 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. 334828. The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein. Some of the initial research was supported by the EPSRC grant EP/I011528/1. A full version with proofs appears in [10].



© Andreas Göbel, Leslie Ann Goldberg, and David Richerby;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 350–361



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



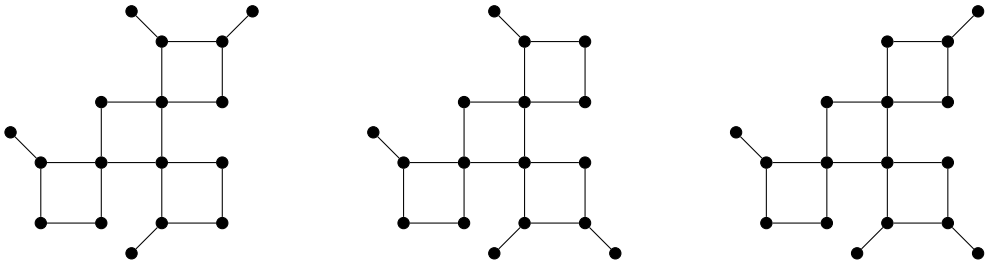


Figure 1 $\oplus$HomsToH_1 and $\oplus$HomsToH_3 are $\oplus$P-complete, but $\oplus$HomsToH_2 is in FP.

We study the complexity of counting homomorphisms modulo 2. For graphs G and H, let $\mathrm{Hom}(G, H)$ be the set of homomorphisms from G to H. For each fixed H, we study the computational problem $\oplus$HomsToH, i.e., computing $|\mathrm{Hom}(G, H)| \bmod 2$, given input G.

The structure of the graph H has a big influence on the complexity of $\oplus$HomsToH. For example, consider the graphs H_1, H_2 and H_3 depicted in Figure 1. Our result implies that $\oplus$HomsToH_1 is complete for the class $\oplus$P (under polynomial-time Turing reductions). H_2 is constructed by moving the top right "bristle" from H_1 down to the bottom right. Under the standard assumption that $\oplus\mathrm{P} \neq \mathrm{FP}$, moving this bristle makes the problem easier – our result implies that $\oplus$HomsToH_2 is solvable in polynomial time. The graph H_3 is constructed by moving the top bristle from left to right in H_2. This makes the problem hard again – $\oplus$HomsToH_3 is $\oplus$P-complete.

The goal of this research is to study the complexity of $\oplus$HomsToH for every fixed graph H and to determine for which graphs H the problem is in FP, for which it is $\oplus$P-complete, and whether there are any H for which the problem has intermediate complexity. In this paper, we give a complete solution to this problem for the class of *cactus graphs*.

A cactus graph is a connected graph in which every edge belongs to at most one cycle. Cactus graphs were first defined by Harary and Uhlenbeck [13] who attributed them to the physicist Husimi and therefore called them *Husimi Trees*. Cactus graphs arise, for example, in the modelling of wireless sensor networks [2] and in the comparison of genomes [18]. Some NP-hard graph problems can be solved in polynomial time on cactus graphs [1].

1.1 The complexity of modular counting

The complexity of modular counting is an interesting topic with some surprising results and we only mention a few highlights here. It is important to note that $\oplus$P (first studied in [12, 17]) is a very large complexity class. We treat $\oplus$P from the point of view of function computation: it is all problems of the form "compute $f(x) \bmod 2$" where computing $f(x)$ is in #P. $\oplus$P is sufficiently powerful that there is randomised polynomial-time reduction [19] from every problem in the polynomial hierarchy to some problem in $\oplus$P. Thus, under the natural hypothesis that problems in the higher levels of the polynomial hierarchy are not solvable in (randomised) polynomial time, $\oplus$P-complete problems are much harder than problems in FP, which is the class of of function-computation problems that are solvable in polynomial time.

The complexity of counting modulo 2 is different from the complexity of decision problems and counting problems. First, consider an NP-complete decision problem. The mod-2

counting version of this problem can be intractable, as you might expect (for example, counting vertex covers or independent sets modulo 2 is $\oplus$P-complete [20]) but it can also be tractable. As an example, consider counting proper 3-colourings of a graph modulo 2. There are an even number of 3-colourings that use all three colours, since there are six permutations of these colours. There are also an even number of 3-colourings that use exactly two colours, since the colours can be swapped. It is easy to count 1-colourings, so it is easy to count all proper colourings modulo 2. Next, consider a #P-complete counting problem. The mod-2 counting version of this problem can be intractable or tractable, as the examples given above illustrate. As another example where the mod-2 counting version is tractable, consider the problem of computing the permanent of a matrix modulo 2. Since $-1 \equiv 1 \pmod 2$, the permanent is equal modulo 2 to the determinant, so it can readily be computed in polynomial time.

Another interesting aspect of modular counting is the fact that the value of the modulus can affect the tractability of the problem. As an example, consider the well-known work of Valiant [20] which identified a certain satisfiability problem where satisfying assignments are easy to count modulo 7 but difficult to count modulo 2.

1.2 Dichotomies for graph homomorphism problems

Determining the border between tractability and intractability for large classes of modular counting problems is an important step towards understanding the structure of the problems themselves. In this paper we work within the context of graph homomorphism problems because graph homomorphisms are general enough to capture a wide variety of combinatorial problems, yet they exhibit sufficient structure that dichotomies exist. Hell and Nešetřil [14] pioneered this direction by completely classifying undirected graphs according to the difficulty of the graph homomorphism decision problem. They showed if a fixed graph H has a self-loop, or is bipartite then the problem of determining whether an input graph has a homomorphism to H is in P. For every other fixed graph H, the decision problem is NP-complete.

Over recent years, dichotomy theorems have also been established for the problem of counting graph homomorphisms and computing weighted sums of homomorphisms. Dyer and Greenhill [7] showed that the problem of counting homomorphisms to H is solvable in polynomial time if every component of H is an isolated vertex, a complete graph with all self-loops present, or a complete bipartite graph with no self-loops. For every other H, it is #P-complete. In particular, there are no graphs H for which the problem has intermediate complexity. This dichotomy was extended to the problem of computing weighted sums of homomorphisms to H. A dichotomy was given by Bulatov and Grohe [3] for the case where the weights are positive, by Goldberg, Grohe, Jerrum and Thurley [11] for the case where the weights are real, and by Cai, Chen and Lu [4] for complex weights.

1.3 Counting graph homomorphisms modulo 2

The first results on the complexity of counting graph homomorphisms modulo 2 were obtained by Faben and Jerrum [8, 9], who made some important structural discoveries which we also use.

An *involution* of a graph is an automorphism of order 2. If σ is an automorphism of a graph H then H^σ denotes the subgraph of H induced by the fixed points of σ.

▶ **Lemma 1.** *([9, Lemma 3.3]) If H is a graph and σ is an involution of H then, for any graph G, $|\mathrm{Hom}(G, H)| \equiv |\mathrm{Hom}(G, H^\sigma)| \pmod 2$.*

The lemma is useful because it enables us to reduce the problem of counting homomorphisms to H modulo 2 to the problem of counting homomorphisms to H^σ modulo 2. This leads naturally to the idea of reduction by involutions. Let $\to$ be the relation on graphs where $H \to H'$ if and only if there is an involution σ of H such that $H' = H^\sigma$. Let $\to^*$ be the transitive closure of $\to$. Faben and Jerrum showed that repeatedly applying $\to$ to a graph H reduces H to a unique involution-free graph, up to isomorphism. Also, to classify the complexity of counting homomorphisms to H, it suffices to study the complexity of counting homomorphisms to its connected components.

▶ **Lemma 2.** *([9, Theorem 6.1]) Let H be an involution-free graph. If H has a connected component H_1 such that $\oplus\text{HomsTo}H_1$ is $\oplus$P-hard with respect to polynomial-time Turing reductions, then $\oplus\text{HomsTo}H$ is also $\oplus$P-hard.*

It is easy to see that, if $\oplus\text{HomsTo}H_j$ is solvable in polynomial time for every connected component H_j of H, then $\oplus\text{HomsTo}H$ is also solvable in polynomial time. Faben and Jerrum used the structural results to give a dichotomy for the complexity of $\oplus\text{HomsTo}H$ when H is a tree. Define the "involution-free reduction" H' of a graph H to be the lexicographically-minimal involution-free graph such that $H \to^* H'$. We can state their result as follows.

▶ **Theorem 3.** *([9, Theorem 3.8]) If H is a tree then $\oplus\text{HomsTo}H$ is $\oplus$P-complete if the involution-free reduction of H has more than one vertex. Otherwise, it is in* FP.

Every involution-free tree is asymmetric (has no non-trivial automorphisms). Thus, the technical work of proving Theorem 3 is to show that $\oplus\text{HomsTo}H$ is $\oplus$P-hard for every asymmetric tree H with more than one vertex. Fortunately, this can be done without too much technical complexity. Developing a dichotomy to cover all graphs seems to be much harder and even the dichotomy for cactus graphs requires a substantial technical effort, as we will see. Nevertheless, there is a general conjecture as to what the outcome would be.

▶ **Conjecture 4** (Faben and Jerrum). *Let H be a (not necessarily simple) graph. $\oplus\text{HomsTo}H$ is in* FP *if the involution-free reduction of H is empty, a single vertex (with or without a self-loop) or a graph with two isolated vertices, exactly one having a self-loop. Otherwise, it is $\oplus$P-complete.*

1.4 Our result

Recall that a cactus graph is a connected, simple graph in which every edge belongs to at most one cycle. Our main result gives a proof of Faben and Jerrum's conjecture for cactus graphs.

▶ **Theorem 5.** *Let H be a simple graph with every edge in at most one cycle. If the involution-free reduction of H has at most one vertex, then $\oplus\text{HomsTo}H$ is solvable in polynomial time. Otherwise, $\oplus\text{HomsTo}H$ is complete for $\oplus$P under polynomial-time Turing reductions.*

To prove this, we must investigate all involution-free cactus graphs, not just the asymmetric ones. This is because, unlike the situation for trees, there are involution-free cactus graphs, such as H_4 in Figure 2, that have non-trivial automorphisms. This graph has no involutions but has an automorphism of order 3 which rotates the cycle. Incidentally, it is easy to see that the graph H_5 in the figure has an involution that moves all vertices, so $\oplus\text{HomsTo}H_5$ is in FP. Our result implies that $\oplus\text{HomsTo}H_4$ is $\oplus$P-complete.

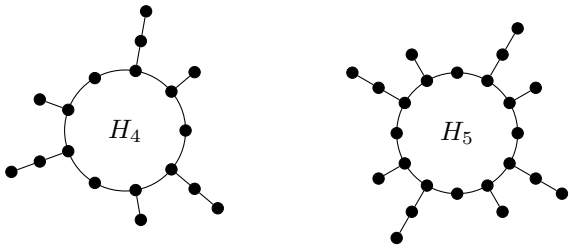


Figure 2 $\oplus\text{HomsTo}H_4$ is $\oplus$P-complete but $\oplus\text{HomsTo}H_5$ is in FP.

To prove the hardness result in Theorem 5, we introduce three graph-theoretic notions: hardness gadgets, partial hardness gadgets, and mosaics. Hardness gadgets and partial hardness gadgets are, as the name suggests, structures for proving $\oplus$P-hardness. Mosaics are graphs built on unions of 4-cycles. They are what is left in inductive cases where hardness gadgets don't exist and we use them in our inductive proof. Our approach is therefore recursive: we decompose involution-free cactus graphs at cut vertices so that every component contains at least one of these three induced structures. We then combine these structures to obtain hardness gadgets in the original graph. If an asymmetric graph H contains a hardness gadget, then it is relatively easy to show that $\oplus\text{HomsTo}H$ is $\oplus$P-complete — the proof is by reduction from the problem of counting independent sets modulo 2, generalising the argument for trees. We will discuss the situation in which H is not asymmetric presently.

Even when H is asymmetric, the most difficult part of the argument is showing that every non-trivial involution-free cactus graph does actually contain a hardness gadget. The presence of cycles greatly complicates this argument, hence the need to define hardness gadgets, partial hardness gadgets and mosaics and to decompose cactus graphs into components with these three different structures, which can then be combined to form hardness gadgets.

When the graph has non-trivial automorphisms, there is a further complication. Suppose that G and H are graphs and that p is a function from $V(G)$ to $2^{V(H)}$. A homomorphism f from G to H is said to satisfy the "pinning" function p if, for every $v \in V(G)$, we have $f(v) \in p(v)$. Now suppose that H is an involution-free graph containing a hardness-gadget. The high-level strategy for proving that $\oplus\text{HomsTo}H$ is $\oplus$P-hard is to first reduce the problem of counting independent sets modulo 2 to the problem of counting pinned homomorphisms from G to H (modulo 2) and then to reduce the latter problem to $\oplus\text{HomsTo}H$. This pinning approach has been used successfully in dichotomy theorems in related domains [3, 5, 6]. When H is asymmetric, the application of pinning works smoothly. Building on work of Lovász [15], Faben and Jerrum reduced the pinned problem to the unpinned one for the case in which the pinning function pins some vertex to an orbit in the automorphism group of H. When H is asymmetric (as it is, when H is a tree), the orbit is just a single vertex, and this is just what is required. If H is not asymmetric, we do not know how to pin a vertex of G to a particular vertex in H. To get around this, we augment G with a copy of H and we pin every vertex in the copy to its own orbit in the automorphism group of H. Every homomorphism from an involution-free cactus graph to itself that respects the orbits of all of its vertices is, in fact, an automorphism of H, and this enables us to solve the problem.

Theorem 5 gives a dichotomy for cactus graphs. If the involution-free reduction of H has at most one vertex then $\oplus\text{HomsTo}H$ is in FP. Otherwise, it is $\oplus$P-complete. Furthermore, the meta-problem of determining which is the case, given input H, is computationally easy. Finding an involution of H reduces in polynomial time to computing the size of H's automorphism group modulo 2. The latter problem is in FP for cactus graphs because, for example, they are planar.

1.5 Notation

Given two graphs G and H (not necessarily vertex-disjoint), $G \cup H$ is the graph $(V(G) \cup V(H), E(G) \cup E(H))$. If E is a set of edges, let $V(E)$ denote the set of endpoints of edges in E and let $G \cup E$ denote the graph $G \cup (V(E), E)$. Given a set $V' \subseteq V(G)$, let $G - V' = G[V(G) \setminus V']$. We use the phrase "$j$-walk" in a graph to refer to a walk of length j.

We use $\Gamma_H(v)$ to denote the set of neighbours of vertex v in H. A *rooted graph* is a pair (H, x) where H is a graph and $x \in V(H)$ is a distinguished vertex, the *root*. An automorphism of (H, x) is an automorphism of H that fixes x.

We use $\mathrm{Aut}(H)$ to denote the automorphism group of H and, for $v \in V(H)$, we use $\mathrm{Orb}_H(v)$ to denote the set of vertices of H in the orbit of v under the action of $\mathrm{Aut}(H)$.

2 Pinning, gadgets and mosaics

In this section, we discuss pinning and define the gadgets we use to prove $\oplus$P-hardness of $\oplus\textsc{HomsTo}H$ problems by reduction from $\oplus$IS, counting independent sets modulo 2.

Recall from the introduction that a homomorphism $f\colon V(G) \to V(H)$ satisfies a *pinning function* $p\colon V(G) \to 2^{V(H)}$ if $f(v) \in p(v)$ for all $v \in V(G)$. Let $\mathrm{HomPin}(G, H, p)$ be the set of homomorphisms from G to H that satisfy the pinning function p. Say that a pinning p is r-restrictive if at most r vertices $v \in V(G)$ have $p(v) \neq V(H)$ and for each such vertex v, $p(v)$ is a union of orbits of the automorphism group of H. We consider the following computational problem, which is parameterised by a graph H and a natural number r.

Name: $\oplus r$-$\textsc{PinnedHomsTo}H$.
Input: A graph G and a r-restrictive pinning function $p\colon V(G) \to 2^{V(H)}$.
Output: $|\mathrm{HomPin}(G, H, p)| \pmod 2$.

Extending the work of Faben and Jerrum who, in turn, built on results of Lovász [15], we prove the following theorem.

▶ **Theorem 6.** *Let H be an involution-free graph and let r be a positive integer. There is a polynomial-time Turing reduction from $\oplus r$-$\textsc{PinnedHomsTo}H$ to $\oplus\textsc{HomsTo}H$.*

We next introduce machinery that we will use to prove that $\oplus r$-$\textsc{PinnedHomsTo}H$ is $\oplus$P-complete when H is an involution-free cactus graph and r is defined appropriately.

▶ **Definition 7.** A *hardness gadget* in a graph H is a tuple $(\beta, s, t, O, i, K, k, w)$ where β is a positive integer, s, t and i are vertices of H, $(O, \{i\}, K)$ is a partition of $\Gamma_H(s)$, and $k\colon K \to \mathbb{N}_{>0}$ and $w\colon K \to V(H)$ are functions. The following conditions must be satisfied.

1. $|O|$ is odd.
2. For any $o \in O$ and $y \in O \cup \{i\}$, s is the unique vertex that is adjacent to o and y and has an odd number of β-walks to t.
3. There are an even number of $(1 + \beta)$-walks from i to t.
4. For all $u \in K$, $w(u)$ has an even number of $k(u)$-walks to u and an odd number of $k(u)$-walks to every vertex in $O \cup \{i\}$.

These conditions simplify if $\beta = 1$, since having an odd number of 1-walks to a vertex is the same as being adjacent to it.

The construction used in our reduction from $\oplus$IS is given formally in Definition 12. Given a graph G and a hardness gadget Γ, we will produce a graph G_Γ that includes a copy of $V(G)$. We call the vertices in this copy, "G-vertices". We will use pinning to consider homomorphisms from G_Γ to H that map all G-vertices to neighbours of s. Part 4 of Definition 7 ensures that there will be an even number of such homomorphisms that map any

G-vertices to members of K. These contribute nothing to the total modulo 2 so the effect is to restrict to homomorphisms that map every G-vertex to $O \cup \{i\}$. Part 3 of the definition will ensure that the number of homomorphisms that map adjacent vertices in G to i is even, so these also do not contribute. Thus, the homomorphisms that remain are those in which an independent set of G-vertices are mapped to i. Our key technical result is that every non-trivial, involution-free cactus graph contains a hardness gadget (Theorem 10).

In some cases, our decomposition might yield subgraphs that do not contain hardness gadgets. We are still able to make progress using structures that can be combined with other parts of the graph to produce a hardness gadget. A partial hardness gadget is, essentially, a simplified hardness gadget that has $K = \emptyset$ and that doesn't yet have a "t" vertex: at a later point, we will find a vertex t with the properties necessary to produce a full hardness gadget.

▶ **Definition 8.** A *partial hardness gadget* in a rooted graph (H, x) is a tuple (s, i, O, P), where s is a vertex of H, $(\{i\}, O)$ is a partition of $\Gamma_H(s)$, and P is a path in H. The tuple satisfies the following conditions.

1. $|O|$ is odd.
2. P is the unique shortest path from x to i in H.
3. Ps is the unique shortest path from x to s in H.
4. For each $o \in O$, Pso is the unique shortest path from x to o in H.

The final structures arising in our decompositions are "mosaics". Some of these (those with "shortcuts", defined below) already contain hardness gadgets. In other cases, a mosaic will provide a "t" vertex for a partial hardness gadget elsewhere in the decomposed graph.

▶ **Definition 9.** An unbristled mosaic is the one-vertex rooted graph or a rooted cactus graph that is a union of 4-cycles. A *mosaic* is a rooted graph (H, x) for which there is a partition (V', V'') of $V(H)$ such that: $x \in V'$, $(H[V'], x)$ is an unbristled mosaic, and $E(H) \setminus E(H[V'])$ is a matching between V'' and a subset of V'. The edges of the matching are called *bristles*.

The graphs in Figure 1 would be mosaics if a root were placed at any vertex on a cycle. Note that every vertex of a mosaic is adjacent to at most one bristle, and that the one-vertex rooted graph and a rooted edge are both mosaics.

A *shortcut* in a mosaic (H, x) is a pair of odd-degree vertices, with degree at least 3, that have a unique shortest path P between them, and this path does not contain x. In the full paper, we show that every mosaic with a shortcut contains a hardness gadget.

3 Finding hardness gadgets

In Sections 6 and 7 of the full paper, we prove the following result.

▶ **Theorem 10.** *Every involution-free cactus graph H with more than one vertex contains a hardness gadget.*

Given a cut vertex v of a graph H, let $H'_1, \ldots, H'_\kappa$ be the connected components of $H - \{v\}$. Let the *split* of H at v be the set of graphs $\{H_1, \ldots, H_\kappa\}$, where $H_j = H[V(H'_j) \cup \{v\}]$. To prove Theorem 10, we mostly proceed by splitting at cut vertices and investigating the resulting components. A key point is that, if $\{H_1, \ldots, H_\kappa\}$ is the split of an involution-free graph H at a cut vertex v then each rooted graph (H_j, v) is involution-free, even though the unrooted graph H_j might not be. This allows us to perform an induction on rooted graphs to establish the following lemma. Theorem 10 then follows by choosing an appropriate root and constructing a hardness gadget from the contents of the split at the root.

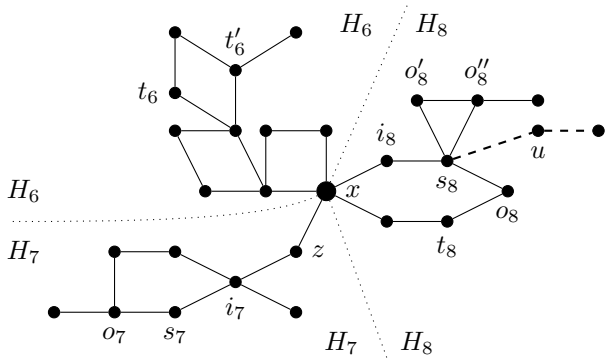


Figure 3 An example graph illustrating the proof ideas of Theorem 10 and Lemma 11.

▶ **Lemma 11.** *Every involution-free rooted cactus graph (H, x) contains a hardness gadget, contains a partial hardness gadget or is a shortcut-free mosaic.*

Rather than attempting to sketch the lengthy and technical proof of Lemma 11, we will work through an example that illustrates the main techniques. Consider the cactus graph of Figure 3. It is involution-free (in fact, asymmetric) and its split at the vertex x gives the three involution-free rooted graphs (H_6, x), (H_7, x) and (H_8, x).

We see immediately that (H_6, x) is a mosaic, and it is shortcut-free, since it has only one odd-degree vertex on a cycle (the degree of x in H_6 is two). Note also that (H_6, x) is asymmetric but the unrooted graph H_6 has an involution that exchanges x with the vertex at distance 2 from it on the same 4-cycle.

Consider, now, (H_7, x). This graph contains the partial hardness gadget $(s_7, i_7, \{o_7\}, xzi_7)$: $(\{i_7\}, \{o_7\})$ partitions $\Gamma_{H_7}(s_7)$, $|\{o_7\}|$ is odd, xzi_7 is the unique shortest x–i_7 path, xzi_7s_7 is the unique shortest x–s_7 path and $xzi_7s_7o_7$ is the unique shortest x–o_7 path.

Now, we turn our attention to (H_8, x), in which we will demonstrate a hardness gadget. In the first instance, consider the graph without the dashed path, the easier case. As the notation suggests, we take $s = s_8$ and $i = i_8$. A helpful feature for us here is the even-length cycle that includes these two vertices: by choosing t to be the vertex t_8, half way around the cycle from i, and taking $\beta = 2$ (so the length of the cycle is $2(\beta + 1)$), we ensure that requirement 3 of the definition of hardness gadgets is met (an even number of $(\beta + 1)$-walks from i to t). We take $O = \{o_8, o'_8, o''_8\}$, which has odd cardinality so satisfies requirement 1. Requirement 2 is that, for each $o \in O$ and $y \in O \cup \{i\}$, s is the unique vertex adjacent to o and y that has an odd number of β-walks to t. $\beta = 2$ and s and x are the only vertices that send an odd number of 2-walks to t. Since x is not adjacent to any vertex in O, s meets the requirement. Finally, since $(O, \{i\})$ is already a partition of $\Gamma_{H_8}(s)$, we set $K = \emptyset$ and requirement 4 is vacuous. Therefore, writing $\perp$ for the function with empty domain,

$$\Gamma = (\beta, s, t, O, i, K, k, w) = (2, s_8, t_8, \{o_8, o'_8, o''_8\}, i_8, \emptyset, \perp, \perp)$$

is a hardness gadget in H_8.

To demonstrate a hardness gadget with non-empty K, consider the rooted cactus graph (H'_8, x) formed by adding the dashed edges to H_8. We take s, t, O, i and β as before but, now, $(O, \{i\})$ is not a partition of $\Gamma_{H'_8}(s)$. Thus, we set $K = \{u\}$, $k(u) = 2$ and $w(u) = u$. u has two 2-walks to itself and one to i and each vertex in O, so requirement 4 is met.

Let us recap: we have split the graph H at cut vertex x and demonstrated that each of the three components of this split contains a hardness gadget or a partial hardness gadget,

or is a shortcut-free mosaic. We now illustrate Theorem 10 by showing how to combine these to produce a hardness gadget in H.

In fact, this is rather easy because the hardness gadget Γ in H_8 is also a hardness gadget in H. This is because the requirements for being a hardness gadget depend on the number of 3-walks from i_8, o_8, o'_8 and o''_8 to t_8 and the number of 2-walks from u to vertices adjacent to s_8; however, none of these walks can ever leave H_8. In the full version of the paper, we give formal *distance requirements* that allow us to determine more generally when a hardness gadget in an induced subgraph of H is also a hardness gadget in H.

Our goal is to illustrate the proof techniques, so we will continue and find a second hardness gadget in H by combining the mosaic (H_6, x) with the partial hardness gadget $(s_7, i_7, \{o_7\}, xzi_7)$ in (H_7, x). As we remarked earlier, if we can find appropriate values for t and β, the partial hardness gadget will become a hardness gadget with $K = \emptyset$. The properties we require of t and β are the following:

- there are an even number of $(1+\beta)$-walks from i_7 to t;
- s_7 is the unique vertex adjacent to o_7 that has an odd number of β-walks to t; and
- s_7 is the unique vertex adjacent to both o_7 and i_7 that has an odd number of β-walks to t.

Since s_7 is the only vertex adjacent to both o_7 and i_7, the third property follows from the second. To make the second property easy to verify, we will choose t to have a unique shortest path in H to o_7, and this path will go through s and have length $1 + \beta$.

Consider the vertices t_6 and t'_6, which are not adjacent but are on the same cycle, and which have degree 2 and 3, respectively. Further, each has a unique shortest path to x and these two paths differ only in their last edge. It is not hard to see that every involution-free mosaic with at least one cycle must contain a pair of vertices with these properties and, in the full paper, we call such a pair of vertices, along with the shared section of their shortest paths to x, a *2,3-path*.

We are going to take $\beta = 6$ (the distance from s_7 to $\{t_6, t'_6\}$) and we claim that we can choose one of $t = t_6$ or $t = t'_6$ to satisfy the first two properties. In fact, either choice satisfies the second property, since either choice for t gives a unique 7-walk to o_7.

To verify the claim, we will show that t_6 and t'_6 have different numbers of 7-walks to i_7, modulo 2. Therefore, one of them has an even number of 7-walks, and that will be our choice for t. There is a unique 5-path from i_7 to each of t_6 and t'_6: write this path as $x_1x_2 \dots x_6$, where $i_7 = x_1$. Every 7-walk from x_1 to x_6 is of one of the following two types:

1. walks that replace one of the edges (x_3, x_4), (x_4, x_5) or (x_5, x_6) by going along the other three edges of the 4-cycle that contains it; and
2. walks that replace one of the vertices x_a $(1 \le a \le 6)$ with the 2-walk $x_a y x_a$, for some $y \in \Gamma_H(x_a)$.

There are exactly three type-1 walks from i_7 to each of t_6 and t'_6. The number of type-2 walks from i_7 to t'_6 is exactly one greater than the number to t_6. For $1 \le a \le 5$, there are the same number of choices for y in each case; however, for $a = 6$, there are three choices of y from t'_6 but only two from t_6. Therefore, the number of 7-walks from i_7 to exactly one of t_6 and t'_6 is even, and we choose that vertex to be t. The reader is invited to check that there are, in total, twenty 7-walks to t'_6 and nineteen to t_6. Thus, the hardness gadget is

$$(\beta, s, t, O, i, K, k, w) = (6, s_7, t'_6, \{o_7\}, i_7, \emptyset, \perp, \perp)\,.$$

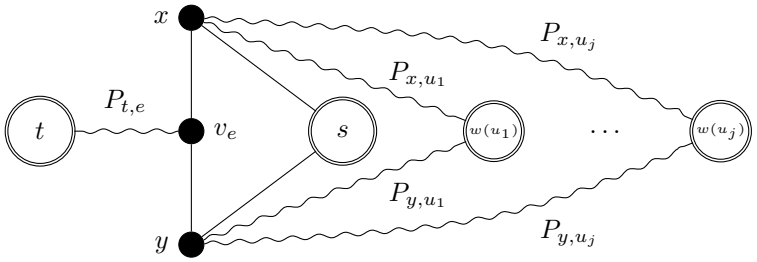


Figure 4 The induced subgraph of G_Γ corresponding to the edge $(x, y) \in E(G)$, with $K = \{u_1, \ldots, u_j\}$. H-vertices have double circles and are pinned in the proof of Theorem 13.

4 Counting homomorphisms to cactus graphs

Having shown that every involution-free cactus graph with more than one vertex contains a hardness gadget, we now use these gadgets to show $\oplus$P-completeness of $\oplus$HomsToH for non-trivial involution-free cactus graphs H. The reduction is from $\oplus$IS, which is $\oplus$P-complete [20]. The reduction is more complicated than the case for trees because an involution-free cactus graph is not necessarily asymmetric — recall the graph H_4 in Figure 2.

In the following definition, "adding a new path P from x to y" in a graph G means forming a graph $G \cup P$ where $V(G) \cap V(P) = \{x, y\}$.

▶ **Definition 12.** Let $\Gamma = (\beta, s, t, O, i, K, k, w)$ be a hardness gadget in H. For any graph G, we construct the graph G_Γ as follows. Begin with the graph $G' = (V', E(H))$ where $V' = V(G) \cup V(H) \cup \{v_e \mid e \in E(G)\}$ (these three sets are assumed to be disjoint) and add:

- for every vertex $x \in V(G)$, the edge (x, s);
- for every edge $e = (x, y) \in E(G)$, the edges (x, v_e) and (y, v_e);
- for every edge $e \in E(G)$, a new β-path $P_{t,e}$ from t to v_e; and
- for every vertex $x \in V(G)$ and every $u \in K$, a new $k(u)$-path $P_{x,u}$ from x to $w(u)$.

In G_Γ, we refer to vertices that are in $V(G)$ as *G-vertices* and those in $V(H)$ as *H-vertices*. Figure 4 illustrates the construction.

Our construction, G_Γ, is more complex than the construction used for trees, because our hardness gadgets are more general than the corresponding structures in trees and because we must deal with graphs H that are involution-free but still have non-trivial automorphisms. To see the problem of non-trivial automorphisms, consider an involution-free cactus graph H that contains a hardness gadget Γ that is moved by an automorphism π of H. We want to pin one vertex to the s-vertex of Γ and another to the t-vertex. However, we can only pin to the orbits of these vertices, which include $\pi(s)$ and $\pi(t)$, respectively. We must avoid counting "inconsistent" homomorphisms that, for example, map the first vertex to s and the second to $\pi(t)$ because we do not know how many of these homomorphisms exist.

▶ **Theorem 13.** *$\oplus$HomsToH is $\oplus$P-complete for every involution-free cactus graph H that contains a hardness gadget.*

Proof (sketch). Using Theorem 6, it suffices to reduce $\oplus$IS to $\oplus r$-PinnedHomsToH where $r = |V(H)|$. Let G be the graph whose independent sets we wish to count and let $\Gamma = (\beta, s, t, O, i, K, k, w)$ be a hardness gadget in H. Let p be the pinning function that maps every H-vertex v to $\mathrm{Orb}_H(v)$ and every other vertex of G_Γ to $V(H)$ and let Φ be the set of

homomorphisms from G_Γ to H that satisfy p. Let $\mathcal{I}(G)$ be the set of independent sets in G. We claim that $|\Phi| \equiv |\mathcal{I}(G)| \pmod 2$.

It can be shown that any $\phi \in \Phi$ acts as an automorphism on the H-vertices. Let $\Phi_\pi \subseteq \Phi$ be set of homomorphisms where this automorphism is π. Writing id for the trivial automorphism, every $\phi \in \Phi_{\mathrm{id}}$ has $\phi(s) = s$ and, for all G-vertices v, $\phi(v) \in O \cup \{i\} \cup K$. For each G-vertex v, $|\{\phi \in \Phi_{\mathrm{id}} \mid \phi(v) \in K\}|$ is even because, when $\phi(v) \in K$, $P_{v,w(\phi(v))}$ can map to an even number of $k(\phi(v))$-walks in H. So, to compute $|\Phi_{\mathrm{id}}|$ modulo 2, it suffices to count the homomorphisms $\phi \in \Phi_{\mathrm{id}}$ where $\phi(v) \in O \cup \{i\}$ for all G-vertices v. For such a homomorphism, let S_ϕ be the set of G-vertices mapped to i. If $S \in \mathcal{I}(G)$, each G-vertex not in S can map to any of the odd number of elements of O and, for each edge e, we must have $\phi(v_e) = s$ and $P_{t,e}$ can map to an odd number of β-walks in H. If $S \notin \mathcal{I}(G)$, there are an even number of homomorphisms ϕ with $S_\phi = S$ because, if adjacent G-vertices x and y map to i, there are an even number of ways to map the paths $P_{t,(x,y)}x$ and $P_{t,(x,y)}y$ to $(1+\beta)$-walks in H. Thus, $|\Phi_{\mathrm{id}}| \equiv |\mathcal{I}(G)| \pmod 2$. For any automorphism π of H, $|\Phi_\pi| = |\Phi_{\mathrm{id}}|$, so $|\Phi| = |\Phi_{\mathrm{id}}|\,|\operatorname{Aut} H|$. H is involution-free so, by Cauchy's Group Theorem [16], $|\operatorname{Aut}(H)|$ is odd, so $|\Phi| \equiv |\Phi_{\mathrm{id}}| \equiv |\mathcal{I}(G)| \pmod 2$. ◀

We can now prove our main result.

▶ **Theorem 5.** *Let H be a simple graph with every edge in at most one cycle. If the involution-free reduction of H has at most one vertex, then $\oplus\textsc{HomsTo}H$ is solvable in polynomial time. Otherwise, $\oplus\textsc{HomsTo}H$ is complete for $\oplus$P under polynomial-time Turing reductions.*

Proof. Let H' be the involution-free reduction of H. If H' has at most one vertex then $\oplus\textsc{HomsTo}H'$ is trivially solvable in polynomial time. By Lemma 1, every graph G satisfies $|\operatorname{Hom}(G,H)| \equiv |\operatorname{Hom}(G,H')| \pmod 2$ so $\oplus\textsc{HomsTo}H$ is also solvable in polynomial time.

If H' has more than one vertex, then some component H_1 of H' has more than one vertex (since H' is involution-free). Also, H_1 is involution-free. Since H_1 is an induced subgraph of H, it is a cactus graph. By Theorems 10 and 13, $\oplus\textsc{HomsTo}H_1$ is $\oplus$P-hard. By Lemma 2, $\oplus\textsc{HomsTo}H'$ is $\oplus$P-hard. But Lemma 1 gives a reduction from $\oplus\textsc{HomsTo}H'$ to $\oplus\textsc{HomsTo}H$, so $\oplus\textsc{HomsTo}H$ is also $\oplus$P-hard. ◀

References

1 B. Ben-Moshe, B. K. Bhattacharya, Q. Shi, and A. Tamir. Efficient algorithms for center problems in cactus networks. *Theor. Comput. Sci.*, 378(3):237–252, 2007.

2 B. Ben-Moshe, A. Dvir, M. Segal, and A. Tamir. Centdian computation in cactus graphs. *J. Graph Algorithms Appl.*, 16(2):199–224, 2012.

3 A. A. Bulatov and M. Grohe. The complexity of partition functions. *Theor. Comput. Sci.*, 348(2–3):148–186, 2005.

4 J.-Y. Cai, X. Chen, and P. Lu. Graph homomorphisms with complex values: A dichotomy theorem. In *Proc. ICALP (1)*, pages 275–286, 2010.

5 N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Inform. Comput.*, 125(1):1–12, 1996.

6 M. E. Dyer, L. A. Goldberg, and M. Jerrum. The complexity of weighted Boolean #CSP. *SIAM J. Comput.*, 38(5):1970–1986, 2009.

7 M. E. Dyer and C. S. Greenhill. The complexity of counting graph homomorphisms. *Random Struct. Algorithms*, 17(3–4):260–289, 2000.

8 J. Faben. *The Complexity of Modular Counting in Constraint Satisfaction Problems.* PhD thesis, Queen Mary, University of London, 2012.

9 J. Faben and M. Jerrum. The complexity of parity graph homomorphism: an initial investigation. *CoRR*, abs/1309.4033, 2013.

10 Andreas Göbel, Leslie Ann Goldberg, and David Richerby. The complexity of counting homomorphisms to cactus graphs modulo 2. *CoRR*, abs/1307.0556, 2013.

11 L. A. Goldberg, M. Grohe, M. Jerrum, and M. Thurley. A complexity dichotomy for partition functions with mixed signs. *SIAM J. Comput.*, 39(7):3336–3402, 2010.

12 L. M. Goldschlager and I. Parberry. On the construction of parallel computers from various bases of Boolean functions. *Theor. Comput. Sci.*, 43:43–58, 1986.

13 F. Harary and G. E. Uhlenbeck. On the number of Husimi trees. I. *Proc. Nat. Acad. Sci. U. S. A.*, 39:315–322, 1953.

14 P. Hell and J. Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990.

15 L. Lovász. Operations with structures. *Acta Math. Acad. Sci. Hungar.*, 18:321–328, 1967.

16 J. H. McKay. Another proof of Cauchy's group theorem. *The American Mathematical Monthly*, 66:119, 1959.

17 Christos H. Papadimitriou and Stathis Zachos. Two remarks on the power of counting. In *Proceedings of the 6th GI-Conference on Theoretical Computer Science*, pages 269–276, London, UK, UK, 1982. Springer-Verlag.

18 B. Paten, M. Diekhans, D. Earl, J. St. John, J. Ma, B. B. Suh, and D. Haussler. Cactus graphs for genome comparisons. *J. Comput. Biol.*, 18(3):469–481, 2011.

19 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

20 L. G. Valiant. Accidental algorithms. In *Proc. FOCS*, pages 509–517, 2006.

Irreversible computable functions

Mathieu Hoyrup

Inria, LORIA, Villers-lès-Nancy, France
mathieu.hoyrup@inria.fr

Abstract

The strong relationship between topology and computations has played a central role in the development of several branches of theoretical computer science: foundations of functional programming, computational geometry, computability theory, computable analysis. Often it happens that a given function is not computable simply because it is not continuous. In many cases, the function can moreover be proved to be non-computable in the stronger sense that it does not preserve computability: it maps a computable input to a non-computable output. To date, there is no connection between topology and this kind of non-computability, apart from Pour-El and Richards "First Main Theorem", applicable to linear operators on Banach spaces only.

In the present paper, we establish such a connection. We identify the discontinuity notion, for the inverse of a computable function, that implies non-preservation of computability. Our result is applicable to a wide range of functions, it unifies many existing *ad hoc* constructions explaining at the same time what makes these constructions possible in particular contexts, sheds light on the relationship between topology and computability and most importantly allows us to solve open problems. In particular it enables us to answer the following open question in the negative: if the sum of two shift-invariant ergodic measures is computable, must these measures be computable as well? We also investigate how generic a point with computable image can be. To this end we introduce a notion of genericity of a point w.r.t. a function, which enables us to unify several finite injury constructions from computability theory.

1998 ACM Subject Classification F.1.1 Models of Computation/Computability theory, F.4.1 Mathematical Logic/Computability theory

Keywords and phrases Computability theory, computable analysis, finite injury, generic set

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.362

1 Introduction

Many problems in classical computability theory [13] and computable analysis [12, 17] amount to studying the computability of some function f defined on continuous spaces such as the Cantor space or the space of real numbers. One is usually interested in three increasingly stronger notions of computability for f:

(i) $f(x)$ is computable for every computable x;
(ii) $f(x)$ is computable *relative* to x for every x;
(iii) $f(x)$ is computable relative to x for every x, *uniformly* in x.

In the first case we say that f is *computably invariant* (terminology introduced in [1]). In the third case we simply say that f is *computable*. It happens that many interesting functions are not computable and even not computably invariant. For instance Braverman and Yampolsky proved that the function mapping a parameter to the corresponding Julia set does not satisfy (ii); they later strengthened that result by proving that it does not satisfy (i) either. By contrast, the function mapping a parameter to the corresponding *filled* Julia set does satisfy condition (ii), while it does not satisfy (iii) because it is discontinuous [2].


© Mathieu Hoyrup;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 362–373

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


While functions that are not computable often fail to be computably invariant, the proof of the former is usually much simpler than the proof of the latter. Indeed, it is often based on the fundamental result that a computable function must be continuous. Hence proving that a function is not computable is often a purely topological argument.

However proving that a function is not computably invariant is usually much more challenging, as a counterexample must be constructed, by encoding the halting set or by using more involved computability-theoretic arguments based on priority methods, e.g. Our point is that topology is still at play in many computability-theoretic constructions[1]. Usually the construction of a computable element whose image is not computable implicitly makes use of the discontinuity of the function. Of course mere discontinuity is not sufficient in general to carry out such a construction: there exist discontinuous functions that are computably invariant, such as the floor function or the function that maps a real number to its binary expansion. More is needed and our question is: what discontinuity property is needed to make such a construction possible?

Such discontinuity properties have already been sought by several authors. Pour-El and Richards "First Main Theorem" [12] shows that in the case of linear operators with c.e. closed graph, if the operator is unbounded (i.e., discontinuous) then it is not computably invariant (it is actually an equivalence). Their result subsumes many *ad hoc* constructions, such as Myhill's differentiable computable function whose derivative is not computable [10]. As part of their open problem no. 7, Pour-El and Richards ask whether their First Main Theorem can be extended to nonlinear operators. A generalization of their theorem to certain algebraic structures was proved by Brattka [1], applicable to operators on the set of compact subsets of $\mathbb{R}$.

In these results, the underlying algebraic structures enable the authors to provide counterexamples via explicit expressions (such as linear combinations of basic elements with well-chosen weights) by encoding the halting set, which contrasts with many situations in computability theory where explicit constructions are rarely possible and priority methods are often needed to build counterexamples (Friedberg-Muchnik construction of Turing incomparable c.e. sets, e.g.). This observation allows one to hope for stronger results whose proofs involve more complicated, non-explicit constructions.

In this paper we present such a result, applicable to inverses of computable functions. We work on effective topological spaces and effective Polish spaces without additional structure, which makes our result applicable in many situations. We introduce a topological notion, *irreversibility* of a function, whose effective version entails the existence of a non-computable point whose image is computable. We think that this notion is rather simple to verify on particular instances. The proof of the result implicitly uses the priority method with finite injury. We think that our discontinuity notion is rather natural and, in concrete situations, much easier to verify than constructing a computable element whose pre-image is not computable. In other words, our result is not merely an abstract generalization of existing constructions, but a powerful theorem that provides insight into computability theory, as illustrated by the numerous examples we give.

This work was originally motivated by the following question, left open in [4]: are there two non-computable shift-invariant ergodic measures whose sum is computable? As an application of our main result, we positively answer this question.

[1] for instance the role of Baire category in computability theory has been revealed by several authors (see [9] e.g.)

We push our investigation further by studying the following question: how non-computable can a point with a computable image be? We introduce a notion of genericity of a point w.r.t. a function and prove that generic points with computable images exist. The construction unifies several finite injury arguments.

The paper is organized as follows: in Section 2 we introduce basic notions of computable analysis; in Section 3 we introduce a notion of continuous invertibility at a point and prove that for "almost" every point, if a function is computably invertible at that point then it is continuously invertible there (Theorem 7). In Section 4 we introduce the notion of an *irreversible function*, which in substance expresses that a function is topologically hard to inverse. In Section 5 we present our main result: a function that is topologically hard to inverse is computably hard to inverse, in particular it maps a non-computable point to a computable image. In Section 5.1 we present an application of our main result to the non-computability of the ergodic decomposition. In Section 6 we introduce a notion of genericity w.r.t. a function which unifies several finite injury constructions.

A complete version of the paper with all the proofs is available at `http://hal.inria.fr/hal-00915952`. In the present version, we avoid some technical considerations that are necessary for the proofs of the results but not for their understanding.

2 Background and notations

We assume familiarity with basic computability theory on the natural numbers. We implicitly use Weihrauch's notions of computability on effective topological spaces, based on the standard representation (see [17] for more details), however we do not express them in terms of representations.

2.1 Notations

In a metric space (X, d), if $x \in X$ and $r \in (0, +\infty)$ then we denote the open ball with center x and radius r by $B(x, r) = \{x' \in X : d(x, x') < r\}$. We denote the corresponding closed ball by $\overline{B}(x, r) = \{x' \in X : d(x, x') \leq r\}$. The Cantor space of infinite binary sequences, or equivalently subsets of $\mathbb{N}$, is denoted by $2^{\mathbb{N}}$. The halting set, denoted $\emptyset'$, is the set of numbers of Turing machines that halt. It is a noncomputable set that is computably enumerable (c.e.).

2.2 Effective topology

An *effective topological space* $(X, \tau, \mathcal{B})$ consists of a topological space (X, τ) together with a countable basis $\mathcal{B} = \{B_0, B_1, \ldots\}$ numbered in such a way that the finite intersection operator is computable. An open subset $U \subseteq X$ is *effectively open* if $U = \bigcup_{k \in W} B_k$ for some c.e. set $W \subseteq \mathbb{N}$.

To a point $x \in X$ we associate $N(x) = \{n \in \mathbb{N} : x \in B_n\}$. By an *enumeration of* $N(x)$ we mean a total function $f : \mathbb{N} \to \mathbb{N}$ whose range is $N(x)$. A point x is *computable* if $N(x)$ is c.e., i.e. if $N(x)$ has a computable enumeration.

Given points x, y in effective topological spaces X, Y respectively, we say that y is *computable relative to* x if there is an oracle Turing machine M that, given any enumeration of $N(x)$ as oracle, outputs an enumeration of $N(y)$. We denote it by $M^x = y$. In other words, y is computable relative to x if $N(y)$ is enumeration reducible to $N(x)$. As proved by Selman [14] and pointed out by Miller [8], y is computable relative to x if and only if

every enumeration of $N(x)$ computes an enumeration of $N(y)$ (uniformity is not explicitly required, but is a consequence).

A (possibly partial) function $f : X \to Y$ is *computable* if there is a machine M such that for every $x \in \mathrm{dom}(f)$, $M^x = f(x)$. A computable function is always continuous.

2.3 Effective Polish spaces

An *effective Polish space* is a topological space such that there exists a dense sequence $s_0, s_1, \ldots$ of points, called *simple* points and a complete metric d inducing the topology, such that all the reals numbers $d(s_i, s_j)$ are computable uniformly in (i, j). Every effective Polish space can be made an effective topological space, taking as canonical basis the open balls $B(s, r)$ with s simple point and r positive rational together with a standard effective numbering.

In an effective Polish space, a point x is computable if and only if for every $\epsilon > 0$ a simple point s can be computed, uniformly in ϵ, such that $d(s, x) < \epsilon$.

We will be concerned with computability and Baire category, so we will naturally meet the notion of a 1-generic point: a point that does not belong to any "effectively meager set" in the following sense.

▶ **Definition 1.** $x \in X$ is 1-*generic* if x does not belong to the boundary of any effective open set. In other words, for every effective open set U, either $x \in U$ or there exists a neighborhood B of x disjoint from U.

By the Baire category theorem, every Polish space is a Baire space so 1-generic points exist and form a co-meager set.

3 A non-uniform result

Let X be an effective Polish space, Y an effective topological space and $f : X \to Y$ a (total) computable function.

To introduce informally the results of this section, assume temporarily that f is one-to-one. If f^{-1} is computable, i.e. if every x is computable relative to $f(x)$ *uniformly* in x, then f^{-1} is continuous. As mentioned earlier uniformity is crucial here: that some x is computable relative to $f(x)$ does not imply in general that f^{-1} is continuous at $f(x)$. Theorem 7 below surprisingly shows that a non-uniform version can still be obtained, valid at most points.

Let us now make it precise and formal. We do not assume anymore that f is one-to-one.

When focusing on the problem of inverting a function, one comes naturally to the following basic notions:

- f is *invertible* at x if x is the only pre-image of $f(x)$,
- f is *locally invertible* at x if x is isolated in the pre-image of $f(x)$.

If one has access to x via its image only, then x is determined unambiguously in the first case, with the help of a discrete advice (a basic open set isolating x) in the second case. However, "being uniquely determined" is not sufficient in practice: physically or computationally, one cannot know entirely $f(x)$ in one step, but progressively as a limit of finite approximations. We need to consider stronger, topological versions of the two basic notions of invertibility, expressing that x can be recovered from the knowledge of its image given by finer and finer neighborhoods.

▶ **Definition 2.** Let $f : X \to Y$ be a function. We say that f is *continuously invertible at* x if the pre-images of the neighborhoods of $f(x)$ form a neighborhood basis of x, i.e. for every neighborhood U of x there exists a neighborhood V of $f(x)$ such that $f^{-1}(V) \subseteq U$.

We say that f is *locally continuously invertible at* x if there exists a neighborhood B of x such that the restriction of f to B is continuously invertible at x, i.e. for every neighborhood U of x there exists a neighborhood V of $f(x)$ such that $B \cap f^{-1}(V) \subseteq U$.

Observe that these notions are very natural when investigating the problem of inverting a function: we think that they are not technical *ad hoc* conditions.

Every effective topological space Y has a countable basis hence is sequential, i.e. continuity notions can be expressed in terms of sequences, which may be more intuitive. We will be particularly interested in the negations of these notions, which we characterize now.

▶ Proposition 3.1. f is not continuously invertible at x if and only if there exist $\delta > 0$ and a sequence x_n such that $d(x, x_n) > \delta$ and $f(x_n)$ converges to $f(x)$.

f is not locally continuously invertible at x if and only if for every $\epsilon > 0$ there exist $\delta > 0$ and a sequence x_n such that $\epsilon > d(x, x_n) > \delta$ and $f(x_n)$ converges to $f(x)$.

Let us illustrate these notions on a few examples.

▶ **Example 3.** If f is one-to-one then f is continuously invertible at x if and only if f^{-1} is continuous at $f(x)$.

▶ **Example 4.** The real function $f(x) = x^2$ is continuously invertible exactly at 0, and locally continuously invertible everywhere (for $x \neq 0$ take for B an open interval avoiding 0).

▶ **Example 5.** The projection $\pi_1 : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ which maps $A_1 \oplus A_2 = \{2n : n \in A_1\} \cup \{2n+1 : n \in A_2\}$ to A_1 is not locally continuously invertible anywhere. Indeed, given $A_1, A_2 \in 2^{\mathbb{N}}$, $A_1 \oplus A_2$ is not isolated in the pre-image by π_1 of $A_1 = \pi_1(A_1 \oplus A_2)$.

▶ **Example 6.** Let X be the Cantor space $2^{\mathbb{N}}$ with the product topology τ generated by the cylinders $[u]$, $u \in 2^*$, Y be the Cantor space with the positive topology τ_{Scott} generated by the sets $\{A \subseteq \mathbb{N} : F \subseteq A\}$ where F varies among the finite subsets of $\mathbb{N}$. The computable elements of the two effective topological spaces are the computable sets and the c.e. sets respectively. Consider the enumeration operator $\text{Enum} := \text{id} : X \to Y$. Enum is computable and one-to-one but its inverse is discontinuous. More precisely, (i) it is continuously invertible exactly at $\mathbb{N}$, (ii) it is locally continuously invertible exactly at the co-finite sets: if A is co-finite then let B be a cylinder specifying all the 0's in A, every cylinder containing A is the intersection of a Scott open set with B.

In general continuous invertibility at a point is strictly stronger than local continuous invertibility. This is not the case for linear operators, where a dichotomy appears. Following Pour-El and Richards [12], by a linear operator $T : X \to Y$ between Banach spaces we mean a linear function $T : \mathcal{D}(T) \to Y$ where $\mathcal{D}(T)$ is a subspace of X.

▶ Proposition 3.2. Let X, Y be Banach spaces and $T : X \to Y$ a one-to-one linear operator.

- If T^{-1} is bounded then T is continuously invertible everywhere.
- If T^{-1} is unbounded then T is nowhere locally continuously invertible.

Proof. The first point simply follows from the fact that T^{-1} is continuous. Assume that T^{-1} is unbounded. There exists a sequence $a_n \in X$ such that $\|a_n\| = 1$ and $\|T(a_n)\| \to 0$. Let $x \in X$ and $\epsilon > 0$. Take $\delta = \epsilon/3$ and define $x_n = x + 2\delta a_n$: $T(x_n)$ converges to $T(x)$ and $\epsilon > \|x - x_n\| > \delta$ for all n. ◀

Observe that in the case when T is not one-to-one, T is also nowhere locally continuously invertible, with exactly the same proof (one can take $a_n = a$ for some a with $\|a\| = 1$ and $\|T(a)\| = 0$).

We now come to our first result.

▶ **Theorem 7.** *Let $f : X \to Y$ be a computable function and $x \in X$ a 1-generic point. If x is computable relative to $f(x)$ then f is locally continuously invertible at x.*

Proof idea. Assume that f is not locally continuously invertible at x and that there is a Turing machine M that computes x on oracle $f(x)$. We show that x belongs to the boundary of an effective open set U, i.e. that x is not 1-generic.

Given a point y, there are two possible ways in which a machine may fail to compute y from $f(y)$: either it diverges, or it outputs something that is incompatible with y. The latter can be recognized in finite time: we then say that $M^{f(y)}$ *positively* fails to compute y. Our effective open set U is the set of points y such that $M^{f(y)}$ positively fails to compute y.

First, if f is not continuously invertible at x, there exists $\delta > 0$ and a sequence x_n such that $d(x_n, x) > \delta$ and $f(x_n)$ converges to $f(x)$. If n is sufficiently large then $f(x_n)$ is arbitrarily close to $f(x)$ so $M^{f(x_n)}$ computes an arbitrarily refined approximation of x. If we take n so large that $M^{f(x_n)}$ computes x at precision $< \delta/2$, then $M^{f(x_n)}$ positively fails to compute x_n so x_n belongs to U.

Now, if f is not *locally continuously invertible at x* then x_n can be taken arbitrarily close to x, so x belongs to the closure of U. ◀

In the sequel we introduce a condition on f which roughly means that f is "almost nowhere" locally continuously invertible and that entails (i) the existence of an x that is not computable relative to $f(x)$ (Theorem 13) and, better, (ii) the existence of a non-computable x such that $f(x)$ is computable (Theorem 20).

4 Reversibility

We define two dual notions for a function: reversibility (Section 4.1) and irreversibility (Section 4.2). In the sense of Baire category, a reversible function is continuously invertible almost everywhere; an irreversible function is almost nowhere locally continuously invertible.

4.1 Reversible functions

Let X, Y be T_0 topological spaces. For a continuous function $f : X \to Y$, the following are equivalent:

- f is one-to-one and $f^{-1} : f(X) \to X$ is continuous,
- the initial topology of f is the topology of X, i.e. for every open set $U \subseteq X$ there exists an open set $V \subseteq Y$ such that $U = f^{-1}(V)$.

A function satisfying these conditions can be *reversed* in the sense that x can be recovered from $f(x)$ for every x: x is not only uniquely determined by $f(x)$, but a neighborhood basis of x can be progressively constructed from a neighborhood basis of $f(x)$.

We first consider a slight weakening of this notion.

▶ **Definition 8.** We say that f is *reversible* if for every non-empty open set $U \subseteq X$ there is an open set $V \subseteq Y$ such that $\emptyset \neq f^{-1}(V) \subseteq U$.

We say that f is *effectively reversible* if $V = V_U$ can moreover be computed from U (basic open set).

▶ Proposition 4.1. If f is continuous and reversible then it is continuously invertible at every point in a dense G_δ-set.

If f is computable and effectively reversible then there is a dense effective G_δ-set D such that $f_{|D}$ is one-to-one and its inverse is computable on $f(D)$, i.e. x is uniformly computable from $f(x)$ when $x \in D$.

In particular if x is 1-generic then x is computable relative to $f(x)$.

4.2 Irreversible functions

We now consider the dual notion: an *irreversible* function is a function that is not reversible, not even locally.

▶ **Definition 9.** f is *irreversible* if for every open set $B \subseteq X$ the restriction $f_{|B} : B \to f(B)$ is not reversible.

Formally, f is irreversible if for every non-empty open set B there exists a non-empty open set $U_B \subseteq B$ such that there is no open set V satisfying $\emptyset \neq f^{-1}(V) \cap B \subseteq U_B$.

In other words, each pre-image of an open set that intersects B does so outside U_B. If $x \in U_B$ then we will never know it from $f(x)$, even with the help of the advice $x \in B$.

Observe that one can assume w.l.o.g. that $f^{-1}(V) \cap B \not\subseteq \overline{U}_B$. Indeed, one can replace U_B by some ball $B(s, r)$ such that $\overline{B}(s, r) \subseteq U_B$.

An application of an irreversible function f to x comes with a loss of information about x, that can hardly be recovered. Being irreversible is orthogonal to not being one-to-one: the function $x \mapsto x^2$ is not one-to-one but not irreversible: x can be (continuously or computably) recovered from x^2; a one-to-one function can be irreversible if its inverse is dramatically discontinuous (examples of such functions will be encountered in the sequel).

In terms of sequences, f is irreversible if and only if for every B there exists a non-empty open set $U_B \subseteq B$ such that for every $x \in U_B$ there is a sequence $x_n \in B \setminus U_B$ such that $f(x_n)$ converges to $f(x)$.

As announced, the set of points at which an irreversible function is locally continuously invertible is small in the sense of Baire category.

▶ Proposition 4.2. Let f be irreversible. There is a dense G_δ-set D such that f is not locally continuously invertible at any $x \in D$.

In other words, for almost every x the application of f to x comes with a "topological information" loss.

The preceding proposition does not rule out the possibility that the restriction of f to a "large" set be continuously invertible (for instance, the characteristic function of the rational numbers is nowhere continuous, but its restriction to the co-meager set of irrational numbers is continuous). The next assertion shows that this is not possible.

▶ Proposition 4.3. Let f be irreversible and $C \subseteq X$ be such that $f_{|C} : C \to f(C)$ is an homeomorphism. Then C is nowhere dense.

Proof. Assume the closure of C contains a ball B. $U_B \cap C$ is non-empty. Let $x \in U_B \cap C$. There exists a sequence $x_n \in B \setminus \overline{U}_B$ such that $f(x_n)$ converges to $f(x)$. By density of C in B, x_n can be taken in C. As $f_{|C}$ is an homeomorphism and $f(x_n)$ converges to $f(x)$, x_n should converges to x and eventually enter U_B, which gives a contradiction. ◀

▶ **Example 10.** Let f be a constant function defined on the Polish space X. f is irreversible if and only if X is perfect, i.e. has no isolated point.

▶ **Example 11.** The first projection $\pi_1 : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ from Example 5 is irreversible. Indeed, to $B = [w]$, associate $U_B = [w00]$. The intersection with $[w]$ of the pre-image of any cylinder cannot be contained in $[w00]$: knowing arbitrarily many bits of $\pi_1(A)$ and the first $|w|$ bits of A does not give any information about the next odd bit of A, so it does not enable one to guess that A belongs to $[w00]$.

In the definition of an irreversible function (Definition 9), B and U_B can be assumed w.l.o.g. to be basic balls.

▶ **Definition 12.** f is *effectively irreversible* if U_B can be computed from B.

The following result is the effective version of Proposition 4.3.

▶ **Theorem 13.** *If f is effectively irreversible then for every 1-generic x, x is not computable relative to $f(x)$.*

Proof. The dense G_δ-set provided by Proposition 4.2 is effective when f is effectively irreversible so it contains every 1-generic point. Hence for every 1-generic x, f is not locally continuously invertible at x. We now apply Theorem 7. ◀

In other words, if x is 1-generic then the application of f to x comes with an "algorithmic information" loss. So if f is effectively irreversible then there exists some x that is not computable relative to $f(x)$.

4.3 Examples

Several well-known results in computability theory can be interpreted using Theorem 13 as consequences of the effective irreversibility of some computable function.

▶ **Example 14.** Consider the enumeration operator of Example 6. Enum is effectively irreversible: to each cylinder $B = [w]$ associate $U_B = [w0]$.

Applying Theorem 13 then gives: if A is 1-generic then A and $\mathbb{N} \setminus A$ have incomparable enumeration degrees. Such an A was first proved to exist by Selman [14].

▶ **Example 15.** The projection $\pi_1 : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ from Examples 5 and 11 is effectively irreversible. Applying Theorem 13 to π_1 and symmetrically to the second projection π_2 gives Jockush and Posner's result [6] that if $A = A_1 \oplus A_2$ is 1-generic then A_1 and A_2 are Turing incomparable, which implies Kleene-Post theorem, taking a $\emptyset'$-computable 1-generic set.

▶ **Example 16.** Jockush [7] proved that every 1-generic $A \in 2^{\mathbb{N}}$ is c.e.a., i.e. A computes some B such that A is c.e. relative to B but not computable relative to B. The proof goes as follows: let $f(A) = \{\langle i, j\rangle : i \in A \wedge \langle i, j\rangle \notin A\}$ (where $\langle\rangle$ is a computable one-to-one pairing function such that $\langle i, j\rangle > i$). f is computable, if A is 1-generic then A is c.e. in $f(A)$ as $i \in A \iff \exists j, \langle i, j\rangle \in f(A)$. We show that f is effectively irreversible, which by Theorem 13 implies that if A is 1-generic then $A \not\leq_T f(A)$.

First observe that f is not one-to-one: given A and i such that $i \notin A$ and $\langle i, 0\rangle \notin A$, there exists $\hat{A} \neq A$ such that $f(\hat{A}) = f(A)$. Add $\langle i, 0\rangle$ to A, and each time some k is added, add all the pairs $\langle k, j\rangle$ that are not already in. One easily checks that $f(\hat{A}) = f(A)$. As a result, given a cylinder $B = [u]$, let $U_B = [u] \cap \{A : i \notin A \text{ and } \langle i, 0\rangle \notin A\}$. For every $A \in U_B$ there is $\hat{A} \in B \setminus U_B$ such that $f(\hat{A}) = f(A)$, so $f^{-1}f(A)$ intersects $B \setminus U_B$: knowing $f(A)$ and that $A \in B$ does not enable one to know that $A \in U_B$.

Again, linear operators provide a large class of examples. An effective Banach space is a Banach space which is an effective Polish space with the metric induced by the norm, such that 0 is a computable point and the vector space operations are computable functions. Many classical Banach spaces $\mathbb{R}$, $\mathcal{C}[0,1]$ (with the uniform norm) or $L^1[0,1]$ are effective Banach spaces.

▶ Proposition 4.4. Let X, Y be effective Banach spaces and $T : X \to Y$ a computable linear operator. Assume that either T is not one-to-one or T is one-to-one and T^{-1} is unbounded. Then T is effectively irreversible.

▶ **Example 17.** Applying Proposition 4.4 and Theorem 13 to the integration operator that maps $f \in \mathcal{C}[0,1]$ to $F : x \mapsto \int_0^x f(t)\,\mathrm{d}t$ gives that if $f \in \mathcal{C}[0,1]$ is 1-generic then f is not computable relative to its primitive F that vanishes at 0.

▶ **Example 18.** Applying Proposition 4.4 and Theorem 13 to the canonical injection from $\mathcal{C}[0,1]$ to $L^1[0,1]$ gives that if $f \in \mathcal{C}[0,1]$ is 1-generic then it is not computable relative to itself, as an element of $L^1[0,1]$. In other words, the description of f as an element of $L^1[0,1]$ contains strictly less algorithmic information than the description of f as an element of $\mathcal{C}[0,1]$.

▶ **Example 19.** A function $f : \mathbb{N} \to \mathbb{N}$ can be described by enumerating its graph or by enumerating the complement of its graph. The former alternative gives in general strictly more information about the function than the latter. Let us make it precise.

Every function F with finite domain induces the cylinder $[F]$ of functions $f : \mathbb{N} \to \mathbb{N}$ extending F. The product topology on the Baire space $\mathbb{B}$ is generated by the cylinders. The negative topology is generated by the complements of the cylinders, as a subbasis. The identity $\mathrm{id} : (\mathbb{B}, \tau) \to (\mathbb{B}, \tau_{\mathrm{neg}})$ is computable: from f one can enumerate the cylinders that are incompatible with f, but the converse cannot be done. id is effectively irreversible: to a cylinder $B = [F]$, associate $U_B = [F] \cup \{n \mapsto 0\}$ where n is fresh, i.e. not in the domain of F.

By Theorem 13, if $f : \mathbb{N} \to \mathbb{N}$ is 1-generic then it is not computable relative to every co-enumeration of its graph.

5 The constructive result

We now present the main result of the paper. It is the constructive version of Theorem 13 as it makes $f(x)$ computable. The construction uses a priority argument with finite injury.

▶ **Theorem 20.** *If f is effectively irreversible then there exists a non-computable x such that $f(x)$ is computable.*

The proof is given in the appendix. The proof uses the priority method with finite injury, which can be seen as a game between a player, computing $f(x)$, and infinitely many opponents (all the Turing machines) trying to compute x.

5.1 Application to the ergodic decomposition

We now present an application of Theorem 20. Let P be a Borel probability measure P over the Cantor space. P is *computable* if the real numbers $P[w]$ are uniformly computable. P is *shift-invariant* if $P[w] = P[0w] + P[1w]$ for each finite string w. P is *ergodic* if it cannot be written as $P = \frac{1}{2}(P_1 + P_2)$ with $P_1 \neq P_2$ both shift-invariant.

The ergodic decomposition theorem says that every shift-invariant measure can be uniquely decomposed into a convex combination (possible uncountable) of ergodic measures. Our question is: given a computable shift-invariant measure, can we compute in a sense its ergodic decomposition? This question was implicitly addressed by V'yugin [16] who constructed a counter example: a countably infinite combination of ergodic measures which is computable but not effectively decomposable. In [4] we raised the following question: does the ergodic decomposition become computable when restricting to finite combinations? As an application of Theorem 20, we solve the problem and prove that it is already non-effective in the finite case:

▶ **Theorem 21.** *There exist two ergodic shift-invariant measures P and Q such that neither P nor Q is computable but $P+Q$ is computable.*

The strategy is as follows: the mapping $(P,Q) \mapsto P+Q$ is computable, two-to-one on the space $\mathcal{E} \times \mathcal{E}$ of pairs of ergodic measures and we prove

▶ **Theorem 22.** *The function $(P,Q) \mapsto P+Q$ defined on $\mathcal{E} \times \mathcal{E}$ is effectively irreversible.*

which implies the result applying Theorem 20.

6 Genericity

Given an effectively irreversible function f,

- Theorem 13 tells us that if x is 1-generic then x is not computable relative to $f(x)$,
- Theorem 20 tells us that there exist non-computable x such that $f(x)$ is computable.

The two results are "disjoint" in the sense that in general a single x cannot at the same time be 1-generic and have a computable image, except for some particular functions like constant functions. We raise the following question: is it possible to bring the two results closer together? How far can x be from being computable, given that $f(x)$ is computable? How *generic* can x be?

We now give an answer to that question. We recall that a topological space always comes with an order called the *specialization order*: $x \leq y$ iff every neighborhood of x is also a neighborhood of y. $x \leq y$ means that if one describes x by listing its basic neighborhoods then one can never distinguish x from y. When the space is Hausdorff, the specialization order is trivial. Here $\leq$ denotes the specialization order on the target space Y of f.

▶ **Definition 23.** x is *f-generic* if x is 1-generic in the subspace $\uparrow_f x := \{x' : f(x) \leq f(x')\}$. In other words, x is f-generic if for every effective open set U, either $x \in U$ or there exists a neighborhood B of x such that $B \cap U \cap \uparrow_f x = \emptyset$.

For instance, taking the first projection $\pi_1 : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ of Example 15, $A = A_1 \oplus A_2$ is π_1-generic iff A_1 is 1-generic relative to A_2.

Here we focus on a few particular instances of this notion, when f is the identity from a space to itself with two different topologies. We will consider

1. the enumeration operator $\mathrm{Enum} = \mathrm{id} : (2^{\mathbb{N}}, \tau_{\mathrm{prod}}) \to (2^{\mathbb{N}}, \tau_{\mathrm{Scott}})$ (Examples 6 and 14),
2. $\mathrm{id} : (2^{\mathbb{N}}, \tau_{\mathrm{prod}}) \to (2^{\mathbb{N}}, \tau_{\mathrm{lex}})$ where τ_{lex} is generated by the sets $\{y \in 2^{\mathbb{N}} : x <_{\mathrm{lex}} y\}$ and
3. $\mathrm{id} : (\mathrm{CL}(2^{\mathbb{N}}), \tau_{\text{hit-or-miss}}) \to (\mathrm{CL}(2^{\mathbb{N}}), \tau_{\mathrm{miss}})$. Here, $\mathrm{CL}(2^{\mathbb{N}})$ is the set of non-empty closed subsets of the Cantor space. τ_{miss} is generated by the sets $\mathcal{U}_u = \{P \in \mathrm{CL}(2^{\mathbb{N}}) : P \cap [u] = \emptyset\}$ where $u \in 2^*$. $\tau_{\text{hit-or-miss}}$ is generated by the sets $\mathcal{U}_u$ together with their complements.

Definition 23 is instantiated as follows:

▶ **Definition 24.** 1. A *generic c.e. set* x is a c.e. set that is 1-generic in the subspace $\{y \in 2^{\mathbb{N}} : x \subseteq y\}$.
2. A *generic left-c.e. real* x is a left-c.e. real that is 1-generic in the subspace $\{y \in 2^{\mathbb{N}} : x \leq_{\text{lex}} y\}$.
3. A *generic* Π^0_1*-class* P is a Π^0_1-class that is 1-generic in the subspace $\{Q \in \text{CL}(2^{\mathbb{N}}) : P \supseteq Q\}$.

A generic element belongs to every effective open set that is dense *above* it, for the corresponding specialization order (while a 1-generic elements belongs to every effective open set that is dense *along* it). The next result is the sought combination of Theorems 13 and 20.

▶ **Theorem 25.** *There exists a co-infinite generic c.e. set, a co-infinite generic left-c.e. real and a generic* Π^0_1*-class without isolated points.*

Proof idea. Kurtz built a left-c.e. weakly 1-generic real (see [11]). The construction even gives a generic left-c.e. real. The construction of a generic c.e. set and of a generic Π^0_1-class are exactly the same, replacing the lexicographic order by inclusion $\subseteq$ of sets and reverse inclusion of classes respectively, which are the specialization orders of the underlying topologies. ◀

Theorem 25 is indeed a strengthening of Theorem 20: in Theorem 7, the 1-genericity assumption can actually be weakened to f-genericity (at least for the particular functions under consideration).

▶ Proposition 6.1. In each one of the three cases, if x is generic inside $\uparrow_f x$ and f is not locally continuously invertible at x then x is not computable.

Proof. Using compactness of the space, one can show that f is not locally continuously invertible at x iff x is not isolated in $\uparrow_f x$, i.e. x belongs to the closure of $\uparrow_f x \setminus \{x\}$. If x is computable then the complement of $\{x\}$ is an effective open set, so x cannot be generic inside $\uparrow_f x$. ◀

Theorem 25 embodies many simple finite injury arguments as Friedberg-Muchnik theorem.

▶ Proposition 6.2. Let A be a co-infinite generic c.e. set. A is hypersimple, $A = A_1 \oplus A_2$ where A_1 and A_2 are Turing incomparable, A is not autoreducible.

Proof. Same argument as for 1-generic sets, observing that the involved open set is not only dense *along* A, but even *above* A. For instance, to prove that $A_2 \not\leq_T A_1$, given a Turing functional ϕ, let $U = \{A_1 \oplus A_2 : \exists n, \phi^{A_1}(n) = 0 \wedge A_2(n) = 1\}$. If $\phi^{A_1} = A_2$ then replacing a 0 in A_2 by a 1 arbitrarily far gives an element of U arbitrarily close to $A_1 \oplus A_2$ that is *above* (i.e. is a superset of) $A_1 \oplus A_2$. ◀

It happens that the co-infinite generic c.e. sets are exactly the p-generic sets defined by Ingrassia [5].

Downey and LaForte [3] proved the existence of non-computable left-c.e. reals x all of whose presentations are computable: each prefix-free c.e. set A of finite binary strings satisfying $\sum_{w \in A} 2^{-|w|} = x$ is actually a computable set. Stephan and Wu [15] proved that any such real is strongly Kurtz-random. It must even be a generic left-c.e. real.

▶ Proposition 6.3. If x is a non-computable left-c.e. real all of whose presentations are computable then x is a generic left-c.e. real.

Proof. Let U be an effective open set that does not contain x: we must find $y > x$ such that $[x, y)$ is disjoint from U. First replace U by $V = U \cup [0, x)$. Let A be a prefix-free c.e. set such that $V = \bigcup_{w \in A}[w]$. The set $A_{<x} = \{w \in A : w <_{\text{lex}} x\}$ is a presentation of x hence it

is computable, so $A_{>x} = \{w \in A : w >_{\text{lex}} x\} = A \setminus A_{<x}$ is c.e. hence $y := \inf \bigcup_{w \in A_{>x}} [w]$ is right-c.e. As x is not computable and $x \leq y$, one has $x < y$ and we get the result as $[x, y)$ is disjoint from U. ◀

▶ Proposition 6.4. A generic Π^0_1-class without isolated point has no computable member.

Proof. Let x be computable. Consider the collection $\mathcal{U} = \{P : x \notin P\}$. $\mathcal{U}$ is an effective open set in the space $(\text{CL}(2^{\mathbb{N}}), \tau_{\text{hit-or-miss}})$ (and even in the topology τ_{miss}). $\mathcal{U}$ is dense and better: for every P without isolated point, there exist $Q \subseteq P$ in $\mathcal{U}$ arbitrarily close to P, so $\mathcal{U}$ is dense below P (here the specialization order is the reverse inclusion). As a result, if P is a generic Π^0_1-class without isolated point then P belongs to $\mathcal{U}$, i.e. $x \notin P$. ◀

Acknowledgements. The author wishes to thank Peter Gács, Emmanuel Jeandel and Cristóbal Rojas for discussions on the subject and helpful comments on a draft of the paper, Christopher Porter for suggesting Example 16 and the anonymous referees for their useful comments.

References

1 V. Brattka. Computable invariance. *Theor. Comput. Sci.*, 210(1):3–20, 1999.
2 M. Braverman and M. Yampolsky. *Computability of Julia Sets*. Springer, 2008.
3 R. G. Downey and G. LaForte. Presentations of computably enumerable reals. *Theor. Comput. Sci.*, 284(2):539–555, 2002.
4 M. Hoyrup. Randomness and the ergodic decomposition. In *CiE*, volume 6735 of *LNCS*, pages 122–131. Springer, 2011.
5 M. Ingrassia. *P-genericity for Recursively Enumerable Sets*. PhD thesis, University of Illinois at Urbana-Champaign, 1981.
6 J. Jockusch, Carl G. and D. B. Posner. Double jumps of minimal degrees. *J. Symb. Log.*, 43(4):715–724, 1978.
7 C. G. Jockush. Degrees of generic sets. In *Recursion Theory: its Generalisations and Applications*, pages 110–139. Cambridge University Press, 1980.
8 J. S. Miller. Degrees of unsolvability of continuous functions. *J. Symb. Log.*, 69(2):555–584, 2004.
9 J. Myhill. Category methods in recursion theory. *Pac. J. Math.*, 11:1479–1486, 1961.
10 J. Myhill. A recursive function, defined on a compact interval and having a continuous derivative that is not recursive. *Michigan Math. J.*, 18(2):97–98, 1971.
11 A. Nies. *Computability and randomness*. Oxford logic guides. Oxford University Press, 2009.
12 M. B. Pour-El and J. I. Richards. *Computability in Analysis and Physics*. Perspectives in Mathematical Logic. Springer, Berlin, 1989.
13 H. J. Rogers. *Theory of Recursive Functions and Effective Computability*. MIT Press, Cambridge, MA, USA, 1987.
14 A. L. Selman. Arithmetical reducibilities I. *Math. Log. Quart.*, 17(1):335–350, 1971.
15 F. Stephan and G. Wu. Presentations of K-trivial reals and Kolmogorov complexity. In *CiE*, volume 3526 of *LNCS*, pages 461–469. Springer, 2005.
16 V. V. V'yugin. Effective convergence in probability and an ergodic theorem for individual random sequences. *SIAM Theory of Probability and Its Applications*, 42(1):39–50, 1997.
17 K. Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.

Ehrenfeucht-Fraïssé Games on Omega-Terms

Martin Huschenbett[1] **and Manfred Kufleitner**[2]

1 **Institut für Theoretische Informatik**
Technische Universität Ilmenau, Germany
martin.huschenbett@tu-ilmenau.de

2 **Institut für Formale Methoden in der Informatik***
Universität Stuttgart, Germany
kufleitner@fmi.uni-stuttgart.de

Abstract

Fragments of first-order logic over words can often be characterized in terms of finite monoids or finite semigroups. Usually these algebraic descriptions yield decidability of the question whether a given regular language is definable in a particular fragment. An effective algebraic characterization can be obtained from identities of so-called omega-terms. In order to show that a given fragment satisfies some identity of omega-terms, one can use Ehrenfeucht-Fraïssé games on word instances of the omega-terms. The resulting proofs often require a significant amount of book-keeping with respect to the constants involved. In this paper we introduce Ehrenfeucht-Fraïssé games on omega-terms. To this end we assign a labeled linear order to every omega-term. Our main theorem shows that a given fragment satisfies some identity of omega-terms if and only if Duplicator has a winning strategy for the game on the resulting linear orders. This allows to avoid the book-keeping.

As an application of our main result, we show that one can decide in exponential time whether all aperiodic monoids satisfy some given identity of omega-terms, thereby improving a result of McCammond (Int. J. Algebra Comput., 2001).

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.4.3 Formal Languages

Keywords and phrases regular language, first-order logic, finite monoid, Ehrenfeucht-Fraïssé games, pseudoidentity

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.374

1 Introduction

By combining a result of McNaughton and Papert [12] with Schützenberger's characterization of star-free languages [16], a given language over finite words is definable in first-order logic if and only if its syntactic monoid is finite and aperiodic. The implication from left to right can be shown using Ehrenfeucht-Fraïssé games, see e.g. [17]. A similar result for two-variable first-order logic FO^2 was obtained by Thérien and Wilke [19]: A language is definable in FO^2 if and only if its syntactic monoid belongs to the variety **DA**. Both the variety **DA** and the class of finite aperiodic monoids can be defined using identities of omega-terms. Roughly speaking, omega-terms are words equipped with an additional operation, the ω-power. If M is a finite monoid, then there exists a positive integer ω_M such that $u^{\omega_M} = (u^{\omega_M})^2$ for all $u \in M$. We call u^{ω_M} the *idempotent* generated by u. Every mapping $h : \Lambda \to M$ uniquely extends to omega-terms over the alphabet Λ by setting $h(uv) = h(u)h(v)$ and $h(u^\omega) = h(u)^{\omega_M}$. Now,

* The second author was supported by the German Research Foundation (DFG) under grant DI 435/5-1 and by the Technische Universität München, Germany.



© Martin Huschenbett and Manfred Kufleitner;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 374–385



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



the monoid M satisfies an identity $u = v$ of omega-terms u and v over Λ if for every mapping $h : \Lambda \to M$ we have $h(u) = h(v)$. A finite monoid is *aperiodic* if and only if it satisfies $a^\omega = a^\omega a$, and it is in **DA** if and only if it satisfies $(abc)^\omega b(abc)^\omega = (abc)^\omega$, see e.g. [15]. Showing that some first-order fragment $\mathcal{F}$ satisfies an identity $u = v$ of omega-terms u, v usually works as follows. Suppose $\mathcal{F}$ does not satisfy $u = v$. Then there exists a formula $\varphi \in \mathcal{F}$ such that the syntactic monoid of $L(\varphi)$ does not satisfy $u = v$. The depth n of the formula φ defines an n-round Ehrenfeucht-Fraïssé game on instances of u and v (i.e., on finite words which are obtained by replacing the ω-powers by fixed positive integers depending on n). Giving a winning strategy for Duplicator yields a contradiction, thereby showing that $\mathcal{F}$ satisfies $u = v$. Usually, playing the game on u and v involves some non-trivial book-keeping since one has to formalize intuitive notions such as positions being near to one another or being close to some border. For first-order logic and for FO^2 the book-keeping is still feasible [17, 5] whereas for other fragments such as the quantifier alternation inside FO^2 this task becomes much more involved (and therefore other techniques are applied [10, 18]).

Instead of defining new instances of a given omega-term depending on the fragment and the number of rounds in the Ehrenfeucht-Fraïssé game, we give a single instance which works for all fragments of first-order logic and any number of rounds. In addition, we allow an infinite number of rounds. The fragments we consider in this paper rely on an abstract notion of logical fragments as introduced in [9]. We show that a fragment $\mathcal{F}$ satisfies an identity of omega-terms if and only if Duplicator has a winning strategy for the Ehrenfeucht-Fraïssé game for $\mathcal{F}$ on the instances of the omega-terms. These instances are labeled linear orders which, in general, are not finite words.

An obvious application of our main result is the simplification of proofs showing that some fragment $\mathcal{F}$ satisfies a given identity of omega-terms. The main reason is that with this new approach one can avoid the book-keeping. It is slightly less straightforward that one can use this approach for solving word problems for omega-terms over varieties of finite monoids. Let $\mathbf{V}$ be a variety of finite monoids. Then the word problem for omega-terms over $\mathbf{V}$ is the following: Given two omega-terms u and v, does every monoid in $\mathbf{V}$ satisfy the identity $u = v$? This problem was solved for various varieties, see e.g. [2, 11, 13]. Using our main result, one approach to solving such word problems is as follows. First, find a logical fragment for $\mathbf{V}$. Second, find a winning strategy for Duplicator on omega-terms satisfied by this fragment. Third, use this winning strategy for finding the desired decision algorithm. In the case of aperiodic monoids, we use this scheme for improving the decidability result of McCammond [11] by showing that the word problem for omega-terms over aperiodic monoids is solvable in exponential time.

Historically, the greek letter ω is used for two different things which are frequently used thoughout this paper: First, the idempotent power of an element and second, the smallest infinite ordinal. In order to avoid confusion in our presentation, we chose to follow the approach of Perrin and Pin [14] by using π instead of ω to denote idempotent powers. In particular, we will use the exponent π in omega-terms which is why we will call them π-terms in the remainder of this paper.

2 Preliminaries

As mentioned above, one of the central notions in this paper are so called π-terms. In order to make their interpretation by several semantics possible in a uniform way, we follow an algebraic approach. A *π-algebra* is a structure $(U, \cdot, {}^\pi)$ comprised of an associative binary operation $\cdot$ and a unary operation ${}^\pi$ on a carrier set U. The application of $\cdot$ is usually written

as juxtaposition, i.e., $uv = u \cdot v$, and the application of $^\pi$ as u^π. A *π-term* is an arbitrary element of the free π-algebra T_Λ generated by Λ, where Λ *is a countably infinite set which is fixed for the rest of this paper*. We also use this set as a universe for letters (of alphabets).

Monoids as π-Algebras. Let M be a monoid. For any $k \geq 1$ we extend M to a π-algebra, called *k-power algebra on M*, by defining $u^\pi = u^k$ for $u \in M$. Suppose that M is finite. An element $u \in M$ is *idempotent* if $u^2 = u$. We extend M to another π-algebra, called *idempotency algebra on M*, by defining u^π for $u \in M$ to be the unique idempotent element in the set $\{\, u^k \mid k \geq 1 \,\}$. In fact, there are infinitely many $k \geq 1$, called *idempotency exponents of M*, such that for each $u \in M$ the element u^k is idempotent, i.e., the k-power algebra and the idempotency algebra on M coincide. An identity $s = t$ of π-terms $s, t \in T_\Lambda$ *holds in M* if every π-algebra morphism h from T_Λ into the idempotency algebra on M satisfies $h(s) = h(t)$.

The set of all *finite words* over an alphabet $A \subseteq \Lambda$ is A^*. Let $L \subseteq A^*$ be a language over a finite alphabet $A \subseteq \Lambda$. The *syntactic congruence* of L is the equivalence relation $\equiv_L$ on A^* defined by $u \equiv_L v$ if $xuy \in L$ is equivalent to $xvy \in L$ for all $x, y \in A^*$. In fact, $\equiv_L$ is a monoid congruence on A^*. The quotient monoid $M_L = A^*/\equiv_L$ is called *syntactic monoid* of L. It is finite precisely if L is regular. Suppose that L is regular and let $k \geq 1$ be an idempotency exponent of M_L. Then the map sending each $w \in A^*$ to its $\equiv_L$-class is a π-algebra morphism from the k-power algebra on A^* onto the idempotency algebra on M_L. Thus, any identity $s = t$ of π-terms $s, t \in T_\Lambda$ holds in M_L if and only if every π-algebra morphism h from T_Λ into the k-power algebra on A^* satisfies $h(s) \equiv_L h(t)$.

Generalized Words. The third semantic domain we consider is the class of generalized words. A *generalized word* (over Λ) is a triple $u = (P_u, \leq_u, \ell_u)$ comprised of a (possibly empty) linear ordering $(P_u, \leq_u)$ being labeled by a map $\ell_u \colon P_u \to \Lambda$. The set $\mathrm{dom}(u) = P_u$ is the *domain* of u, its elements are called *positions* of u. We write $u(p)$ instead of $\ell_u(p)$ for $p \in P_u$. The *order type* of u is the isomorphism type of $(P_u, \leq_u)$. We regard any finite word $w = a_1 \dots a_n \in \Lambda^*$ as a generalized word by defining $\mathrm{dom}(w) = [1, n]$, $\leq_w$ as the natural order on $[1, n]$ and $w(k) = a_k$ for $k \in [1, n]$. On that view, generalized words indeed generalize finite words. As of now, we mean "generalized word" when writing just "word". Two words u and v are *isomorphic* if there exists an isomorphism f of linear orderings from $(\mathrm{dom}(u), \leq_u)$ to $(\mathrm{dom}(v), \leq_v)$ such that $u(p) = v(f(p))$ for all $p \in \mathrm{dom}(u)$. *We identify isomorphic words.* We denote the set of all (isomorphism classes of) *countable* words by Λ^{LO}. The exponent LO is for *linear order*. We regard Λ^* as a subset of Λ^{LO}.

Let $u, v \in \Lambda^{\mathsf{LO}}$ be two words. Their *concatenation* is the word $uv \in \Lambda^{\mathsf{LO}}$ defined by $\mathrm{dom}(uv) = \mathrm{dom}(u) \uplus \mathrm{dom}(v)$, $\leq_{uv}$ makes all positions of u smaller than those of v and retains the respective orders inside u and inside v, and $(uv)(p)$ is $u(p)$ if $p \in \mathrm{dom}(u)$ and $v(p)$ if $p \in \mathrm{dom}(v)$. The set Λ^{LO} with concatenation forms a monoid. On finite words this concatenation coincides with the usual definition and hence Λ^* is a submonoid of Λ^{LO}.

It is customary to regard $n \in \mathbb{N}$ also as the order type of the natural linear ordering on $[1, n]$. We extend the notion of the n-power algebra on Λ^{LO} to arbitrary countable order types τ as follows. Let $(T, \leq_T)$ be a linear ordering of isomorphism type τ. The *τ-power* of any word $u \in \Lambda^{\mathsf{LO}}$ is the word $u^\tau \in \Lambda^{\mathsf{LO}}$ defined by $\mathrm{dom}(u^\tau) = \mathrm{dom}(u) \times T$, $(p, t) \leq_{u^\tau} (p', t')$ if $t <_T t'$ or if $t = t'$ and $p \leq_u p'$, and $(u^\tau)(p, t) = u(p)$. We extend the monoid Λ^{LO} to a π-algebra, called *τ-power algebra on Λ^{LO}*, by defining $u^\pi = u^\tau$ for $u \in \Lambda^{\mathsf{LO}}$. We denote by $[\![\cdot]\!]_\tau$ the unique π-algebra morphism from T_Λ into this π-algebra mapping each $a \in \Lambda$ to the word consisting of a single position which is labeled by a. Finally, notice that there are two definitions of the n-power algebra on Λ^{LO} around, but actually they coincide.

Logic over Words. *For the rest of this paper, we fix a countably infinite set $\mathbb{V}$ of (first-order) variables $x, y, z, \ldots$.* The syntax of first-order logic over words is given by

$$\varphi ::= \top \mid \bot \mid \mathsf{empty} \mid x = y \mid \lambda(x) = a \mid x < y \mid x \leq y \mid \mathsf{suc}(x, y) \mid \mathsf{min}(x) \mid \mathsf{max}(x) \mid \neg\varphi \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid \exists x\, \varphi \mid \forall x\, \varphi\,,$$

where $x, y \in \mathbb{V}$ and $a \in \Lambda$. The set of all formulae is denoted by FO. Brackets can be omitted when originating no ambiguity. The *free variables* $\mathrm{FV}(\varphi)$ of a formula $\varphi \in \mathsf{FO}$ are defined as usual. A *sentence* is a formula φ with $\mathrm{FV}(\varphi) = \emptyset$.

We only give a brief sketch of the semantics of formulae. Let $\mathbb{X} \subseteq \mathbb{V}$ be a *finite* set of variables. An $\mathbb{X}$-*valuation on* u is a pair $\langle u, \alpha \rangle$ consisting of a word $u \in \Lambda^{\mathsf{LO}}$ and a map $\alpha\colon \mathbb{X} \to \mathrm{dom}(u)$. It is a model of a formula $\varphi \in \mathsf{FO}$ with $\mathrm{FV}(\varphi) \subseteq \mathbb{X}$, in symbols $\langle u, \alpha \rangle \models \varphi$, if u satisfies the formula φ under the following assumptions:

- variables range over positions of u and free variables are interpreted according to α,
- $\top$ is always satisfied, $\bot$ never, and empty only in case $\mathrm{dom}(u) = \emptyset$,
- the function symbol λ is interpreted by the labeling map $\ell_u\colon \mathrm{dom}(u) \to \Lambda$ and
- the predicates $<$, $\leq$, suc, min and max are evaluated in the linear ordering $(\mathrm{dom}(u), \leq_u)$, where $\mathsf{suc}(x, y)$ means that y is the immediate successor of x.

We identify any word $u \in \Lambda^{\mathsf{LO}}$ with the only $\emptyset$-valuation on u, namely $\langle u, \emptyset \rangle$ with $\emptyset$ also denoting the empty map. Thus, for sentences φ the meaning of $u \models \varphi$ is well-defined. Let $A \subseteq \Lambda$ be a finite alphabet and $\varphi \in \mathsf{FO}$ a sentence. Due to the result of Büchi, Elgot, and Trakhtenbrot [4, 7, 20], the *language over A defined by* φ, namely $L_A(\varphi) = \{\, w \in A^* \mid w \models \varphi \,\}$, is regular. A language $L \subseteq A^*$ is *definable in* a class $\mathcal{F} \subseteq \mathsf{FO}$ of formulae if there exists a sentence $\varphi \in \mathcal{F}$ such that $L = L_A(\varphi)$.

Fragments. We reintroduce (a slight variation of) the notion of a fragment as a class of formulae obeying natural syntactic closure properties [9]. A *context* is a formula μ with a unique occurrence of an additional constant predicate $\circ$ which is intended to be a placeholder for another formula $\varphi \in \mathsf{FO}$. The result of replacing $\circ$ in μ by φ is denoted by $\mu(\varphi)$. Unfortunately, the notion of a fragment as defined in [9, Definition 1] is slightly too weak for our purposes. We require one more *natural* syntactic closure property, namely condition **4.** in Definition 2.1 below. Condition **6.** is missing in the exposition in [9]. Nevertheless, since we only add requirements, every fragment in our sense is still a fragment in the sense of [9].

▶ **Definition 2.1.** A *fragment* is a non-empty set of formulae $\mathcal{F} \subseteq \mathsf{FO}$ such that for all contexts μ, formulae $\varphi, \psi \in \mathsf{FO}$, $a \in \Lambda$ and $x, y \in \mathbb{V}$ the following conditions are satisfied:

1. If $\mu(\varphi) \in \mathcal{F}$, then $\mu(\top) \in \mathcal{F}$, $\mu(\bot) \in \mathcal{F}$, and $\mu(\lambda(x) = a) \in \mathcal{F}$.
2. $\mu(\varphi \vee \psi) \in \mathcal{F}$ if and only if $\mu(\varphi) \in \mathcal{F}$ and $\mu(\psi) \in \mathcal{F}$.
3. $\mu(\varphi \wedge \psi) \in \mathcal{F}$ if and only if $\mu(\varphi) \in \mathcal{F}$ and $\mu(\psi) \in \mathcal{F}$.
4. If $\mu(\neg\neg\varphi) \in \mathcal{F}$, then $\mu(\varphi) \in \mathcal{F}$.
5. If $\mu(\exists x\, \varphi) \in \mathcal{F}$ and $x \notin \mathrm{FV}(\varphi)$, then $\mu(\varphi) \in \mathcal{F}$.
6. If $\mu(\forall x\, \varphi) \in \mathcal{F}$ and $x \notin \mathrm{FV}(\varphi)$, then $\mu(\varphi) \in \mathcal{F}$.

It is *closed under negation* if the following condition is satisfied:

7. If $\varphi \in \mathcal{F}$, then $\neg\varphi \in \mathcal{F}$.

It is *order-stable* if the following condition is satisfied:

8. $\mu(x < y) \in \mathcal{F}$ if and only if $\mu(x \leq y) \in \mathcal{F}$.

It is suc-*stable* if the following two conditions are satisfied:

9. If $\mu(\mathsf{suc}(x, y)) \in \mathcal{F}$, then $\mu(x = y) \in \mathcal{F}$, $\mu(\mathsf{max}(x)) \in \mathcal{F}$ and $\mu(\mathsf{min}(y)) \in \mathcal{F}$.
10. If $\mu(\mathsf{min}(x)) \in \mathcal{F}$ or $\mu(\mathsf{max}(x)) \in \mathcal{F}$, then $\mu(\mathsf{empty}) \in \mathcal{F}$.

Table 1 A single round of the $\mathcal{F}$-game in configuration $S = (\mathcal{G}, \langle u, \alpha\rangle, \langle v, \beta\rangle)$.

1. Spoiler chooses $\mathsf{Q}x$	**2.** Spoiler chooses q in	**3.** Duplicator chooses r in	**4.** resulting configuration $S[\mathsf{Q}x, q, r]$
$\mathsf{Q}x = \exists x$	$\mathrm{dom}(u)$	$\mathrm{dom}(v)$	$(\mathcal{G}/\exists x, \langle u, \alpha[x/q]\rangle, \langle v, \beta[x/r]\rangle)$
$\mathsf{Q}x = \forall x$	$\mathrm{dom}(v)$	$\mathrm{dom}(u)$	$(\mathcal{G}/\forall x, \langle u, \alpha[x/r]\rangle, \langle v, \beta[x/q]\rangle)$
$\mathsf{Q}x = \neg\exists x$	$\mathrm{dom}(v)$	$\mathrm{dom}(u)$	$(\mathcal{G}/\neg\exists x, \langle v, \beta[x/q]\rangle, \langle u, \alpha[x/r]\rangle)$
$\mathsf{Q}x = \neg\forall x$	$\mathrm{dom}(u)$	$\mathrm{dom}(v)$	$(\mathcal{G}/\neg\forall x, \langle v, \beta[x/r]\rangle, \langle u, \alpha[x/q]\rangle)$

Examples for fragments in this sense include all classes of formulae which are obtained from full first-order logic FO by limiting the quantifier depth (e.g., FO_n), the number of quantifier alternations (e.g., Σ_n and Π_n), the number of quantified variables (e.g., FO^m), the available predicates (e.g., first-order logic $\mathsf{FO}[<]$ without min, max, suc) or combinations of those.

The *quantifier depth* $\mathrm{qd}(\varphi)$ of a formula $\varphi \in \mathsf{FO}$ is defined as usual. A fragment $\mathcal{F}$ has *bounded quantifier depth* if there is an $n \in \mathbb{N}$ such that $\mathrm{qd}(\varphi) \leq n$ for all $\varphi \in \mathcal{F}$. For any $n \in \mathbb{N}$ and every fragment $\mathcal{F}$ the set $\mathcal{F}_n = \{\, \varphi \in \mathcal{F} \mid \mathrm{qd}(\varphi) \leq n \,\}$ is a fragment of bounded quantifier depth. Moreover, the fragment $\mathcal{F}_n$ is order-stable (respectively suc-stable) in case $\mathcal{F}$ has the according property.

3 Ehrenfeucht-Fraïssé Games for Arbitrary Fragments

In this section, we introduce an Ehrenfeucht-Fraïssé game for arbitrary fragments of first-order logic on generalized words and develop its basic theory. Before we can describe this game, we need to define some notation. In the following, we call the "negated quantifiers" $\neg\exists$ and $\neg\forall$ also *quantifiers*. The set of all quantifiers (in this sense) is $\mathcal{Q} = \{\exists, \forall, \neg\exists, \neg\forall\}$. For a quantifier $\mathsf{Q} \in \mathcal{Q}$ and a variable $x \in \mathbb{V}$, the *reduct* of $\mathcal{F}$ by $\mathsf{Q}x$ is the set

$$\mathcal{F}/\mathsf{Q}x = \{\, \varphi \in \mathsf{FO} \mid \mathsf{Q}x\,\varphi \in \mathcal{F} \,\} \ .$$

Whenever this set is not empty, it is a fragment as well.

Now, let $\mathcal{F}$ be a fragment and u, v two words over Λ. We are about to describe the *$\mathcal{F}$-game on (u, v)*. A *configuration* of this game is a triple $S = (\mathcal{G}, \langle u, \alpha\rangle, \langle v, \beta\rangle)$ comprised of a non-empty, iterated reduct $\mathcal{G}$ of $\mathcal{F}$ and $\mathbb{X}$-valuations $\langle u, \alpha\rangle$ and $\langle v, \beta\rangle$ on u and v for the same arbitrary *finite* subset $\mathbb{X} \subseteq \mathbb{V}$. To emphasize the set $\mathbb{X}$, we also speak of an *$\mathbb{X}$-configuration*. The game starts in the $\emptyset$-configuration $(\mathcal{F}, u, v)$ and goes on for an arbitrary—possibly infinite—number of rounds. Assuming that the game is currently in configuration $S = (\mathcal{G}, \langle u, \alpha\rangle, \langle v, \beta\rangle)$, a single round proceeds as follows (see Table 1 for a summary of this procedure):

1. Spoiler chooses a quantifier $\mathsf{Q} \in \mathcal{Q}$ and a variable $x \in \mathbb{V}$ such that $\mathcal{G}/\mathsf{Q}x \neq \emptyset$.
2. Spoiler chooses a position q (like "quest") in the domain of u if $\mathsf{Q} \in \{\exists, \neg\forall\}$ or in the domain of v if $\mathsf{Q} \in \{\forall, \neg\exists\}$.
3. Duplicator chooses a position r (like "reply") in the domain of the other word.
4. The resulting configuration $S[\mathsf{Q}x, q, r]$ consists of the reduct $\mathcal{G}/\mathsf{Q}x$ and the extension of the valuations $\langle u, \alpha\rangle$ and $\langle v, \beta\rangle$ by variable x at positions q and r, accordingly. Whenever Q is a negated quantifier, the role of the two extended valuations is additionally interchanged (see the last column of Table 1 for a formal definition of $S[\mathsf{Q}x, q, r]$).

Whenever a player cannot perform a choice because $\mathcal{G}$ contains no more quantified formulae or the domain of the according word is empty, the game immediately stops and

the other player wins. Besides the inability of Duplicator to move, the winning condition for Spoiler is to reach an $\mathbb{X}$-configuration $(\mathcal{G}, \langle u, \alpha\rangle, \langle v, \beta\rangle)$ such that there exists a literal $\varphi \in \mathcal{G}$ with $\mathrm{FV}(\varphi) \subseteq \mathbb{X}$ and $\langle u, \alpha\rangle \models \varphi$ but $\langle v, \beta\rangle \not\models \varphi$; in this case the game immediately stops. Duplicator's goal is simply to prevent Spoiler from winning. In particular, Duplicator wins all games that go on forever. Due to this circumstance, the $\mathcal{F}$-game is determined, i.e., either Spoiler or Duplicator has a winning strategy on (u, v).

▶ Remark. The $\mathcal{F}$-game is quite asymmetric since Spoiler is not allowed to choose before his first move whether he wants to play on (u, v) or on (v, u). This may lead to the situation that he has a winning strategy on (u, v) but not on (v, u) or vice versa. This asymmetry is owed to the circumstance that $\mathcal{F}$ might not be closed under negation. As soon as $\mathcal{F}$ is assumed to be closed under negation this phenomenon disappears and Spoiler has a winning strategy on (u, v) if and only if he has a winning strategy on (v, u). We also note that, in general, the winning condition for Spoiler can be asymmetric since it does not rely on any notion of isomorphism. ◀

If the quantifier depth of a fragment $\mathcal{F}$ is bounded by $n \in \mathbb{N}$, the $\mathcal{F}$-game lasts at most n rounds. In particular, for any fragment $\mathcal{F}$ the $\mathcal{F}_n$-game can be regarded as an n-round version of the $\mathcal{F}$-game. For instance, the FO_n-game resembles the classical n-round Ehrenfeucht-Fraïssé game. The following result is an adaption of the Ehrenfeucht-Fraïssé Theorem to the $\mathcal{F}$-game for fragments of bounded quantifier depth.

▶ **Theorem 3.1.** *Let $\mathcal{F}$ be a fragment of bounded quantifier depth. For all words $u, v \in \Lambda^{\mathsf{LO}}$ the following are equivalent:*

1. *$u \models \varphi$ implies $v \models \varphi$ for all sentences $\varphi \in \mathcal{F}$ and*
2. *Duplicator has a winning strategy in the $\mathcal{F}$-game on (u, v).*

A proof of this theorem can easily be achieved along the lines of a proof of the classical version, cf. [8]. In fact, such a proof reveals that the implication "**2.** $\Rightarrow$ **1.**" even holds if the quantifier depth of $\mathcal{F}$ is not bounded. In contrast, the implication "**1.** $\Rightarrow$ **2.**" substantially relies the boundedness of the quantifier depth of $\mathcal{F}$. If ζ denotes the order type of the integers $\mathbb{Z}$, then Duplicator has a winning strategy in the FO_n-game on $(a^\zeta, a^{\zeta+\zeta})$ for each $n \in \mathbb{N}$ and hence $a^\zeta \models \varphi$ implies $a^{\zeta+\zeta} \models \varphi$ for all sentences $\varphi \in \mathsf{FO}$, but Spoiler has a winning strategy in the infinite FO-game on $(a^\zeta, a^{\zeta+\zeta})$.

The objective of the remainder of this section is to identify additional requirements on $\mathcal{F}$ and/or u, v such that the boundedness of the quantifier depth can be omitted. It turns out that the property introduced in Definition 3.2 below in combination with suc-stability of the fragment is sufficient for this purpose and still allows for the applications in Section 4. The order types of the sets $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{Z}_{<0}$ ordered naturally are denoted by ω, ζ, η and ω^*, respectively. Then $\omega + \zeta \cdot \eta + \omega^*$ is the order type of the word $a^\omega\big(a^\zeta\big)^\eta a^{\omega^*}$, where $a \in \Lambda$.

▶ **Definition 3.2.** Let $\varrho = \omega + \zeta \cdot \eta + \omega^*$. A word $u \in \Lambda^{\mathsf{LO}}$ is *ϱ-rational* if it can be constructed from the finite words in Λ^{LO} using the operations of concatenation and ϱ-power only or, equivalently, if $u = [\![t]\!]_\varrho$ for some π-term $t \in T_\Lambda$.

▶ **Theorem 3.3.** *Let $\mathcal{F}$ be a suc-stable fragment. For all ϱ-rational words $u, v \in \Lambda^{\mathsf{LO}}$ the following are equivalent:*

1. *$u \models \varphi$ implies $v \models \varphi$ for all sentences $\varphi \in \mathcal{F}$ and*
2. *Duplicator has a winning strategy in the $\mathcal{F}$-game on (u, v).*

As already mentioned, the implication "**2.** $\Rightarrow$ **1.**" can be shown using the very same proof as for the according implication of Theorem 3.1. The key idea behind proving the implication

"**1. ⇒ 2.**" is as follows: Theorem 3.1 provides us for each $n \in \mathbb{N}$ with a winning strategy for Duplicator in the $\mathcal{F}_n$-game on (u, v). A winning strategy in the $\mathcal{F}$-game is obtained by defining a limit of all those strategies. This limit process relies on the ϱ-rationality of the underlying words and is formalized by Lemma 3.6 below. A major ingredient of its proof is Proposition 3.4.

In order to keep notation concise, we abbreviate the circumstance that Duplicator has a winning strategy in a configuration $S = (\mathcal{F}, \langle u, \alpha\rangle, \langle v, \beta\rangle)$ by $\langle u, \alpha\rangle \lesssim_{\mathcal{F}} \langle v, \beta\rangle$. Since the $\mathcal{F}$-game is determined, $\langle u, \alpha\rangle \not\lesssim_{\mathcal{F}} \langle v, \beta\rangle$ hence means that Spoiler has a winning strategy in S. The relation $\lesssim_{\mathcal{F}}$ is reflexive and transitive, i.e., a preorder on the set of all configurations. It induces an equivalence $\approx_{\mathcal{F}}$ defined by $\langle u, \alpha\rangle \approx_{\mathcal{F}} \langle v, \beta\rangle$ if $\langle u, \alpha\rangle \lesssim_{\mathcal{F}} \langle v, \beta\rangle$ and $\langle v, \beta\rangle \lesssim_{\mathcal{F}} \langle u, \alpha\rangle$.

▶ **Proposition 3.4.** *Let $\mathcal{F}$ be a* suc*-stable fragment, $k \in \mathbb{N}$ and $\langle u_i, \alpha_i\rangle, \langle v_i, \beta_i\rangle$ $\mathbb{X}_i$-valuations with mutually disjoint $\mathbb{X}_i$ for $i \in [1, k]$. If $\langle u_i, \alpha_i\rangle \lesssim_{\mathcal{F}} \langle v_i, \beta_i\rangle$ for each $i \in [1, k]$, then $\langle u_1 \cdots u_k, \alpha_1 \cup \cdots \cup \alpha_k\rangle \lesssim_{\mathcal{F}} \langle v_1 \cdots v_k, \beta_1 \cup \cdots \cup \beta_k\rangle$.* ◀

▶ **Lemma 3.5.** *Let $\mathcal{F}$ be a* suc*-stable fragment with quantifier depth bounded by $n \in \mathbb{N}$ and $u, v \in \Lambda^{\mathsf{LO}}$. If $u \lesssim_{\mathcal{F}} v$, then $u^m \lesssim_{\mathcal{F}} v^\varrho$ and $u^\varrho \lesssim_{\mathcal{F}} v^m$ for all $m \geq 2^{n+1} - 1$.* ◀

The following lemma formalizes the limit process mentioned above.

▶ **Lemma 3.6.** *Let $\mathcal{F}$ be a* suc*-stable fragment, $x \in \mathbb{V}$ and $\langle u, \alpha\rangle$ an $\mathbb{X}$-valuation on a ϱ-rational word $u \in \Lambda^{\mathsf{LO}}$. For every infinite sequence $(q_i)_{i\in\mathbb{N}} \in \mathrm{dom}(u)^{\mathbb{N}}$ there exists a position $q \in \mathrm{dom}(u)$ such that for all $n \in \mathbb{N}$ there are arbitrarily large $i \in \mathbb{N}$ with $\langle u, \alpha[x/q_i]\rangle \lesssim_{\mathcal{F}_n} \langle u, \alpha[x/q]\rangle$.*

Proof. To simplify notation, we call a position q with the property above a $\langle u, \alpha\rangle$*-limit point* of the sequence $(q_i)_{i\in\mathbb{N}}$ (w.r.t. to $\mathcal{F}$ and x). Using this terminology, we have to show that every sequence $(q_i)_{i\in\mathbb{N}} \in \mathrm{dom}(u)^{\mathbb{N}}$ possesses a $\langle u, \alpha\rangle$-limit point. Since neither $\alpha[x/q_i]$ nor $\alpha[x/q]$ would depend on $\alpha(x)$, we may simply assume that $x \notin \mathbb{X}$. We proceed by induction on the ϱ-rational construction of u.

Base case: u is finite. Since $\mathrm{dom}(u)$ is finite, there exists a $q \in \mathrm{dom}(u)$ such that $q = q_i$ for infinitely many $i \in \mathbb{N}$. Thus, q is a $\langle u, \alpha\rangle$-limit point of $(q_i)_{i\in\mathbb{N}}$.

Inductive step 1: $u = v_1 v_2$ with ϱ-rational words v_1, v_2. The valuation $\langle u, \alpha\rangle$ splits into valuations $\langle v_1, \beta_1\rangle$ and $\langle v_2, \beta_2\rangle$ such that $\alpha = \beta_1 \cup \beta_2$. For either $\ell = 1$ or $\ell = 2$ we have $q_i \in \mathrm{dom}(v_\ell)$ for infinitely many $i \in \mathbb{N}$. Let I be the set of these i. By the induction hypothesis, there is a $\langle v_\ell, \beta_\ell\rangle$-limit point $q \in \mathrm{dom}(v_\ell)$ of the subsequence $(q_i)_{i\in I}$. Proposition 3.4 implies that q is also a $\langle u, \alpha\rangle$-limit point of $(q_i)_{i\in\mathbb{N}}$.

We split the inductive step for ϱ-powers in two parts, one for $\mathbb{X} = \emptyset$ and another for $\mathbb{X} \neq \emptyset$.

Inductive step 2: $u = v^\varrho$ with a ϱ-rational v and $\mathbb{X} = \emptyset$. Let $(P, \leq_P)$ be a linear ordering of isomorphism type ϱ such that $\mathrm{dom}(u) = \mathrm{dom}(v) \times P$. For each $i \in \mathbb{N}$ we write $q_i = (s_i, p_i)$. For every $p \in P$ let $\overleftarrow{\tau}_p$ and $\vec{\tau}_p$ be the order types of the suborders of $(P, \leq_P)$ induced by the open intervals $(-\infty, p)$ and $(p, +\infty)$, respectively. Then $\varrho = \overleftarrow{\tau}_p + 1 + \vec{\tau}_p$. Due to the nature of ϱ, each of $\overleftarrow{\tau}_p$ and $\vec{\tau}_p$ is either finite or equals ϱ. However, the case that $\overleftarrow{\tau}_p$ and $\vec{\tau}_p$ both are finite at the same time cannot occur. Accordingly, we distinguish three cases:

Case 1: $\overleftarrow{\tau}_{p_i} = \vec{\tau}_{p_i} = \varrho$ for infinitely many $i \in \mathbb{N}$. Let I be the set of these i. By the induction hypothesis, there exists a $\langle v, \emptyset\rangle$-limit point $s \in \mathrm{dom}(v)$ of the subsequence $(s_i)_{i\in I}$. We pick some $j \in I$. Proposition 3.4 reveals that $q = (s, p_j)$ is a $\langle u, \alpha\rangle$-limit point of $(q_i)_{i\in\mathbb{N}}$.

Case 2: $\overleftarrow{\tau}_{p_i}$ is finite and $\vec{\tau}_{p_i} = \varrho$ for infinitely many $i \in \mathbb{N}$. Let I be the set of these i. If there is an order type which occurs infinitely often among the $\overleftarrow{\tau}_{p_i}$ with $i \in I$, the same

argumentation as in Case 1 applies. Henceforth, we assume that such an order type does not exist. By the induction hypothesis, the subsequence $(s_i)_{i\in I}$ possesses a $\langle v, \emptyset\rangle$-limit point $s \in \mathrm{dom}(v)$. Let $p \in P$ be arbitrary with $\overleftarrow{\tau}_p = \vec{\tau}_p = \varrho$. We show that $q = (s, p)$ is a $\langle u, \alpha\rangle$-limit point of $(q_i)_{i\in\mathbb{N}}$.

Let $n \in \mathbb{N}$. Due to the choice of I and s, there are arbitrarily large $i \in I$ such that $\overleftarrow{\tau}_{p_i}$ is of size at least $2^{n+1} - 1$ and $\langle v, \emptyset[x/s_i]\rangle \lesssim_{\mathcal{F}_n} \langle v, \emptyset[x/s]\rangle$. Lemma 3.5 then implies $v^{\overleftarrow{\tau}_{p_i}} \lesssim_{\mathcal{F}_n} v^\varrho$. Since also $v^{\vec{\tau}_{p_i}} \lesssim_{\mathcal{F}_n} v^\varrho$, Proposition 3.4 yields $\langle u, \emptyset[x/q_i]\rangle \lesssim_{\mathcal{F}_n} \langle u, \emptyset[x/q]\rangle$.

Case 3: $\overleftarrow{\tau}_{p_i} = \varrho$ and $\vec{\tau}_{p_i}$ is finite for infinitely many $i \in \mathbb{N}$. Symmetric to Case 2.

Inductive step 3: $u = v^\varrho$ with a ϱ-rational v and $\mathbb{X} \neq \emptyset$. Let $(P, \leq_P)$ be as above. Recall that $\mathbb{X}$ is supposed to be finite. Let $\tilde{p}_1 <_P \cdots <_P \tilde{p}_k$ be an enumeration of all positions $p \in P$ for which there exists a variable $y \in \mathbb{X}$ with $\alpha(y) \in \mathrm{dom}(v) \times \{p\}$. We consider the open intervals $P_0 = (-\infty, \tilde{p}_1)$, $P_\ell = (\tilde{p}_\ell, \tilde{p}_{\ell+1})$ for $\ell \in [1, k-1]$, and $P_k = (\tilde{p}_k, +\infty)$ in $(P, \leq_P)$. For $\ell \in [0, k]$ we let τ_ℓ be the order type of the suborder induced by P_ℓ. Then $\varrho = \tau_0 + 1 + \tau_1 + 1 + \cdots + 1 + \tau_k$ and hence $u = v^{\tau_0} v v^{\tau_1} v \cdots v v^{\tau_k}$. Due to the nature of ϱ, each τ_ℓ is either finite or equals ϱ. Since for every finite τ_ℓ the word v^{τ_ℓ} is the concatenation of τ_ℓ copies of v, the factorization of u above is an alternative ϱ-rational construction of u. This construction has the additional property that α does not map into the ϱ-powers v^ϱ but only in the individual intermediate copies of v. Thus, the induction hypothesis and the inductive steps 1 and 2 above yield the claim. ◀

Now, we are prepared to prove the remaining implication of Theorem 3.3.

Proof of Theorem 3.3, "1. ⇒ 2.". We show that Duplicator can maintain the invariant of staying in configurations which are *good* for her. A configuration $(\mathcal{G}, \langle u, \alpha\rangle, \langle v, \beta\rangle)$ of the $\mathcal{F}$-game on (u, v) is considered to be *good* for Duplicator if $\langle u, \alpha\rangle \lesssim_{\mathcal{G}_n} \langle v, \beta\rangle$ for every $n \in \mathbb{N}$. Statement **1.** and Theorem 3.1 imply that the initial configuration $(\mathcal{F}, u, v)$ is good. Moreover, good configurations do not meet Spoiler's winning condition as they particularly satisfy $\langle u, \alpha\rangle \lesssim_{\mathcal{G}_0} \langle v, \beta\rangle$. Consequently, it suffices to provide a strategy for Duplicator which never leaves the set of good configurations since such a strategy is a winning strategy.

Suppose Spoiler chooses the quantifier $\mathsf{Q}x$ and the quest q in a good configuration $(\mathcal{G}, \langle u, \alpha\rangle, \langle v, \beta\rangle)$. We only demonstrate the case $\mathsf{Q} = \exists$, where $q \in \mathrm{dom}(u)$. For every $i \in \mathbb{N}$ we have $\langle u, \alpha\rangle \lesssim_{\mathcal{G}_{i+1}} \langle v, \beta\rangle$ and hence there exists $r_i \in \mathrm{dom}(v)$ such that $\langle u, \alpha[x/q]\rangle \lesssim_{\mathcal{G}_{i+1}/\exists x} \langle v, \beta[x/r_i]\rangle$. Since $\mathcal{G}_{i+1}/\exists x = (\mathcal{G}/\exists x)_i$, this is the same as $\langle u, \alpha[x/q]\rangle \lesssim_{(\mathcal{G}/\exists x)_i} \langle v, \beta[x/r_i]\rangle$. Due to Lemma 3.6 applied to the sequence $(r_i)_{i\in\mathbb{N}}$, there exists $r \in \mathrm{dom}(v)$ such that, for every $n \in \mathbb{N}$, there are arbitrarily large $i \in \mathbb{N}$ with $\langle v, \beta[x/r_i]\rangle \lesssim_{(\mathcal{G}/\exists x)_n} \langle v, \beta[x/r]\rangle$. We show that the configuration $S[\exists x, q, r] = (\mathcal{G}/\exists x, \langle u, \alpha[x/q]\rangle, \langle v, \beta[x/r]\rangle)$ is good again.

Let $n \in \mathbb{N}$. Due to the choice of r, there is an $i \geq n$ with $\langle v, \beta[x/r_i]\rangle \lesssim_{(\mathcal{G}/\exists x)_n} \langle v, \beta[x/r]\rangle$. Above, the position r_i was chosen such that $\langle u, \alpha[x/q]\rangle \lesssim_{(\mathcal{G}/\exists x)_i} \langle v, \beta[x/r_i]\rangle$. Since $n \leq i$, this implies $\langle u, \alpha[x/q]\rangle \lesssim_{(\mathcal{G}/\exists x)_n} \langle v, \beta[x/r_i]\rangle$ and in turn $\langle u, \alpha[x/q]\rangle \lesssim_{(\mathcal{G}/\exists x)_n} \langle v, \beta[x/r]\rangle$. ◀

4 Ehrenfeucht-Fraïssé Games on Identities

Identities play an important role in the study of the expressive power of first-order fragments. A recurring problem is to show that a certain identity of π-terms holds in the syntactic monoid/semigroup of every language definable in the fragment under consideration. Theorems 4.1 and 4.2 below can remarkably simplify this task, as demonstrated at the end of this section. In fact, the two theorems are just slight variations of one another and the sole

reason for having two theorems is that the suc-predicate does not play well with syntactic monoids but only with syntactic semigroups.

▶ **Theorem 4.1.** *Let $\mathcal{F}$ be an order-stable fragment not containing the predicates* suc, min, max *and* empty. *For all π-terms $s, t \in T_\Lambda$ the following are equivalent:*
1. *The identity $s = t$ holds in the syntactic monoid of every language definable in $\mathcal{F}$.*
2. *Duplicator has winning strategies in the $\mathcal{F}$-games on $([\![s]\!]_\varrho, [\![t]\!]_\varrho)$ and $([\![t]\!]_\varrho, [\![s]\!]_\varrho)$.*

▶ **Theorem 4.2.** *Let $\mathcal{F}$ be a* suc*-stable and order-stable fragment. For all π-terms $s, t \in T_\Lambda$ the following are equivalent:*
1. *The identity $s = t$ holds in the syntactic semigroup of every language definable in $\mathcal{F}$ over non-empty words.*
2. *Duplicator has winning strategies in the $\mathcal{F}$-games on $([\![s]\!]_\varrho, [\![t]\!]_\varrho)$ and $([\![t]\!]_\varrho, [\![s]\!]_\varrho)$.* ◀

The main ingredients of the proofs of both theorems are Theorem 3.3 and [9, Proposition 2] which is restated as Proposition 4.3 below.

▶ **Proposition 4.3.** *Let $\mathcal{F}$ be a fragment, $A, B \subseteq \Lambda$ finite alphabets and h a monoid morphism from A^* into B^*. Suppose the following:*
1. *If $\mathcal{F}$ contains the predicate $\leq$ or $<$, then $\mathcal{F}$ is order-stable or $h(A) \subseteq B \cup \{\varepsilon\}$.*
2. *If $\mathcal{F}$ contains the predicate* suc, min, max *or* empty, *then $\varepsilon \notin h(A)$.*

Then $h^{-1}(L)$ is $\mathcal{F}$-definable whenever $L \subseteq B^*$ is $\mathcal{F}$-definable.

Applying this proposition to $\mathcal{F}$-games yields that monoid morphisms satisfying the two conditions above preserve the existence of winning strategies for Duplicator.

▶ **Corollary 4.4.** *Let $\mathcal{F}$, A, B and h be as in Proposition 4.3 satisfying conditions **1.** and **2.**. Moreover, let $\mathcal{F}$ be a* suc*-stable. Then $u \lesssim_\mathcal{F} v$ implies $h(u) \lesssim_\mathcal{F} h(v)$ for all $u, v \in A^*$.*

Proof. Let $u, v \in A^*$ with $u \lesssim_\mathcal{F} v$. Since finite words are ϱ-rational and due to Theorem 3.3, it suffices to show that $h(u) \models \varphi$ implies $h(v) \models \varphi$ for all sentences $\varphi \in \mathcal{F}$. Consider a sentence $\varphi \in \mathcal{F}$. By Proposition 4.3, there is a sentence $\psi \in \mathcal{F}$ such that $L_A(\psi) = h^{-1}\big(L_B(\varphi)\big)$. Altogether, $h(u) \models \varphi$ implies $u \models \psi$ and since $u \lesssim_\mathcal{F} v$ this implies $v \models \psi$ which in turn implies $h(v) \models \varphi$. ◀

The following corollary is an immediate consequence of Proposition 3.4 and Lemma 3.5.

▶ **Corollary 4.5.** *Let $\mathcal{F}$ be a* suc*-stable fragment whose quantifier depth is bounded by $n \in \mathbb{N}$ and let $t \in T_\Lambda$ be a π-term. Then $[\![t]\!]_\varrho \approx_\mathcal{F} [\![t]\!]_m$ for all $m \geq 2^{n+1} - 1$.* ◀

The previous results allow us to show Theorems 4.1 and 4.2. However, since their proofs are as similar as their statements, we only demonstrate the first one.

Proof of Theorem 4.1. Let $A \subseteq \Lambda$ be the finite set containing all $a \in \Lambda$ appearing in s or t. We show both implications separately.

"1. $\Rightarrow$ 2.". By Theorem 3.3, it suffices to show for every sentence $\varphi \in \mathcal{F}$ that $[\![s]\!]_\varrho \models \varphi$ if and only if $[\![t]\!]_\varrho \models \varphi$. Consider a sentence $\varphi \in \mathcal{F}$ and put $n = \mathrm{qd}(\varphi)$. We put $L = L_A(\varphi)$ and let $k \geq 2^{n+1} - 1$ be an idempotency exponent of M_L. We consider an arbitrary π-algebra morphism h from T_A into the k-power algebra on A^* with $h(a) = a$ for each $a \in A$. Because $s = t$ holds in M_L, we have $h(s) \equiv_L h(t)$. Since $h(s) = [\![s]\!]_k$ as well as $h(t) = [\![t]\!]_k$ and by Corollary 4.5, we obtain $h(s) \approx_{\mathcal{F}_n} [\![s]\!]_\varrho$ and $h(t) \approx_{\mathcal{F}_n} [\![t]\!]_\varrho$. Altogether, we conclude that $[\![s]\!]_\varrho \models \varphi$ if and only if $h(s) \models \varphi$ if and only if $h(t) \models \varphi$ if and only if $[\![t]\!]_\varrho \models \varphi$.

"2. $\Rightarrow$ 1.". Let $B \subseteq \Lambda$ be a finite alphabet and $L \subseteq B^*$ a language defined by a sentence $\varphi \in \mathcal{F}$. Let $n = \mathrm{qd}(\varphi)$ and $k \geq 2^{n+1} - 1$ be an idempotency exponent of M_L. We have to show that every π-algebra morphism g from T_A into the k-power algebra on B^* satisfies $g(s) \equiv_L g(t)$. Consider such a morphism g and let h be the unique monoid morphism from A^* into B^* defined by $h(a) = g(a)$ for each $a \in A$. Then $g(s) = h([\![s]\!]_k)$ and $g(t) = h([\![t]\!]_k)$. Corollary 4.5 and the assumption $[\![s]\!]_\varrho \approx_{\mathcal{F}} [\![t]\!]_\varrho$ yield $[\![s]\!]_k \approx_{\mathcal{F}_n} [\![s]\!]_\varrho \approx_{\mathcal{F}_n} [\![t]\!]_\varrho \approx_{\mathcal{F}_n} [\![t]\!]_k$. We conclude $g(s) \approx_{\mathcal{F}_n} g(t)$ by Corollary 4.4. By Proposition 3.4, we obtain $ug(s)v \approx_{\mathcal{F}_n} ug(t)v$ for all $u, v \in B^*$. Since $\varphi \in \mathcal{F}_n$, this finally implies $g(s) \equiv_L g(t)$. ◀

In the remainder of this section, we demonstrate two applications of Theorem 4.1 by providing quite short proofs of two well-known results. The following corollary can be obtained by combining a result of McNaughton and Papert [12] with Schützenberger's characterization of star-free languages [16]. A more direct proof can, for instance, be found in [17]. A finite monoid M is called *aperiodic* if the identity $a^\pi a = a^\pi$ holds in M.

▶ **Corollary 4.6.** *The syntactic monoid of every first-order definable language is aperiodic.*

Proof. The predicates suc, min, max and empty can be expressed in $\mathsf{FO}[<]$. By Theorem 4.1, it suffices to show $[\![a^\pi a]\!]_\varrho \approx_{\mathsf{FO}[<]} [\![a^\pi]\!]_\varrho$. The property $\varrho + 1 = \varrho$ of the order type ϱ implies $[\![a^\pi a]\!]_\varrho = [\![a^\pi]\!]_\varrho$ and the claim follows. ◀

The second application relates definability in $\mathsf{FO}^2[<]$ to the class $\mathbf{DA}$. The fragment $\mathsf{FO}^2[<]$ consists of all formulae not containing the predicates suc, min, max and empty which quantify over two fixed variables $x_1, x_2 \in \mathbb{V}$ only. The class $\mathbf{DA}$ consists of all finite monoids in which the identity $(abc)^\pi b (abc)^\pi = (abc)^\pi$ holds. A significant amount of book-keeping is involved when showing that the syntactic monoid of every $\mathsf{FO}^2[<]$-definable language is in $\mathbf{DA}$ by applying the classical Ehrenfeucht-Fraïssé game approach, see e.g. [5][1]. On the other hand, the abstract idea of this proof is very simple: Duplicator copies every move near the left and near the right border, and he does not need to care in the center. We now show that this idea can easily be formalized when using Theorem 4.1.

▶ **Corollary 4.7.** *The syntactic monoid of any language definable in* $\mathsf{FO}^2[<]$ *is in* $\mathbf{DA}$.

Proof. Let $s = (abc)^\pi b (abc)^\pi$ and $t = (abc)^\pi$. Again by Theorem 4.1, it suffices to show $[\![s]\!]_\varrho \approx_{\mathsf{FO}^2[<]} [\![t]\!]_\varrho$. With $u = (abc)^\omega (abc)^{\zeta \cdot \eta}$ and $v = (abc)^{\zeta \cdot \eta} (abc)^{\omega^*}$ we obtain

$$[\![s]\!]_\varrho = u\,(abc)^{\omega^*} b (abc)^\omega\, v \qquad \text{and} \qquad [\![t]\!]_\varrho = u\,(abc)^{\omega^*} (abc)^\omega\, v\,.$$

Since $\mathsf{FO}^2[<]$ is closed under negation and due to Proposition 3.4, it further suffices to show that Duplicator has a winning strategy in the $\mathsf{FO}^2[<]$-game on

$$\left((abc)^{\omega^*} b (abc)^\omega,\ (abc)^{\omega^*} (abc)^\omega\right).$$

The strategy is to choose a reply that is labeled by the same letter as the request and such that the positions corresponding to x_1 and x_2 are in the same order in both words. This is always possible, since in both words there are always infinitely many positions to the left (respectively to the right) of any position which are labeled by a given letter from a, b, c. ◀

[1] Actually, the proof given in [5] does not use the language of Ehrenfeucht-Fraïssé games, but it can easily be restated this way.

5 The Word Problem for π-Terms over Aperiodic Monoids

The word problem for π-terms over aperiodic monoids was solved by McCammond [11] by computing normal forms. In the process of computing these normal forms the intermediate terms can grow and, to the best of our knowledge, neither the worst-case running time nor the maximal size of the intermediate terms has been estimated (and it seems to be difficult to obtain such results). In this section we give an exponential algorithm for solving the word problem for π-terms over aperiodic monoids. Our algorithm does not compute normal forms as π-terms; instead we show that the evaluation under $[\![\cdot]\!]_\varrho$ can be used as a normal form for π-terms.

▶ **Theorem 5.1.** *Given two π-terms $s, t \in T_\Lambda$, one can decide whether the identity $s = t$ holds in every aperiodic monoid in time exponential in the size of s and t.*

The proof is a reduction to the isomorphism problem for regular words, cf. [3]. These generalized words particularly include all ϱ-rational words and can be described by expressions similar to π-terms but using ω-power, ω^*-power and dense shuffle instead of the π-power. Due to [3, Theorem 79], one can decide in polynomial time whether two such expressions describe isomorphic words. The characterization underlying the reduction is as follows:

▶ **Proposition 5.2.** *For all π-terms $s, t \in T_\Lambda$ the following conditions are equivalent:*
1. *The identity $s = t$ holds in every aperiodic finite monoid.*
2. $[\![s]\!]_\varrho = [\![t]\!]_\varrho$.

Proof. "1. $\Rightarrow$ 2.". The results in [11] imply that the identity $s = t$ can be deduced from the following list of axioms, where $n \geq 1$:

$$(uv)w = u(vw) \qquad (u^\pi)^\pi = u^\pi \qquad (u^n)^\pi = u^\pi$$
$$u^\pi u^\pi = u^\pi \qquad u^\pi u = uu^\pi = u^\pi \qquad (uv)^\pi u = u(vu)^\pi .$$

As a matter of fact, the ϱ-power algebra on Λ^{LO} satisfies these axioms as well. Consequently, $[\![s]\!]_\varrho = [\![t]\!]_\varrho$ can be proved along a deduction of the identity $s = t$ from the axioms.

"2. $\Rightarrow$ 1.". Due to Eilenberg's Variety Theorem [6], the pseudovariety of aperiodic monoids is generated by the class of syntactic aperiodic monoids. The latter are precisely the syntactic monoids of first-order definable languages [12, 16]. By Theorem 4.1 the identity $s = t$ holds in the syntactic monoid of every such language. ◀

Proof of Theorem 5.1. In order to apply the decision procedure from [3, Theorem 79], we have to translate s and t into expressions generating the same words and which do not use ϱ-power but ω-power, ω^*-power and dense shuffle instead. Such a translation can be based on the identity $u^\varrho = u^\omega \left(u^{\omega^*} u^\omega\right)^\eta u^{\omega^*}$ which holds for all words $u \in \Lambda^{\mathsf{LO}}$. Therein, the η-power is a special case of the dense shuffle. Since this translation leads to a blow-up which is exponential in the number of nested applications of π-powers within s and t, we can decide $[\![s]\!]_\varrho = [\![t]\!]_\varrho$ in time at most exponential in the size of s and t. ◀

6 Summary

For every π-term t we define a labeled linear order $[\![t]\!]_\varrho$, and every first-order fragment $\mathcal{F}$ over finite words naturally yields a (possibly infinite) Ehrenfeucht-Fraïssé game on labeled linear orders. The important property of these constructions is that $\mathcal{F}$ satisfies an identity

$s = t$ of π-terms s and t if and only if Duplicator has a winning strategy in the $\mathcal{F}$-game on $[\![s]\!]_\varrho$ and $[\![t]\!]_\varrho$. We note that $[\![t]\!]_\varrho$ does not depend on $\mathcal{F}$. Usually showing that a fragment $\mathcal{F}$ satisfies an identity $s = t$ requires a significant amount of book-keeping which in most cases is not part of the actual proof idea. Our main results Theorem 4.1 and Theorem 4.2 allow to formalize such proof ideas without further book-keeping, see e.g. Corollary 4.7. A probably less obvious application of our main result are word problems for π-terms over varieties of finite monoids. We show that the word problem for π-terms over aperiodic finite monoids is solvable in exponential time (Theorem 5.1), thereby improving a result of McCammond [11].

Several possible extensions of our result come to mind: Other implicit operations (see [1] for further details on implicit operations), logical fragments beyond classical first-order logic, and other structures such as infinite words, trees or Mazurkiewicz traces.

References

1 J. Almeida. *Finite Semigroups and Universal Algebra*. World Scientific, 1994.
2 J. Almeida and M. Zeitoun. An automata-theoretic approach to the word problem for ω-terms over $\mathbf{R}$. *Theoret. Comput. Sci.*, 370(1–3):131–169, 2007.
3 S. L. Bloom and Z. Ésik. The equational theory of regular words. *Information and Computation*, 197(1–2):55–89, 2005.
4 J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.
5 V. Diekert, P. Gastin, and M. Kufleitner. A survey on small fragments of first-order logic over finite words. *Int. J. Found. Comput. Sci.*, 19(3):513–548, 2008.
6 S. Eilenberg. *Automata, Languages, and Machines*, vol. B. Academic Press, 1976.
7 C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.
8 N. Immerman and D. Kozen. Definability with bounded number of bound variables. *Information and Computation*, 83(2):121–139, Nov. 1989.
9 M. Kufleitner and A. Lauser. Lattices of logical fragments over words. In *ICALP 2012, Proceedings Part II*, volume 7392 of *LNCS*, pp. 275–286. Springer, 2012.
10 M. Kufleitner and A. Lauser. Quantifier alternation in two-variable first-order logic with successor is decidable. In *STACS 2013, Proceedings*, vol. 20 of *LIPIcs*, pages 305–316. Dagstuhl Publishing, 2013.
11 J. P. McCammond. Normal forms for free aperiodic semigroups. *Int. J. Algebra Comput.*, 11(5):581–625, 2001.
12 R. McNaughton and S. Papert. *Counter-Free Automata*. The MIT Press, 1971.
13 A. Moura. The word problem for ω-terms over $\mathbf{DA}$. *Theoret. Comput. Sci.*, 412(46):6556–6569, 2011.
14 D. Perrin and J.-É. Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, 2004.
15 J.-É. Pin. *Varieties of Formal Languages*. North Oxford Academic, 1986.
16 M. P. Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control*, 8:190–194, 1965.
17 H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.
18 H. Straubing. Algebraic characterization of the alternation hierarchy in $\mathrm{FO}^2[<]$ on finite words. In *CSL 2011, Proc.*, vol. 12 of *LIPIcs*, pp. 525–537. Dagstuhl Publishing, 2011.
19 D. Thérien and Th. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *STOC 1998, Proceedings*, pp. 234–240. ACM Press, 1998.
20 B. A. Trakhtenbrot. Finite automata and logic of monadic predicates (in Russian). *Dokl. Akad. Nauk SSSR*, 140:326–329, 1961.

Faster Sparse Suffix Sorting

Tomohiro I[1], Juha Kärkkäinen[2], and Dominik Kempa[3]

1 **Kyushu University, Japan**
`tomohiro.i@inf.kyushu-u.ac.jp`
2 **University of Helsinki, Finland**
`juha.karkkainen@cs.helsinki.fi`
3 **University of Helsinki, Finland**
`dominik.kempa@cs.helsinki.fi`

Abstract

The sparse suffix sorting problem is to sort $b = o(n)$ arbitrary suffixes of a string of length n using $o(n)$ words of space in addition to the string. We present an $\mathcal{O}(n)$ time Monte Carlo algorithm using $\mathcal{O}(b \log b)$ space and an $\mathcal{O}(n \log b)$ time Las Vegas algorithm using $\mathcal{O}(b)$ space. This is a significant improvement over the best prior solutions by Bille et al. (ICALP 2013): a Monte Carlo algorithm running in $\mathcal{O}(n \log b)$ time and $\mathcal{O}(b^{1+\epsilon})$ space or $\mathcal{O}(n \log^2 b)$ time and $\mathcal{O}(b)$ space, and a Las Vegas algorithm running in $\mathcal{O}(n \log^2 b + b^2 \log b)$ time and $\mathcal{O}(b)$ space. All the above results are obtained with high probability not just in expectation.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases string algorithms, sparse suffix sorting, sparse suffix trees, Karp–Rabin fingerprints, space–time tradeoffs

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.386

1 Introduction

Given a string T of length n and a set of b positions in T, the *sparse suffix sorting problem* is to sort the set of suffixes of T starting at those positions. The problem can be easily solved in $\mathcal{O}(n)$ time and space by constructing the suffix array, i.e., by sorting the set of all suffixes of T [8]. However, if the space is restricted to $\mathcal{O}(b)$ words in addition to the string the problem becomes more difficult. A straightforward solution is to use plain string sorting without taking advantage of the overlaps between the suffixes, but this requires at least $\Omega(nb)$ time in the worst case since the total length of the suffixes as separate strings is $\Omega(nb)$. More generally, we are interested in space–time tradeoffs, i.e., solutions using $\mathcal{O}(s)$ words of extra space for $s \in [b..n]$.

An efficient sparse suffix sorting algorithm has many potential applications. In the space-efficient Burrows–Wheeler transform algorithm in [9], sparse suffix sorting is used for two purposes: to sort a random sample of suffixes in order to partition the suffix array into approximately equal parts, and then to sort each of those parts separately. A similar approach might be useful for external memory and distributed suffix array construction. A sorted random sample of suffixes might be useful for estimating various measures and statistics on the string, but this direction has not been studied much because, until recently, there was no efficient algorithm for sorting a random sample. The full suffix array has numerous applications in text indexing and mining [1]. If application specific information allows us to focus on a smaller set of text positions, a full text index can be replaced with a much smaller sparse text index [2], the construction of which requires sparse suffix sorting.


© Tomohiro I, Juha Kärkkäinen, and Dominik Kempa;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 386–396

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


There are many results for efficiently sorting certain special types of sets of suffixes (see [2]), but the general case of sorting an arbitrary set of suffixes has proved to be much harder. We are aware of two previous results improving on the naive bounds mentioned above. One uses the technique of *difference cover sampling* [4, 8] for sorting the suffixes in $\mathcal{O}(n^2/s)$ time and $\mathcal{O}(s)$ extra space for any $s \in [b..n]$ [8, Sect. 8]. The other result, by Bille et al. [2], is a Monte Carlo algorithm running in $\mathcal{O}(n \log^2 b / \log(1 + s/b))$ time and $\mathcal{O}(s)$ extra space. In particular, it runs in $\mathcal{O}(n \log^2 b)$ time when using $\mathcal{O}(b)$ space and in $\mathcal{O}(n \log b)$ time when using $\mathcal{O}(b^{1+\epsilon})$ space for any constant $\epsilon > 0$. Bille et al. also describe a deterministic verification algorithm to obtain a Las Vegas algorithm running in $\mathcal{O}(n \log^2 n + b^2 \log b)$ time using $\mathcal{O}(b)$ words of extra space.

In this paper, we improve on the results of Bille et al. by presenting a randomized Monte Carlo algorithm that runs in $\mathcal{O}(n + (nb/s) \log s)$ time using $\mathcal{O}(s)$ words of extra space for any $s \in [b..n]$. In particular, the time is $\mathcal{O}(n \log b)$ when using $\mathcal{O}(b)$ space and $\mathcal{O}(n)$ when using $\mathcal{O}(b \log b)$ space. As with the algorithm of Bille et al., our algorithm may produce an incorrect answer but the probability of this happening can be made smaller than n^{-c} for any constant c. In addition, we obtain a faster Las Vegas algorithm by describing a deterministic verification algorithm running in $\mathcal{O}(n \log b)$ time using $\mathcal{O}(b)$ words of extra space.

Our algorithms use some of the same basic tools and techniques as those of Bille et al., but in a quite different way. Karp–Rabin fingerprints have a key role in both Monte Carlo algorithms, but our underlying sorting algorithm is the optimal radix sorting of Paige and Tarjan [11] rather than the quicksort of Bille et al. Similarly to Bille et al., our verification algorithm utilizes the transitivity of equality by finding cycles in graphs connecting substrings that are supposed be equal, but our algorithms for processing the graphs are entirely different.

2 Problem and Model of Computation

Let $T = T[1..n]$ be a string of length n. To simplify algorithms, we assume that $T[n]$ is a unique character, ensuring that the set of suffixes is prefix-free. For $i \in [1..n]$, let $T_i = T[i..n]$ denote the suffix of length $n - i + 1$ and let $lcp(i, j)$ denote the length of the *longest common prefix* between the suffixes T_i and T_j. Let $\mathcal{S} \subseteq [1..n]$ be a set of b positions in T. The problem we want to solve is to sort the set of suffixes $T_{\mathcal{S}} = \{T_i : i \in \mathcal{S}\}$ lexicographically.

The *sparse suffix array* $SSA[1..b]$ is the array containing the positions in $\mathcal{S}$ in the lexicographical order of the suffixes. The associated *LCP array* $LCP[2..b]$ contains the longest common prefix lengths for adjacent suffixes in the sorted set, that is, $LCP[i] = lcp(SSA[i], SSA[i-1])$ for all $i \in [2..b]$. Given the arrays SSA and LCP, we can easily compute the sparse suffix tree SST for $T_{\mathcal{S}}$ in $\mathcal{O}(b)$ time [2] (see Section 4 for a definition of SST). Conversely, SSA and LCP can be constructed in $\mathcal{O}(b)$ time by a depth first traversal of SST. If we are given SSA but not LCP, we can easily compute the LCP in $\mathcal{O}(n \log b)$ time and $\mathcal{O}(b)$ extra space using the techniques of Bille et al. [2] or the ones in this paper. Thus sorting the suffixes is the key problem. The Monte Carlo sorting algorithm of Section 4 outputs SST and the deterministic verification algorithm of Section 5 takes SSA and LCP as input.

The model of computation is the word RAM with word size $\Omega(\log n)$. For the most part, our algorithms operate in the comparison model, i.e., we only assume that characters can be compared in constant time. However, for fingerprint computation in the Monte Carlo algorithm, we need the stronger assumption that, for any character a, we can in constant time compute an integer representation $\rho(a)$ such that $\rho(a) = \rho(b)$ if and only if $a = b$. Note that we do not assume that the integer representation can be used for order comparison.

3 Basic Techniques

3.1 Karp–Rabin fingerprints

Let q be a prime and choose $r \in [0..q-1]$ uniformly at random.[1] The fingerprint for a substring of $T[i..j]$ is defined as

$$FP[i..j] = \sum_{k=i}^{j} \rho(T[k]) \cdot r^{j-k} \bmod q \ .$$

Clearly, if $T[i..i+\ell] = T[j..j+\ell]$ then $FP[i..i+\ell] = FP[j..j+\ell]$. On the other hand, if $T[i..i+\ell] \neq T[j..j+\ell]$ then $FP[i..i+\ell] \neq FP[j..j+\ell]$ with a probability at least $1-\ell/q$ [6]. Since we are comparing only substrings of equal length, the number of different possible substring comparisons is less than n^3. Thus, for any positive constant c, we can set q to be a prime larger than n^{c+4} (but still small enough to fit in $\mathcal{O}(1)$ words) to make the fingerprint function perfect with probability at least $1-n^{-c}$.

The fingerprint of a string of length ℓ can obviously be computed in $\mathcal{O}(\ell)$ time, but for Karp–Rabin fingerprints [10], we can additionally use the following equations to compute fingerprints more efficiently from existing fingerprints. For all $i \leq j \leq k$,

$$\begin{aligned} FP[i..j+1] &= FP[i..j] \cdot r + \rho(T[j+1]) \bmod q \\ FP[i..k] &= FP[i..j] \cdot r^{k-j} + FP[j+1..k] \bmod q \\ FP[j+1..k] &= FP[i..k] - FP[i..j] \cdot r^{k-j} \bmod q \end{aligned} \tag{1}$$

3.2 Grouping

By grouping, we mean the task of ordering a multiset so that equal elements are grouped together but the groups can be in arbitrary order. Suppose we have a zero-initialized array A of s words. Given $k = o(s)$ integers from a universe of size s, we can group but not sort the integers in $\mathcal{O}(k)$ time by a simple distribute-and-collect procedure while leaving A in the zero-initialized state ready to be reused. Thus d such grouping tasks can be executed in $\mathcal{O}(s+kd)$ time. Sorting could be done in $\mathcal{O}(s+kd)$ time offline (as a single batch) but online sorting would require $\mathcal{O}(sd)$ time. The online setting occurs in radix sorting. Thus, grouping k integers from a universe of size $u > s$ by a modified LSD radix sort with radix s can be done in $\mathcal{O}(s + k\log_s u)$ time while sorting would need $\mathcal{O}(s\log_s u)$ time. A prominent example of this technique is the string sorting algorithm of Paige and Tarjan [11]. It uses a similar modification of MSD radix sorting into a grouping algorithm to construct an *unsorted* compact trie of the strings, and then sorts the child edges of all nodes in one batch. Our suffix sorting algorithm has the same structure, though the unsorted trie construction is different and not based on MSD radix grouping (but does involve LSD radix grouping). Grouping and unsorted compact tries have a key role in our algorithm because fingerprints cannot be used for order comparisons.

3.3 String periodicity

An integer $p \in [0..m-1]$ is a *period* of a string $X[1..m]$ if $X[i] = X[i+p]$ for all $i \in [1..m-p]$. A basic result on periods is the following:

[1] The choice of r is the only random operation in the algorithm.

▶ **Lemma 1** (Weak Periodicity Lemma [7]). *If p_1 and p_2 are periods of a string $X[1..m]$ and $p_1 + p_2 \leq m$, then $\gcd(p_1, p_2)$ is a period of X too.*

Our verification algorithm uses this in the form of the following generalization to multiple periods:

▶ **Corollary 2.** *Integers $p_1, p_2, \ldots, p_k \in [0..\lceil m/2 \rceil]$ are periods of a string $X[1..m]$ if and only if $\gcd(p_1, p_2, \ldots, p_k)$ is a period of X.*

The value $\gcd(p_1, p_2, \ldots, p_k)$ can be computed in $\mathcal{O}(k + \log m)$ time using the Euclidean algorithm [3]. Note that zero is a period of all nonempty strings, and all of the above holds for 0-periods too if we define $\gcd(p, 0) = p$ for all p.

4 Monte Carlo Algorithm

In this section, we describe a Monte Carlo algorithm for sparse suffix sorting. The output is actually the sparse suffix tree SST of the b suffixes in $T_{\mathcal{S}}$, but this is equivalent to suffix sorting as explained above. The construction of SST uses a novel technique of gradual refinement.

We start by defining a relaxed form of compact tries. An *ℓ-strict compact trie* is a rooted tree, where each internal node has at least two children. The edges are labelled by strings, and induced by the edge labelling, each node v is labelled by the concatenation of the edge labels from the root to v. For any two edges with the same parent node, their labels must be different, one label cannot be a prefix of the other, and the labels cannot share a common prefix of length ℓ (ℓ-strictness). The trie is ordered if the child edges of each node are lexicographically ordered by their labels. The standard compact trie is the (unique) ordered 1-strict compact trie.

An *ℓ-strict sparse suffix tree* ℓ-SST for the set $T_{\mathcal{S}}$ is an ℓ-strict compact trie with b leaves representing exactly the suffixes in $T_{\mathcal{S}}$. The edge labels are substrings of T and can be represented in constant space by pointers to T. Thus the size of an ℓ-SST is $\mathcal{O}(b)$ words. The standard sparse suffix tree SST (the desired output of the algorithm) is the ordered 1-SST.

The algorithm begins by constructing ℓ-SST for $\ell = 2^{\lceil \log n \rceil}$, which is just the root and b leaves with the edges labelled by the suffixes. Then, in each round of the computation, the algorithm halves the value of ℓ and computes an ℓ-SST from the 2ℓ-SST until $\ell = 1$. Given the 1-SST, we can sort the child edges of all nodes using $\mathcal{O}(b \log b)$ character comparisons to obtain SST.

To compute ℓ-SST from 2ℓ-SST, we perform the following steps:

1. For each edge with a label of length at least ℓ, compute the fingerprint $FP[i..i + \ell - 1]$, where i is the starting position of the edge label in T.
2. Use LSD radix grouping to group all edges from Step 1 using the key (u, f), where u is the parent node of the edge and f is the fingerprint from Step 1.
3. For each group of at least two edges with the same key (u, f), add a new node v and an edge (u, v) to the tree. The new node v becomes the parent node of the edges in the group. The label of (u, v) has length ℓ and the starting position is taken from one of the old edges. The labels of the old edges are shortened by ℓ by moving the starting positions forward by ℓ positions. If v is the only child of u after the stage, we remove u by concatenating its parent and child edges.

It is easy to check that the tree resulting from the above steps (i) is still a valid compact trie, (ii) retains the labels of all nodes inherited from 2ℓ-SST, and (iii) satisfies the ℓ-strictness condition. It is thus a valid ℓ-SST.

The remaining detail is how to compute the fingerprints. We assume that we can use $\mathcal{O}(s)$ words of extra space for some $s \in [b..n]$. We divide the string T into blocks of size $h = n/s$. For all $i \in [1..n/h]$, we compute and store the values $FP[1..ih]$ and $r^{ih} \bmod q$. We can then compute the fingerprint of any substring of T of length ℓ in $\mathcal{O}(\min(\ell, n/s))$ time using Equations (1).

▶ **Theorem 3.** *Any set of b suffixes of a string of length n can be sorted correctly with high probability in $\mathcal{O}(n + (bn/s)\log s)$ time using $\mathcal{O}(s)$ words of space in addition to the string for any $s \in [b..n]$.*

Proof. With high probability, there are no fingerprint collisions, which is enough to ensure the correctness of the algorithm by the above discussion.

Most of the data structures including the tree fit in $\mathcal{O}(b)$ space during the whole computation. We need $\mathcal{O}(s)$ space for the precomputed fingerprints and powers of r, as well as for the bucket array in LSD radix grouping. Thus the total space requirement is $\mathcal{O}(s)$ words in addition to the string T.

The time complexity is dominated by the fingerprint computation. The initialization of the precomputed fingerprints needs $\mathcal{O}(n)$ time. In the first $\log(s)$ rounds (i.e., while $\ell \geq n/s$), each fingerprint computation in Step 1 takes $\mathcal{O}(n/s)$ time, and in the later rounds $\mathcal{O}(\ell)$ time. Thus the total time for fingerprint computation is $\mathcal{O}(n + (bn/s)\log s)$. In Step 2 of each round, we are grouping $\mathcal{O}(b)$ integers of $\mathcal{O}(\log n)$-bits each using a bucket array of size s, which takes $\mathcal{O}(b \log_s n)$ time. The total grouping time is $\mathcal{O}(b \log^2 n / \log s) = \mathcal{O}((bn/s)\log s)$. Everything else can be done in $\mathcal{O}(b \log n) = \mathcal{O}((bn/s)\log s)$ time. ◀

By choosing $s = b$ and $s = b \log b$, we obtain the following results.

▶ **Corollary 4.** *Any set of b suffixes of a string of length n can be sorted correctly with high probability in $\mathcal{O}(n \log b)$ time using $\mathcal{O}(b)$ words of space in addition to the string.*

▶ **Corollary 5.** *Any set of $b = \mathcal{O}(n/\log n)$ suffixes of a string of length n can be sorted correctly with high probability in $\mathcal{O}(n)$ time using $\mathcal{O}(b \log b)$ words of space in addition to the string.*

Note that if $b \log b = \omega(n)$, we can use a full suffix array construction algorithm to sort $T_{\mathcal{S}}$ in $\mathcal{O}(n)$ time and space (assuming a proper integer alphabet where strings can be radix sorted).

5 Las Vegas Algorithm

In this section we describe a deterministic algorithm to check if the arrays SSA and LCP are correct. Assuming SSA contains a permutation of set $\mathcal{S}$ and besides the trivial sanity checks of LCP, we need to verify if the conditions

$$\begin{aligned} T[a_i..a_i + \ell_i - 1] &= T[b_i..b_i + \ell_i - 1] \\ T[a_i + \ell_i] &< T[b_i + \ell_i] \end{aligned} \tag{2}$$

are satisfied for $i \in [2..b]$, where $a_i = SSA[i-1]$, $b_i = SSA[i]$ and $\ell_i = LCP[i]$.

Checking the second condition takes only $\mathcal{O}(b)$ time. A naive verification of the first one, however, requires $\mathcal{O}(bn)$ operations in the worst-case. We now describe a faster algorithm.

The first version runs in $\mathcal{O}(n \log^2 b)$ time. A refinement yielding $\mathcal{O}(n \log b)$ time is presented next.

5.1 $\mathcal{O}(n \log^2 b)$ time algorithm

Let $m_0 = 3 \cdot 2^{\lfloor \log(n/3) \rfloor}$ and $m_h = m_0/2^h$ for $h > 0$. A substring of T of length m_h for some h is called a *segment*. Any substring can be covered by at most two (possibly overlapping) segments. Thus we can replace the b substring equalities with at most $2b$ segment equalities. A pair of segments whose equality we need to verify is called a *twin pair*. A twin pair of segments of length m_h is called an *h-pair* and its two segments are called *h-segments*.

The algorithm maintains two lists of segments, L and R. Initially, both lists contain all twin pair segments, with L sorted by the left endpoint (i.e., the starting position of the substring in T) and R sorted by right endpoint. Each segment is linked to its twin in the same list. We sort the lists once in $\mathcal{O}(b \log b)$ time in the beginning and maintain the sorted order afterwards. During the algorithm the size of each list never grows beyond the initial size of at most $4b$. The lists L and R are processed symmetrically (left–right symmetry) and completely independently of each other. We focus on describing the algorithm for L.

5.1.1 Overview

The algorithm operates in $\log b$ rounds. In the h-th round, $h \in [0..\lfloor \log b \rfloor]$, we process all h-pairs in L. The set of all h-pairs is partitioned into two subsets, *fully checked* pairs and *partially checked* pairs. The former, relatively small subset is verified by a direct brute-force comparison of the substrings. If a mismatch is detected, the verifier exits with a negative answer. Otherwise, we use the knowledge gained during these naive checks to indirectly verify the equality of the *middle third* of every partially checked pair. All fully checked pairs are removed from L and each partially checked h-pair is replaced by its left half (i.e., the $(h+1)$-pair with the same left endpoints), to obtain the list for the next round. After $\log b$ rounds, all segments on the list have length $\mathcal{O}(n/b)$ and can be checked by brute-force in $\mathcal{O}(n)$ total time.

Consider an initial twin pair that is partially checked and replaced by its first half, which in turn is partially checked and replaced etc., until eventually a pair is fully checked. The verified middle thirds of these pairs together with the final, fully checked pair completely cover the leftmost two thirds of the initial pair. Therefore, if the verifier did not exit prematurely due to a mismatch, the leftmost two thirds of each segment is confirmed to be equal to the leftmost two thirds of its twin. The symmetric processing of the list R will similarly verify all rightmost two thirds resulting in a complete verification.

5.1.2 A single round in detail

Suppose we are at round h and denote $m = m_h$. We start by constructing an undirected multigraph $G = G_h$ from the list L. Consider a partition of T into blocks of size $B = B_h = m/\lceil 6 \log b + 18 \rceil$. Each block corresponds to one vertex in G. We associate every h-segment in L with the block containing the left endpoint. For any h-pair in L we create an edge between the vertices associated with the two segments of the pair. Isolated vertices are omitted during the construction, thus the resulting graph has $|V(G)| = \mathcal{O}(\min(b, (n/m) \log b))$ vertices and $|E(G)| \leq 2b$ edges. This graph is essentially the same as the one in [2].

We start a breadth-first search from an arbitrary vertex v of G. The search continues as long as each new BFS layer (the subset of vertices with the same shortest distance to v)

at least doubles the total size of the BFS tree. Denote the constructed tree as F and the subgraph of G containing all edges inspected during the BFS as G'. Note that F and G' contain the vertices of the last layer but no edges with both ends in the last layer. The h-pairs associated with the tree edges $E(F)$ are chosen as fully checked pairs and the brute force comparisons are performed. The h-pairs associated with non-tree edges $E(G') \setminus E(F)$ become partially checked pairs. We will next describe how to indirectly verify the equality for the middle $m/3$ positions of those pairs.

Let A be an arbitrary h-segment associated with the root v of F. By M we denote the substring of A of length $2m/3$ centered exactly in the middle of A.

▶ **Lemma 6.** *Suppose all h-pairs corresponding to edges of F were successfully verified. Then every h-segment associated with some vertex of G' contains M as a substring.*

Proof. First note that the height of F is at most $\log b + 2$. Let U be an arbitrary segment associated with a vertex $u \in V(G')$. Consider the sequence of segments starting with A, containing all twin pairs associated with the edges on the path from v to u in F, and ending with U. Each adjacent pair of segments in this sequence is either a fully checked pair or is associated with the same vertex. In the former case, the segments are verified to be identical. In the latter case, the segments overlap by more than $m - B$ and the relative position of M inside the segments differ by less than B. Over the whole sequence, the relative position of M changes by less than $B(\log b + 3) \leq m/6$. Since M starts at a distance of $m/6$ from both ends of A, all segments in the sequence including U must contain M. ◀

We exploit this fact as follows. Let $\{u_i, u_j\} \in E(G') \setminus E(F)$ be an arbitrary edge associated with a twin pair $\{U_i, U_j\}$. Let d_i (d_j) be the relative position of M inside U_i (U_j).

▶ **Lemma 7.** *If M has period $p = |d_i - d_j|$, then a substring of length $m/3$ centered in the middle of U_i matches the corresponding substring in U_j. Otherwise $U_i \neq U_j$.*

Proof. If we align U_i and U_j, the occurrences of M inside the segments overlap by $k = |M| - p \geq m/3$. M has period p if and only if that overlapping part is equal in both segments. The lemma follows from the facts that the overlapping parts contain the middle thirds and are contained in the whole segments. ◀

Each non-tree edge of G' generates one periodicity query for M. In order to efficiently answer all queries we observe that p cannot exceed $m/3 = |M|/2$, thus all queries can be reduced to a single one using Corollary 2 (in Section 3). If the reduced periodicity query returns true, we obtain the equality of middle third for all h-pairs associated with $E(G') \setminus E(F)$. Otherwise, Lemma 7 implies that there must be a mismatch in some pair, and hence the verifier returns a negative answer and exits.

After processing G', the edges $E(G')$ are deleted from G along with the vertices that become isolated. The procedure is then repeated for the remaining part of the graph and this continues until the graph is empty.

▶ **Lemma 8.** *Round h of verification takes $\mathcal{O}(b + |V(G_h)|m_h)$ time and $\mathcal{O}(b)$ extra space.*

Proof. The list L is sorted, thus building G takes $\mathcal{O}(b)$ time and space. All BFS searches require $\mathcal{O}(|V(G)| + |E(G)|) = \mathcal{O}(b)$ time in total. The stopping criterion for the BFS implies that deleting the set $E(G')$ along with the introduced isolated vertices removes at least $|V(G')|/2$ vertices from G, thus the BFS trees have altogether $\mathcal{O}(|V(G)|)$ edges. The brute-force checking of all twin pairs associated with such edges therefore takes $\mathcal{O}(|V(G)|m)$ time.

The relative position of M can be easily tracked during the BFS, hence the generation of the periodicity queries of Lemma 7 for all non-tree edges can be done in $\mathcal{O}(b)$ time. Reducing the number of periodicity queries using Corollary 2 consumes $\mathcal{O}(b + g \log m)$ time in total, where $g \leq |V(G)|$ is the number of subgraphs G' processed during the round, and the reduced periodicity queries can be executed in $\mathcal{O}(gm)$ time. Thus total time for all partial checks is $\mathcal{O}(b + gm) = \mathcal{O}(b + |V(G)|m)$. ◀

Note that $\mathcal{O}(b + |V(G_h)|m_h) = \mathcal{O}(n \log b)$ for all h. We perform $\mathcal{O}(\log b)$ rounds, after which we verify the remaining segments in $\mathcal{O}(n)$ time, hence we obtain the following result.

▶ **Theorem 9.** *The correctness of a sparse suffix tree constructed for a set of b suffixes of a string of length n can be deterministically verified in $\mathcal{O}(n \log^2 b)$ time and $\mathcal{O}(b)$ words of space in addition to the string.*

5.2 $\mathcal{O}(n \log b)$ time algorithm

The time complexity of the verification algorithm is dominated by the $\mathcal{O}(|V(G_h)|m_h)$ time spent in each round h for brute force comparisons and partial checks. The number of vertices is bounded by $\mathcal{O}(n \log b/m_h)$ but can be smaller. In this section we show how to modify the algorithm to reduce the number of vertices in the graphs, which decreases the overall verification time to $\mathcal{O}(n \log b)$. The reduction is accomplished by moving segments in a way that concentrates them on certain areas while other areas become empty. The empty areas contribute no vertices to the graphs.

The movement of segments happens at the end of each round. Consider the end of round h when all h-pairs on the list L have been removed or replaced. Let i and j be the left endpoints of the segments in some h-pair that was fully checked during the round, i.e., we know that $T[i..i + m_h - 1] = T[j..j + m_h - 1]$. If an h'-segment in L for some $h' > h$ is completely inside $T[i..i + m_h - 1]$ it can be moved to the corresponding position in $T[j..j + m_h - 1]$ without affecting the correctness of the verification. If we move all segments in L that are inside $T[i..i + m_h - 1]$, there are no left endpoints in the region $[i..i + m_h/2 - 1]$ anymore and we say that the region $[i..i + m_h/2 - 1]$ has been *cleared*.

However, a region that has been cleared may not stay cleared as other moves in the same round or later rounds may reintroduce left endpoints to the region. In the case of the moves in the same round, this is easy to avoid by doing all moves either leftwards or rightwards. As shown later, we cannot fix the direction in advance but have to choose the better direction separately for each round. However, in a single round, all moves are made in the same direction, and it is easy to see that by processing the fully checked pairs in the appropriate order, all cleared regions will stay cleared. Moves in later rounds are more problematic. The cleared region $[i..i + m_h/2 - 1]$ may be almost completely overlapped by a $(h + 1)$-segment with the left endpoint at $i - 1$ and moves to this $(h + 1)$-segment can undo the clearance. To avoid this, we will be more selective with the moves.

Let $h_{\text{last}} = \lfloor \log b \rfloor$ be the last round in the algorithm, and recall that $B_h = m_h/\lceil 6 \log b + 18 \rceil$. Define $d_h = (h_{\text{last}} - h + 2)B_h/2$ for all $h \in [0..h_{\text{last}}]$. If we are moving from $T[i..i + m_h - 1]$ to $T[j..j + m_h - 1]$, we will move segments if and only if their left endpoint is in the *source region* $[i..i + d_h]$, i.e., only the source region will be cleared. The left endpoints are moved to the *target region* $[j..j + d_h]$. Source and target regions are both called *move regions*. The next two lemmas establish the key properties of move regions. The first lemma shows that the source region is a subregion of $[i..i + m_h/2 - 1]$ and thus can indeed be cleared. The second one shows that a sufficiently large part of a cleared source region is *permanently cleared*.

▶ **Lemma 10.** *$d_h < m_h/12$ for all $h \in [0..h_{\text{last}}]$.*

Proof. We just need to note that $h_{\text{last}} - h + 2 < \log b + 3$ and $B_h \leq m_h/(6(\log b + 3))$. ◀

▶ **Lemma 11.** *If a source region $[i..i + d_h]$ is cleared in round h, then the region $[i + d_h - B_h..i + d_h]$ cannot be uncleared in later rounds.*

Proof. Consider the worst case scenario with regard to unclearing $[i..i + d_h]$. Assume there is an $(h+1)$-segment with left endpoint at $i-1$ and a target region extending to $i-1+d_{h+1}$. An $(h + 2)$-segment moved to the end of the target region has a target region extending to $i - 1 + d_{h+1} + d_{h+2}$, an $(h + 3)$-segment moved to the end of that target region has a target region extending to $i - 1 + d_{h+1} + d_{h+2} + d_{h+3}$ and so on. Thus no more than $D_h = \sum_{h'=h+1}^{h_{\text{last}}} d_{h'}$ leftmost positions of $[i..i + d_h]$ can be uncleared. We will show by induction that $D_h = d_h - B_h$, which proves the lemma.

Clearly, $D_{h_{\text{last}}} = 0 = d_{h_{\text{last}}} - B_{h_{\text{last}}}$. Assume then that $D_{h+1} = d_{h+1} - B_{h+1}$ and note that $B_h = 2B_{h+1}$. Thus, $D_h = D_{h+1} + d_{h+1} = 2d_{h+1} - B_{h+1} = (h_{\text{last}} - (h + 1) + 1)B_{h+1} = (h_{\text{last}} - h)B_h/2 = d_h - B_h$. ◀

In order to perform the segment moves efficiently, we will implement the list L as a list of sets, each of which contains segments with the same left endpoint and is implemented as a linked list. This representation supports all the operations we need in the basic algorithm, but additionally we can now remove a full set from the list, insert the set in a different place in the list or merge the set with a different set, all in constant time. When a segment belonging to a fully checked pair is removed from L, we remove it from the corresponding set but add a special node to L in front of the set and store a pointer to the special node in the corresponding move region. If a set contains multiple fully checked segments, they share a single special node. By moving the sets (but not the special nodes) we can execute all the moves from one source region in $\mathcal{O}(d_h) = \mathcal{O}(m_h)$ time. Thus the cost of the move operations is no higher than the cost of the brute force checks.

Due to possibly overlapping move regions, the order in which the moves are made has to be chosen with some care, but it is still possible to completely clear all the source regions in one round as long as all moves are made in the same direction. The direction, leftwards or rightwards, is chosen in each round so that the total area of the permanently cleared regions is maximized.

Finally, we need a minor modification to the way the algorithm deals with the special case, where the subgraph G' consists of a single vertex and one or more selfloop edges. In this case, one of the pairs associated with the edges is fully checked while others are partially checked. This does not affect the time complexity of the round, but ensures that every vertex is adjacent to at least one fully checked edge.

▶ **Theorem 12.** *The correctness of a sparse suffix tree constructed for a set of b suffixes of a string of length n can be deterministically verified in $\mathcal{O}(n \log b)$ time and $\mathcal{O}(b)$ words of space in addition to the string.*

Proof. Except for the segment moves, the algorithm operates as before and, as explained above, the segment moves do not compromise the correctness of the verification. Thus the algorithm is correct. The space complexity is clearly still $\mathcal{O}(b)$.

To prove the time complexity, we will show that the total area covered by the permanently cleared regions in round h is $\Theta(|V(G_h)|m_h/\log b)$. The costs of the form $\mathcal{O}(|V(G_h)|m_h)$ are distributed over the positions in the permanently cleared regions. Then every position gets charged for at most $\mathcal{O}(\log b)$ over the whole algorithm, and thus the total cost is $\mathcal{O}(n \log b)$.

Every vertex in the graph G_h is associated with a fully checked segment and each such segment contains a potential permanently cleared region (PPCR) of size B_h. The direction of moves decides whether a PPCR becomes a PCR. Two PPCRs can overlap only if they are associated with the same vertex or two vertices representing adjacent blocks. Thus there are at least $|V(G_h)|/2$ non-overlapping PPCRs and their total coverage is at least $B_h|V(G_h)|/2 = \Theta(|V(G_h)|m_h/\log b)$. The total coverage of the actual PCRs is at least half of that since we choose the direction to maximize the coverage. Note also that computing the coverage resulting from each direction can be done in $\mathcal{O}(b)$ time. ◀

▶ **Corollary 13.** *Any set of b suffixes of a string of length n can be sorted correctly using $\mathcal{O}(b)$ words of space in addition to the string in time that is $\mathcal{O}(n \log b)$ with high probability.*

6 Concluding Remarks

The time–space complexity of sparse suffix sorting has been a major open problem for a long time. Our new algorithms achieving the time–space product of $\mathcal{O}(nb \log b)$ are a major step towards a solution but open problems remain. Perhaps the main open problem is the deterministic time–space complexity: the best deterministic algorithms have a time–space product of $\mathcal{O}(n^2)$. Can our algorithms be made deterministic? Perhaps the use of fingerprints in the Monte Carlo algorithm can be replaced with a deterministic technique. Or perhaps the deterministic verification algorithm can be transformed into a sorting algorithm.

Some of the techniques developed in this paper, such as the ℓ-strict compact tries and their incremental construction, may have applications outside sparse suffix sorting. The concepts behind the verification algorithm could be useful not only as algorithmic tools but as analysis tools. For example, a major open problem is the size of the smallest grammar of a string particularly in comparison to the size of the Lempel–Ziv factorization of the same string [5]. This problem too involves pairs of identical substrings that overlap each other. The basic overlap graph was introduced in [2] but our algorithms reveal new combinatorial properties of this graph.

Acknowledgements. This work was supported by the Academy of Finland grant 118653 (ALGODAN) and by the Japan Society for the Promotion of Science.

References

1 Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *J. Discrete Algoritms*, 2(1):53–86, 2004.

2 Philip Bille, Johannes Fischer, Inge Li Gørtz, Tsvi Kopelowitz, Benjamin Sach, and Hjalte Wedel Vildhøj. Sparse suffix tree construction in small space. In *Proc. ICALP*, volume 7965 of *LNCS*, pages 148–159, 2013.

3 Gordon H. Bradley. Algorithm and bound for the greatest common divisor of n integers. *Commun. ACM*, 13(7):433–436, 1970.

4 Stefan Burkhardt and Juha Kärkkäinen. Fast lightweight suffix array construction and checking. In *Proc. CPM*, volume 2676 of *LNCS*, pages 55–69, 2003.

5 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, April Rasala, Amit Sahai, and abhi shelat. Approximating the smallest grammar: Kolmogorov complexity in natural models. In *Proc. STOC*, pages 792–801. ACM, 2002.

6 Martin Dietzfelbinger, Joseph Gil, Yossi Matias, and Nicholas Pippenger. Polynomial hash functions are reliable. In *Proc. ICALP*, volume 623 of *LNCS*, pages 235–246, 1992.

7 Nathan J Fine and Herbert S Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math. Soc.*, 16(1):109–114, 1965.

8 J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, 2006.

9 Juha Kärkkäinen. Fast BWT in small space by blockwise suffix sorting. *Theor. Comput. Sci.*, 387(3):249–257, 2007.

10 Richard M Karp and Michael O Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Develop.*, 31(2):249–260, 1987.

11 Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

Generalized Wong sequences and their applications to Edmonds' problems

Gábor Ivanyos[1], Marek Karpinski[2], Youming Qiao[3], and Miklos Santha[4]

1 **Institute for Computer Science and Control, Hungarian Academy of Sciences, Budapest, Hungary**
Gabor.Ivanyos@sztaki.mta.hu

2 **Department of Computer Science, University of Bonn, Bonn, Germany**
marek@cs.uni-bonn.de

3 **Centre for Quantum Technologies, National University of Singapore, Singapore 117543.**
cqtqy@nus.edu.sg

4 **LIAFA, Univ. Paris 7, CNRS, Paris, France / Centre for Quantum Technologies, National University of Singapore, Singapore**
miklos.santha@liafa.jussieu.fr

Abstract

We design two deterministic polynomial time algorithms for variants of a problem introduced by Edmonds in 1967: determine the rank of a matrix M whose entries are homogeneous linear polynomials over the integers. Given a linear subspace $\mathcal{B}$ of the $n \times n$ matrices over some field $\mathbb{F}$, we consider the following problems: *symbolic matrix rank* (SMR) is the problem to determine the maximum rank among matrices in $\mathcal{B}$, while *symbolic determinant identity testing* (SDIT) is the question to decide whether there exists a nonsingular matrix in $\mathcal{B}$. The constructive versions of these problems are asking to find a matrix of maximum rank, respectively a nonsingular matrix, if there exists one.

Our first algorithm solves the *constructive* SMR when $\mathcal{B}$ is spanned by unknown rank one matrices, answering an open question of Gurvits. Our second algorithm solves the constructive SDIT when $\mathcal{B}$ is spanned by triangularizable matrices, but the triangularization is not given explicitly. Both algorithms work over finite fields of size at least $n + 1$ and over the rational numbers, and the first algorithm actually solves (the non-constructive) SMR independent of the field size. Our main tool to obtain these results is to generalize Wong sequences, a classical method to deal with pairs of matrices, to the case of pairs of matrix spaces.

1998 ACM Subject Classification I.1.2 Algebraic algorithms

Keywords and phrases symbolic determinantal identity testing, Edmonds' problem, maximum rank matrix completion, derandomization, Wong sequences

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.397

1 Introduction

In [8] Edmonds introduced the following problem: Given a matrix M whose entries are homogeneous linear polynomials over the integers, determine the rank of M. The problem is the same as determining the maximum rank of a matrix in a linear space of matrices over the rationals. In this paper we consider the same question and certain of its variants over more general fields.



© Gábor Ivanyos, Marek Karpinski, Youming Qiao, and Miklos Santha;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 397–408



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



Let us denote by $M(n, \mathbb{F})$ the linear space of $n \times n$ matrices over a field $\mathbb{F}$. We call a linear subspace $\mathcal{B} \leq M(n, \mathbb{F})$ a *matrix space*. We define the *symbolic matrix rank* problem (SMR) over $\mathbb{F}$ as follows: given $\{B_1, \ldots, B_m\} \subseteq M(n, \mathbb{F})$, determine the maximum rank among matrices in $\mathcal{B} = \langle B_1, \ldots, B_m \rangle$, the matrix space spanned by B_i's. The *constructive* version of SMR is to find a matrix of maximum rank in $\mathcal{B}$ (this is called the maximum rank matrix completion problem in [12] and in [19]). We refer to the weakening of SMR, when the question is to decide whether there exists a nonsingular matrix in $\mathcal{B}$, as the *symbolic determinant identity testing* problem (SDIT), the name used by [20] (in [15] this variant is called Edmonds' problem). The *constructive* version in that case is to find a nonsingular matrix, if there is one in $\mathcal{B}$. We will occasionally refer to any of the above problems as *Edmonds' problem*.

The complexity of the SDIT depends crucially on the size of the underlying field $\mathbb{F}$. When $|\mathbb{F}|$ is a constant then it is NP-hard [5], on the other hand if the field size is large enough (say $\geq 2n$) then by the Schwartz-Zippel lemma [25, 30] it admits an efficient randomized algorithm [21]. Obtaining a deterministic polynomial-time algorithm for the SDIT would be of fundamental importance, since Kabanets and Impagliazzo [20] showed that such an algorithm would imply strong circuit lower bounds which seem beyond current techniques.

Previous works on Edmonds' problems mostly dealt with the case when the given *matrices* $B_1, \ldots, B_m$ satisfy certain *property*. For example, Lovász [22] considered several cases of SMR, including when the B_i's are of rank 1, and when they are skew symmetric matrices of rank 2. These classes were then shown to have deterministic polynomial-time algorithms [12, 23, 16, 13, 11, 19], see Section 1.1 for more details.

Another direction also studied is when instead of the given matrices, the generated *matrix space* $\mathcal{B} = \langle B_1, \ldots, B_m \rangle$ satisfies certain *property*. Since such a property is just a subset of all matrix spaces, we also call it a *class* of matrix spaces. Gurvits [15] has presented an efficient deterministic algorithm for the SDIT over $\mathbb{Q}$, when the matrix space satisfies the so called *Edmonds-Rado property*, whose definition we shall review in Section 1.1. For now we only note that this class includes $\mathbf{R_1}$, the class of *rank*-1 spanned matrix spaces, where a matrix space $\mathcal{B}$ is in $\mathbf{R_1}$ if and only if $\mathcal{B}$ has a basis consisting of rank-1 matrices. This fact was first shown by Lovász [22] via a theorem of Rado and Edmonds [24, 9, 28]. Gurvits stated as an open question the complexity of the SMR for $\mathbf{R_1}$ over finite fields [15, page 456].

The difference between properties of matrices and properties of matrix spaces is critical for Edmonds' problems. For example, given matrices $B_1, \ldots, B_m$, it is presumably hard[1] to determine whether $\mathcal{B} = \langle B_1, \ldots, B_m \rangle$ is in $\mathbf{R_1}$, and to find generating rank-1 matrices for $\mathcal{B}$. Thus the existence of algorithms for SMR when the B_i's are rank-1 does not immediately imply algorithms for matrix spaces in $\mathbf{R_1}$.

Our results are in line with Gurvits' work, namely we present algorithms for two classes of matrix spaces. To be specific, we consider $\mathbf{R_1}$, the class of rank-1 spanned matrix spaces, and the class of (upper-)triangularizable matrix spaces, where a matrix space $\mathcal{B} \leq M(n, \mathbb{F})$ is *triangularizable* if there exist nonsingular $C, D \in M(n, \mathbb{F}')$, where $\mathbb{F}'$ is some extension field of $\mathbb{F}$, such that for all $B \in \mathcal{B}$, the matrix DBC^{-1} is upper-triangular.

To ease the description of our results, we make a few definitions and notations. We denote by $\operatorname{rank}(B)$ the rank of a matrix B, and we set $\operatorname{corank}(B) = n - \operatorname{rank}(B)$. For a matrix space $\mathcal{B}$ we set $\operatorname{rank}(\mathcal{B}) = \max\{\operatorname{rank}(B) \mid B \in \mathcal{B}\}$ and $\operatorname{corank}(\mathcal{B}) = n - \operatorname{rank}(\mathcal{B})$. We say that $\mathcal{B}$ is *singular* if $\operatorname{rank}(\mathcal{B}) < n$, that is if $\mathcal{B}$ does not contain a nonsingular element, and *nonsingular*

[1] At present, we are not aware of the deterministic complexity of computing a rank-1 basis for matrix spaces in $\mathbf{R_1}$. Gurvits made a similar comment in [14].

otherwise. For a subspace $U \leq \mathbb{F}^n$, we set $\mathcal{B}(U) = \langle B(u) \mid B \in \mathcal{B}, u \in U \rangle$. Let c be a nonnegative integer. We say that U is a *c-singularity witness of* $\mathcal{B}$, if $\dim(U) - \dim(\mathcal{B}(U)) \geq c$, and U is a *singularity witness* of $\mathcal{B}$ if for some $c > 0$, it is a c-singularity witness.

Note that if there exists a singularity witness of $\mathcal{B}$ then $\mathcal{B}$ can only be singular. Let us define the *discrepancy* of $\mathcal{B}$ as $\mathrm{disc}(\mathcal{B}) = \max\{c \in \mathbb{N} \mid \exists\ c\text{-singularity witness of } \mathcal{B}\}$. Then it is also clear that $\mathrm{corank}(\mathcal{B}) \geq \mathrm{disc}(\mathcal{B})$. We now state our main theorems.

▶ **Theorem 1.** *Let $\mathbb{F}$ be either $\mathbb{Q}$ or a finite field. There is a deterministic polynomial-time algorithm which solves the SMR if $\mathcal{B}$ is spanned by rank-1 matrices. If the size of the field $\mathbb{F}$ is at least $n+1$, the algorithm solves the constructive SMR, and it also outputs a $\mathrm{corank}(\mathcal{B})$-singularity witness.*

▶ **Theorem 2.** *Let $\mathbb{F}$ be either $\mathbb{Q}$ or a finite field of size at least $n+1$. There is a deterministic polynomial-time algorithm which solves the constructive SDIT if $\mathcal{B}$ is triangularizable. Furthermore, over finite fields, when $\mathcal{B}$ is singular it also outputs a singularity witness.*

We remark that Theorem 1 remains true if we weaken the assumptions by only requiring that $\mathcal{B}$ is rank-1 spanned over some extension field of $\mathbb{F}$ rather than over $\mathbb{F}$. Also, instead of assuming that the whole space $\mathcal{B}$ is rank-1 spanned it is sufficient to suppose that a subspace of $\mathcal{B}$ of co-dimension one is spanned by rank-1 matrices. While the first extension can be achieved easily, the second extension requires some more work (though mostly technical).

1.1 Comparison with previous works

The idea of singularity witnesses was already present in Lovász's work [22]. Among other things, Lovász showed that for the rank-1 spanned case, the equality $\mathrm{corank}(\mathcal{B}) = \mathrm{disc}(\mathcal{B})$ holds, by reducing it to Edmonds' Matroid Intersection theorem [9], which in turn can be deduced from Rado's matroidal generalization of Hall's theorem [24] (see also [28]). Inspired by this fact, Gurvits defined the *Edmonds-Rado property* as the class of matrix spaces which are either nonsingular, or have a singularity witness. He listed several subclasses of the Edmonds-Rado class, including $\mathbf{R_1}$ (by the aforementioned result of Lovász) and triangularizable matrices. A well-known example of a matrix space without the Edmonds-Rado property is the linear space of skew symmetric matrices of size 3 [22].

As we stated already, Gurvits has presented a polynomial-time deterministic algorithm for the SDIT over $\mathbb{Q}$ for matrix spaces with the Edmonds-Rado property. Therefore over $\mathbb{Q}$, his algorithm covers the SDIT for $\mathbf{R_1}$ and for triangularizable matrices. Our algorithms are valid not only over $\mathbb{Q}$ but also over finite fields. In the triangularizable case we also deal with the SDIT, but for $\mathbf{R_1}$ we solve the more general SMR. In fact, it is not hard to reduce SMR for the general to SMR for the triangularizable case (see Lemma 26 in [18]), so solving SMR for the triangularizable case is as hard as the general case. In both cases the algorithms solve the constructive version of the problems, and they also construct singularity witnesses, except for the SDIT over the rationals. Finally, they work in polynomial time when the field size is at least $n+1$. Moreover, for $\mathbf{R_1}$ the algorithm solves the non constructive SMR in polynomial time regardless of the field size, settling the open problem of Gurvits.

Over fields of constant size, the SMR has certain practical implications [16, 17], but is shown to be NP-hard [5] in general. Some special cases have been studied, mostly in the form of the *mixed matrices*, that is linear matrices where each entry is either a variable or a field element. Then by restricting the way variables appear in the matrices some cases turn out to have efficient deterministic algorithms, including when every variable appears at most once ([16], building on [12, 23]), and when the mixed matrix is skew-symmetric

and every variable appears at most twice ([13, 11]). Finally in [19], Ivanyos, Karpinski and Saxena present a deterministic polynomial-time algorithm for the case when among the input matrices $B_1, \ldots, B_m$ all but B_1 are of rank 1.

As a computational model of polynomials, determinants with affine polynomial entries turn out to be equivalent to algebraic branching programs (ABPs) [27, 4] up to a polynomial overhead. Thus the identity test for ABPs is the same as SDIT. For restricted classes of ABPs, (quasi)polynomial-time deterministic identity test algorithms have been devised (cf. [10] and the references therein). Note that identity test results for SDIT and ABPs are in general incomparable. For an application of SDIT to quantum information processing see [6].

Let us comment briefly on the main technical tool we use in our algorithms. We generalize the first and second Wong sequences for matrix pencils (essentially two-dimensional matrix spaces) which have turned out to be useful among others in the area of linear differential-algebraic equations (see the recent survey [26]). These were originally defined in [29] for a pair of matrices (A, B), and were recently used to compute the Kronecker normal form in a numerical stable way [2, 3]. We generalize Wong sequences to the case $(\mathcal{A}, \mathcal{B})$ where $\mathcal{A}$ and $\mathcal{B}$ are matrix spaces, and show that they have analogous basic properties to the original ones. We relate the generalized Wong sequences to Edmonds' problems via singularity witnesses. Essentially this connection allows us to design the algorithm for $\mathbf{R_1}$ using the second Wong sequence, and the algorithm for triangularizable matrix spaces using the first Wong sequence. We remark that techniques similar to the second Wong sequence were already used in [19].

Organization. In Section 2 we define Wong sequences of a pair of matrix spaces, and present their basic properties. In Section 3 the connection between the second Wong sequence and singularity witnesses is shown. Based on this connection we introduce the power overflow problem, and reduce the SMR to it. We also prove here Theorem 1 under the hypothesis that there is a polynomial time algorithm for the power overflow problem. In Section 4 we show an algorithm for the power overflow problem that works in polynomial time for rank-1 spanned matrix spaces. In Section 5 the algorithm for Theorem 2 is outlined, which works for triangularizable matrix spaces. The readers are referred to the full version [18] for certain missing details, and some discussion on the Edmonds-Rado class and some subclasses.

2 Wong sequences for pairs of matrix spaces

For $n \in \mathbb{N}$, we set $[n] = \{1, \ldots, n\}$. We use 0 to denote the zero vector space. In this section we generalize the classical Wong sequences of matrix pencils to the situation of pairs of matrix subspaces. This is the main technical tool in this work. Let V and V' be finite dimensional vector spaces over a field $\mathbb{F}$, and let $\mathrm{Lin}(V, V')$ be the vector space of linear maps from V to V'. We set $n = \dim(V)$ and $n' = \dim(V')$. For $A \in \mathrm{Lin}(V, V')$, and linear subspaces $\mathcal{A} \leq \mathrm{Lin}(V, V')$, $U \leq V$ and $W \leq V'$, we define $A(U) = \{A(u) \mid u \in U\}$, $\mathcal{A}(U) = \langle \{A(u) \mid A \in \mathcal{A}, u \in U\} \rangle$, $A^{-1}(W) = \{v \in V \mid A(v) \in W\}$, and $\mathcal{A}^{-1}(W) = \{v \in V \mid \forall A \in \mathcal{A}, A(v) \in W\}$. Observe that $A(U)$, $\mathcal{A}(U)$ are linear subspaces of V', whereas $A^{-1}(W)$ and $\mathcal{A}^{-1}(W)$ are subspaces of V. Also note that $\mathcal{A}(U) = \langle \cup_{A \in \mathcal{A}} A(U) \rangle$ and $\mathcal{A}^{-1}(W) = \cap_{A \in \mathcal{A}} A^{-1}(W)$. Moreover, if $\mathcal{A}$ is spanned by $\{A_1, \ldots, A_m\}$, then $\mathcal{A}(U) = \langle \cup_{i \in [m]} A_i(U) \rangle$, and $\mathcal{A}^{-1}(W) = \cap_{i \in [m]} A_i^{-1}(W)$. Some easy and useful facts are the following.

▶ Fact 3. For $\mathcal{A}, \mathcal{B} \leq \mathrm{Lin}(V, V')$, and $U, S \leq V$, $W, T \leq V'$, we have:

1. If $U \subseteq S$ and $W \subseteq T$, then $\mathcal{A}(U) \subseteq \mathcal{A}(S)$ and $\mathcal{A}^{-1}(W) \subseteq \mathcal{A}^{-1}(T)$;
2. If $\mathcal{B}(U) \subseteq \mathcal{A}(U)$ and $\mathcal{B}(S) \subseteq \mathcal{A}(S)$, then $\mathcal{B}(\langle U \cup S \rangle) \subseteq \mathcal{A}(\langle U \cup S \rangle)$;
3. If $\mathcal{B}^{-1}(W) \supseteq \mathcal{A}^{-1}(W)$ and $\mathcal{B}^{-1}(T) \supseteq \mathcal{A}^{-1}(T)$, then $\mathcal{B}^{-1}(W \cap T) \supseteq \mathcal{A}^{-1}(W \cap T)$;
4. $\mathcal{A}^{-1}(\mathcal{A}(U)) \supseteq U$, and $\mathcal{A}(\mathcal{A}^{-1}(W)) \subseteq W$.

We now define the two Wong sequences for a pair of matrix subspaces.

▶ **Definition 4.** Let $\mathcal{A}, \mathcal{B} \leq \mathrm{Lin}(V, V')$. The sequence of subspaces $(U_i)_{i\in\mathbb{N}}$ of V is called the *first Wong sequence of* $(\mathcal{A}, \mathcal{B})$, where $U_0 = V$, and $U_{i+1} = \mathcal{B}^{-1}(\mathcal{A}(U_i))$. The sequence of subspaces $(W_i)_{i\in\mathbb{N}}$ of V' is called the *second Wong sequences of* $(\mathcal{A}, \mathcal{B})$, where $W_0 = 0$, and $W_{i+1} = \mathcal{B}(\mathcal{A}^{-1}(W_i))$.

When $\mathcal{A} = \langle A \rangle$ and $\mathcal{B} = \langle B \rangle$ are one dimensional matrix spaces, the Wong sequences for $(\mathcal{A}, \mathcal{B})$ coincide with the classical Wong sequences for the matrix pencil $Ax - B$ [29, 2]. The following properties are straightforward generalizations of those for classical Wong sequences. We start by considering the first Wong sequence.

▶ Proposition 5. Let $(U_i)_{i\in\mathbb{N}}$ be the first Wong sequence of $(\mathcal{A}, \mathcal{B})$. Then for all $i \in \mathbb{N}$, we have $U_{i+1} \subseteq U_i$. Furthermore, $U_{i+1} = U_i$ if and only if $\mathcal{B}(U_i) \subseteq \mathcal{A}(U_i)$.

Proof. Firstly we show that $U_{i+1} \subseteq U_i$, for every $i \in \mathbb{N}$. For $i = 0$, this holds trivially. For $i > 0$, by Fact 3 (1) we get $U_{i+1} = \mathcal{B}^{-1}(\mathcal{A}(U_i)) \subseteq \mathcal{B}^{-1}(\mathcal{A}(U_{i-1})) = U_i$, since $U_i \subseteq U_{i-1}$.

Suppose now that $\mathcal{B}(U_i) \subseteq \mathcal{A}(U_i)$, for some i. Then $U_i \subseteq \mathcal{B}^{-1}(\mathcal{B}(U_i)) \subseteq \mathcal{B}^{-1}(\mathcal{A}(U_i))$ respectively by Fact 3 (4) and (1), which gives $U_{i+1} = U_i$. If $\mathcal{B}(U_i) \not\subseteq \mathcal{A}(U_i)$ then there exist $B \in \mathcal{B}$ and $v \in U_i$ such that $B(v) \notin \mathcal{A}(U_i)$. Thus $v \notin \mathcal{B}^{-1}(\mathcal{A}(U_i)) = U_{i+1}$, which gives $U_{i+1} \subset U_i$. ◀

Given Proposition 5, we see that the first Wong sequence stabilizes after at most n steps at some subspace. That is, for any $(\mathcal{A}, \mathcal{B})$, there exists $\ell \in \{0, \ldots, n\}$, such that $U_0 \supset U_1 \supset \cdots \supset U_\ell = U_{\ell+1} = \ldots$. In this case we call the subspace U_ℓ the *limit* of $(U_i)_{i\in\mathbb{N}}$, and we denote it by U^*.

▶ Proposition 6. U^* is the largest subspace $T \leq V$ such that $\mathcal{B}(T) \subseteq \mathcal{A}(T)$.

Proof. By Proposition 5 we know that U^* satisfies $\mathcal{B}(U^*) \subseteq \mathcal{A}(U^*)$. Consider an arbitrary $T \leq V$ such that $\mathcal{B}(T) \subseteq \mathcal{A}(T)$, we show by induction that $T \subseteq U_i$, for all i. When $i = 0$ this trivially holds. Suppose that $T \subseteq U_i$, for some i. Then by repeated applications of Fact 3 we have $T \subseteq \mathcal{B}^{-1}(\mathcal{B}(T)) \subseteq \mathcal{B}^{-1}(\mathcal{A}(T)) \subseteq \mathcal{B}^{-1}(\mathcal{A}(U_i)) = U_{i+1}$. ◀

Analogous properties hold for the second Wong sequence $(W_i)_{i\in\mathbb{N}}$. In particular the sequence stabilizes after at most n' steps, and there exists a *limit* subspace W^* of $(W_i)_{i\in\mathbb{N}}$. We summarize them in the following proposition.

▶ Proposition 7. Let $(W_i)_{i\in\mathbb{N}}$ be the second Wong sequence of $(\mathcal{A}, \mathcal{B})$. Then

1. $W_{i+1} \supseteq W_i$, for all $i \in \mathbb{N}$. Furthermore, $W_{i+1} = W_i$ if and only if $\mathcal{B}^{-1}(W_i) \supseteq \mathcal{A}^{-1}(W_i)$.
2. The limit subspace W^* is the smallest subspace $T \leq V'$ s.t. $\mathcal{B}^{-1}(T) \supseteq \mathcal{A}^{-1}(T)$.

It is worth noting that the second Wong sequence can be viewed as the dual of the first one in the following sense. Assume that V and V' are equipped with nonsingular symmetric bilinear forms, both denoted by $\langle , \rangle$. For a linear map $A : V \to V'$ let $A^{\mathrm{T}} : V' \to V$ stand for the transpose of A with respect to $\langle , \rangle$. This is the unique map with the property $\langle A^{\mathrm{T}}(u), v\rangle = \langle u, A(v)\rangle$, for all $u \in V'$ and $v \in V$. For a matrix space $\mathcal{A}$, let $\mathcal{A}^{\mathrm{T}}$ be the space $\{A^{\mathrm{T}} | A \in \mathcal{A}\}$. For $U \leq V$, the orthogonal subspace of U is defined as $U^\perp = \{v \in V \mid \langle v, u\rangle = 0$ for all $u \in U\}$. Similarly we define $W^\perp$ for $W \leq V'$. Then we have $((\mathcal{A}^{\mathrm{T}})^{-1}(U)))^\perp = \mathcal{A}(U^\perp)$, and $(\mathcal{A}^{\mathrm{T}}(V))^\perp = \mathcal{A}^{-1}(V^\perp)$. It can be verified that if $(W_i)_{i\in\mathbb{N}}$ is the second Wong sequence of $(\mathcal{A}, \mathcal{B})$ and $(U_i)_{i\in\mathbb{N}}$ the first Wong sequence of $(\mathcal{A}^{\mathrm{T}}, \mathcal{B}^{\mathrm{T}})$, then $W_i = U_i^\perp$. We note that the duality of Wong sequences was, already derived in [2] for pairs of matrices.

For a matrix space $\mathcal{A}$ and a subspace $U \leq V$ given in terms of a basis we can compute $\mathcal{A}(U)$ by applying the basis elements for $\mathcal{A}$ to those of U and then selecting a maximal set of

linearly independent vectors. A possible way of computing $\mathcal{A}^{-1}(U)$ for $U \leq V'$ is to compute first $U^\perp$, then $\mathcal{A}^{\mathrm{T}}(U^\perp)$ and finally $\mathcal{A}^{-1}(U) = (\mathcal{A}^{\mathrm{T}}(U^\perp))^\perp$. Therefore we have

▶ Proposition 8. Wong sequences can be computed using $(n+n')^{O(1)}$ arithmetic operations.

Unfortunately, we are unable to prove that over the rationals the bit length of the entries of the bases describing the Wong sequences remain polynomially bounded in the length of the data for $\mathcal{A}$ and $\mathcal{B}$. However, in Section 3.1 we show that if $\mathcal{A} = \langle A \rangle$, then the first few members of the second Wong sequence which happen to be contained in $\mathrm{im}(A)$ can be computed in polynomial time using an iteration of multiplying vectors by matrices from a basis for $\mathcal{B}$ and by a pseudo-inverse of A.

We also observe that if we consider the bases for $\mathcal{A}$ and $\mathcal{B}$ as matrices over an extension field $\mathbb{F}'$ of $\mathbb{F}$ then the members of the Wong sequences over $\mathbb{F}'$ are just the $\mathbb{F}'$-linear spaces spanned by the corresponding members of the Wong sequences over $\mathbb{F}$. In particular, the limit of the first Wong sequence over $\mathbb{F}$ is nontrivial if and only if the limit of the first Wong sequence over $\mathbb{F}'$ is nontrivial.

3 The second Wong sequence and singularity witnesses

3.1 The connection

As in Section 2, let V and V' be finite dimensional vector spaces over a field $\mathbb{F}$, of respective dimensions n and n'. For $A \in \mathrm{Lin}(V, V')$ we set $\mathrm{corank}(A) = \dim(\ker(A))$. For $\mathcal{B} \leq \mathrm{Lin}(V, V')$, the concepts of c-singularity witnesses, $\mathrm{disc}(\mathcal{B})$ and $\mathrm{corank}(\mathcal{B})$, defined for the case when $n = n'$, can be generalized naturally to $\mathcal{B}$. We also have that $\mathrm{corank}(\mathcal{B}) \geq \mathrm{disc}(\mathcal{B})$, and that a $\mathrm{corank}(\mathcal{B})$-singularity witness of $\mathcal{B}$ does not exist necessarily. Let $A \in \mathcal{B}$, and consider $(W_i)_{i \in \mathbb{N}}$, the second Wong sequence of $(A, \mathcal{B})$. The next lemma states that the limit W^* is basically such a witness under the condition that it is contained in the image of A. Moreover, in this specific case the limit can be computed efficiently.

▶ **Lemma 9.** *Let $A \in \mathcal{B} \leq \mathrm{Lin}(V, V')$, and let W^* be the limit of the second Wong sequence of $(A, \mathcal{B})$. There exists a $\mathrm{corank}(A)$-singularity witness of $\mathcal{B}$ if and only if $W^* \subseteq \mathrm{im}(A)$. If this is the case, then A is of maximum rank and $A^{-1}(W^*)$ is a $\mathrm{corank}(\mathcal{B})$-singularity witness.*

Proof. We prove the equivalence. Firstly suppose that $W^* \subseteq \mathrm{im}(A)$. Then $\dim(A^{-1}(W^*)) = \dim(W^*) + \dim(\ker(A))$. Since $W^* = \mathcal{B}(A^{-1}(W^*))$ and $\dim(\ker(A)) = \mathrm{corank}(A)$, it follows that $A^{-1}(W^*)$ is a $\mathrm{corank}(A)$-singularity witness of $\mathcal{B}$.

Let us now suppose that some $U \leq V$ is a $\mathrm{corank}(A)$-singularity witness, that is $\dim(U) - \dim(\mathcal{B}(U)) \geq \mathrm{corank}(A)$. Then $\dim(U) - \dim(A(U)) \geq \mathrm{corank}(A)$ because $A \in \mathcal{B}$. Since the reverse inequality always holds without any condition on U, we have $\dim(U) - \dim(A(U)) = \mathrm{corank}(A)$. Similarly we have $\dim(U) - \dim(\mathcal{B}(U)) = \mathrm{corank}(A)$ which implies that $\dim(A(U)) = \dim(\mathcal{B}(U))$, and therefore $A(U) = \mathcal{B}(U)$. For a subspace $S \leq V$ the equality $\dim(S) - \dim(A(S)) = \mathrm{corank}(S)$ is equivalent to $\ker(A) \subseteq S$, thus we have $\ker(A) \subseteq U$ from which it follows that $U = A^{-1}(A(U))$. But then $\mathcal{B}^{-1}(A(U)) = \mathcal{B}^{-1}(\mathcal{B}(U)) \supseteq U = A^{-1}(A(U))$. Since W^* is the smallest subspace $T \leq V'$ satisfying $\mathcal{B}^{-1}(T) \supseteq A^{-1}(T)$, we can conclude that $W^* \subseteq A(U)$.

The existence of a $\mathrm{corank}(A)$-singularity witness obviously implies that A is of maximum rank, and when $W^* \subseteq \mathrm{im}(A)$ we have already seen that $A^{-1}(W^*)$ is a $\mathrm{corank}(A)$-singularity witness of $\mathcal{B}$. Since $\mathrm{corank}(A) = \mathrm{corank}(\mathcal{B})$, it is also a $\mathrm{corank}(\mathcal{B})$-singularity witness. ◀

We would like to find an efficient way of testing whether $W^* \subseteq \mathrm{im}(A)$ for a given $A \in \mathcal{B}$. In the computation of the limit W^* of the second Wong sequence of $(A, \mathcal{B})$ the

computationally hard step is applying iteratively A^{-1}. We overcome this difficulty by introducing a pseudo-inverse of A in the computation. We describe now this method.

Let $n = \dim(V)$ and $n' = \dim(V')$. First of all we assume without loss of generality that $n = n'$. Indeed, if $n < n'$ we can add as a direct complement a suitable space to V on which $\mathcal{B}$ acts as zero, and if $n > n'$, we can embed V' into a larger space. In terms of matrices, this means augmenting the elements of $\mathcal{B}$ by zero columns or zero rows to obtain square matrices. This procedure affects neither the ranks of the matrices in $\mathcal{B}$ nor the singularity witnesses.

We say that a nonsingular linear map $A' : V' \to V$ is a *pseudo-inverse* of A if the restriction of A' to $\mathrm{im}(A)$ is the inverse of the restriction of A to a direct complement of $\ker(A)$. Such a map can be efficiently constructed as follows. Choose a direct complement U of $\ker(A)$ in V as well as a direct complement U' of $\mathrm{im}(A)$ in V'. Then take the map $A'_0 : \mathrm{im}(A) \to U$ such that AA'_0 is the identity of $\mathrm{im}(A)$ and take an arbitrary nonsingular linear map $A'_1 : U' \to \ker(A)$. Finally let A' be the direct sum of A'_0 and A'_1.

▶ **Lemma 10.** *Let $A \in \mathcal{B} \leq \mathrm{Lin}(V, V')$ and let A' be a pseudo-inverse of A. There exists a $\mathrm{corank}(A)$-singularity witness of $\mathcal{B}$ if and only if $(\mathcal{B}A')^i(\ker(AA')) \subseteq \mathrm{im}(A)$, for all $i \in [n]$. This can be tested in polynomial time, and if the condition holds then A is of maximum rank and $A'(W^*)$ is a $\mathrm{corank}(\mathcal{B})$-singularity witness which also can be computed deterministically in polynomial time.*

Proof. It follows from Lemma 9 that a $\mathrm{corank}(A)$-singularity witness exists if and only if $W_i \subseteq \mathrm{im}(A)$, for $i = 1, \ldots, n$. Observing that $(\mathcal{B}A')^i(\ker(AA')) \subseteq W_i$ for $i = 1, \ldots, n$, to prove the equivalence it is sufficient to show that if $(\mathcal{B}A')^i(\ker(AA')) \subseteq \mathrm{im}(A)$ for $i = 1, \ldots, n$ then $W_i = (\mathcal{B}A')^i(\ker(AA'))$ for $i = 1, \ldots, n$. The proof is by induction. For $i = 1$ the claim $W_1 = \mathcal{B}A'(\ker(AA'))$ holds since $\ker(AA') = A'^{-1}(\ker(A))$. For $i > 1$, by definition $W_i = \mathcal{B}A^{-1}(W_{i-1})$. Since every subspace $W \leq \mathrm{im}(A)$ satisfies $A^{-1}W = A'W + \ker(A)$, where $+$ denotes the direct sum, we get $W_i \subseteq \mathcal{B}A'(W_{i-1}) + \mathcal{B}(\ker(A))$. Observe that $\mathcal{B}(\ker(A)) = W_1$. We will show that $W_1 \subseteq \mathcal{B}A'(W_{i-1})$ and then we conclude by the inductive hypothesis. We know that $W_1 \subseteq W_{i-1}$ from the properties of the Wong sequence, therefore it is sufficient to show that $W_{i-1} \subseteq \mathcal{B}A'(W_{i-1})$. But $W_{i-1} = AA'(W_{i-1})$ since $W_i \subseteq \mathrm{im}(A)$ and A' is the inverse of A on $\mathrm{im}(A)$.

Based on this equivalence, testing the existence of a $\mathrm{corank}(A)$-singularity witness can be accomplished by a simple algorithm, [18, Lemma 10] for details.

If we find that the condition holds then $A'(W^*)$ by Lemma 9 is a $\mathrm{corank}(\mathcal{B})$-singularity witness, and it can be easily computed from W^*. ◀

3.2 The power overflow problem

For $A \in \mathcal{B} \leq \mathrm{Lin}(V, V')$, we would like to know whether A is of maximum rank in $\mathcal{B}$. With the help of the limit W^* of the second Wong sequence of $(A, \mathcal{B})$ we have established a sufficient condition: we know that if $W^* \subseteq \mathrm{im}(A)$ then A is indeed of maximum rank. Our results until now do not give a necessary condition for the maximum rank. Now we show that the second Wong sequence actually allows to translate this question to the *power overflow* problem (PO) which we define below. As a consequence an efficient solution of the PO guarantees an efficient solution for the SMR. The reduction is mainly based on a theorem of Atkinson and Stephens [1] which essentially says that over big enough fields, in 2-dimensional matrix spaces $\mathcal{B}$, the equality $\mathrm{corank}(\mathcal{B}) = \mathrm{disc}(\mathcal{B})$ holds.

▶ Proposition 11 ([1]). Assume that $|\mathbb{F}| > n$, and let $A, B \in \mathrm{Lin}(V, V')$. If A is a maximum rank element of $\langle A, B\rangle$ then there exists a $\mathrm{corank}(A)$-singularity witness of $\langle A, B\rangle$.

Combining Lemma 10 and Proposition 11 we get also an equivalent condition for A being of maximum rank.

▶ **Lemma 12.** *Assume that $|\mathbb{F}| > n$. Let $A \in \mathcal{B} \leq \mathrm{Lin}(V, V')$, and let A' be a pseudo-inverse of A. Then A is of maximum rank in $\mathcal{B}$ if and only if for every $B \in \mathcal{B}$ and for all $i \in [n]$, we have*

$$(BA')^i(\ker(AA')) \subseteq \mathrm{im}(A).$$

Proof. First observe that A is of maximum rank in $\mathcal{B}$ if and only if for every $B \in \mathcal{B}$, it is of maximum rank in $\langle A, B\rangle$. For a fixed B, by Proposition 11 and Lemma 10, A is of maximum rank in $\langle A, B\rangle$ exactly when $(\langle B, A\rangle A')^i(\ker(AA')) \subseteq \mathrm{im}(A)$, for all $i \in [n]$. From that we can conclude since A' is the inverse of A on $\mathrm{im}(A)$. ◀

This lemma leads us to reduce the problems of deciding if A is of the maximum rank, and finding a matrix of rank larger than A when this is not the case, to the following question.

▶ Problem 13 (The power overflow problem). Given $\mathcal{D} \leq M(n, \mathbb{F})$, $U \leq \mathbb{F}^n$ and $U' \leq \mathbb{F}^n$, output $D \in \mathcal{D}$ and $\ell \in [n]$ s.t. $D^\ell(U) \not\subseteq U'$, if there exists such (D, ℓ). Otherwise say **no**.

The power overflow problem admits an efficient randomized algorithm when $|\mathbb{F}| = \Omega(n)$. For the rank-1 spanned case we show a deterministic solution regardless of the field size.

▶ **Theorem 14.** *Let $\mathcal{D} \leq M(n, \mathbb{F})$ be spanned by rank-1 matrices. Then there exists $D \in \mathcal{D}$ and $\ell \in [n]$ such that $D^\ell(U) \not\subseteq U'$ if and only if there exists $\ell \in [n]$ such that $\mathcal{D}^\ell(U) \not\subseteq U'$. The power overflow problem for $\mathcal{D}$ can be solved deterministically in polynomial time.*

Using this result whose proof is given in Section 4 we are now ready to prove Theorem 1.

Proof of Theorem 1. First we suppose that $|\mathbb{F}| \geq n + 1$. Let A be an arbitrary matrix in $\mathcal{B}$. The algorithm iterates the following process until A becomes of maximum rank.

We run the algorithm of Lemma 10 to test whether $(\mathcal{B}A')^i(\ker(AA')) \subseteq \mathrm{im}(A)$ for $i \in [n]$. If this condition holds then A is of maximum rank, and the algorithm also gives a $\mathrm{corank}(\mathcal{B})$-singularity witness. Otherwise we know by Theorem 14 that there exists $B \in \mathcal{B}$ and $i \in [n]$ such that $(BA')^i(\ker(AA')) \not\subseteq \mathrm{im}(A)$. We apply the algorithm of Theorem 14 with input $\mathcal{B}A'$, $\ker(AA')$ and $\mathrm{im}(A)$, which finds such a couple (B, i). Lemma 12 applied to $\langle A, B\rangle$ implies that A is not of maximum rank in $\langle A, B\rangle$. If A has rank $r \leq n - 1$ which is not maximal in $\langle A, B\rangle$, then the determinant of an appropriate $(r + 1) \times (r + 1)$ minor is a nonzero polynomial of degree at most $r + 1$ which has at most $r + 1 \leq n$ roots. We then pick $n + 1$ arbitrary field elements $\lambda_1, \ldots, \lambda_{n+1}$, and we know that for some $1 \leq j \leq n + 1$ we have $\mathrm{rank}(A + \lambda_j B) > \mathrm{rank}(A)$. We replace A by $A + \lambda_j B$ and restart the process.

At the end of each iteration, by a reduction procedure described in [7] we can achieve that the matrix A, written as a linear combination of $B_1, \ldots, B_m$ has coefficients from a fixed subset $K \subseteq \mathbb{F}$ of size $n + 1$. In fact, if $A = \alpha_1 B_1 + \alpha_2 B_2 \ldots + \alpha_m B_m$ has rank r then for at least one $\kappa_1 \in K$ the matrix $\kappa_1 B_1 + \alpha_2 B_2 \ldots + \alpha_m B_m$ has rank at least r. This way all the coefficients α_j can be replaced with an appropriate element from K.

As in each iteration we either stop (and conclude with A being of maxiaml rank), or increase the rank of A by at least 1, the number of iterations is at most n. Also, each iteration takes polynomial many steps since the processes of Lemma 10 and Theorem 14 are polynomial. Therefore the overall running time is also polynomial. ◀

We can compute the maximum rank over a field of size less than $n + 1$ by running the above procedure over a sufficiently large extension field. The maximum rank will not grow if we go over an extension. This follows from the fact that the equality $\mathrm{corank}(\mathcal{B}) = \mathrm{disc}(\mathcal{B})$ holds over any field if $\mathcal{B}$ is spanned by an arbitrary matrix and by rank one matrices, see [19].

4 The power overflow problem for rank-1 spanned matrix spaces

In this section we prove **Theorem 14**. Given subspaces U, U' of $\mathbb{F}^n$ as well as a basis $\{D_1, \ldots, D_m\}$ for a matrix space $\mathcal{D} \leq M(n, \mathbb{F})$, we will show is that in polynomial time we can decide if $\mathcal{D}^\ell(U) \not\subseteq U'$ for some ℓ, and if this holds then find $D \in \mathcal{D}$ s.t. $D^\ell(U) \not\subseteq U'$.

Formally let $\ell = \ell(\mathcal{D})$ be the smallest integer j s.t. $\mathcal{D}^j(U) \not\subseteq U'$ if such an integer exists, and n otherwise. We start by computing ℓ and for $1 \leq j \leq \ell$, bases $\mathcal{T}_j$ for $\mathcal{D}^j$. Set $\mathcal{T}_1 = \{D_1, \ldots, D_m\}$. If $\mathcal{D}^j(U) \not\subseteq U'$ then we set $\ell = j$ and stop constructing further bases. If $j = n$ and $\mathcal{D}^n(U) \subseteq U'$ then we stop the algorithm and output **no**. Otherwise we compute $\mathcal{T}_{j+1}$ by selecting a maximal linearly independent set form the products of elements in $\mathcal{T}_j$ and $\mathcal{T}_1$.

We are now looking for D such that $D^\ell(U) \not\subseteq U'$. For $i \in [\ell]$, we define subspaces $\mathcal{H}_i$ of $\mathcal{D}$, which play a crucial role in the algorithm:

$$\mathcal{H}_i = \{X \in \mathcal{D} \mid \mathcal{D}^{\ell-j} X \mathcal{D}^{j-1}(U) \subseteq U',\ j = 1, \ldots, i-1, i+1, \ldots, \ell\}.$$

That is, $X \in \mathcal{H}_i$ if and only if whenever X appears in a place other than the ith in a product P of ℓ elements from $\mathcal{D}$ then $P(U) \subseteq U'$. The subspaces $\mathcal{H}_i$ can be computed as follows. Let $x_1, \ldots, x_m$ be formal variables, an element in $\mathcal{D}$ can be written as $X = \sum_{k \in [m]} x_k D_k$. The condition $\mathcal{D}^{\ell-j} X \mathcal{D}^{j-1}(U) \subseteq U'$ is equivalent to the set of the following homogeneous linear equations in the variables x_k: $\langle Z(\sum_{k \in [m]} x_k D_k) Z' u, v \rangle = 0$, where Z is from $\mathcal{T}_{\ell-j}$, Z' is from $\mathcal{T}_{j-1}$, u is from a basis for U and v is from a basis for $U'^{\perp}$. Thus $\mathcal{H}_i$ can be computed by solving a system of polynomially many homogeneous linear equations. Note that the coefficients of the equations are scalar products of vectors from a basis for $U'^{\perp}$ by vectors obtained as applying products of ℓ matrices from $\{D_1, \ldots, D_m\}$ to basis elements for U. The definition of $\mathcal{H}_i$ implies the following.

▶ **Lemma 15.** *For a matrix $X = X_1 + \ldots + X_\ell$ with $X_i \in \mathcal{H}_i$, we have $X^\ell(U) \subseteq U'$ if and only if $X_\ell \cdots X_2 X_1(U) \subseteq U'$.*

Proof. We have $X^m = \sum_\sigma X_{\sigma(\ell)} \cdots X_{\sigma(1)}$, where the summation is over the maps $\sigma : [\ell] \to [\ell]$. When σ is not the identity map then there exists an index j such that $\sigma(j) \neq j$. Then $X_{\sigma(\ell)} \cdots X_{\sigma(1)}(U) \subseteq U'$ by the definition of $\mathcal{H}_{\sigma(j)}$. ◀

In general, $\mathcal{H}_i$ can be 0. In our setting, due to the existence of rank one generators, fortunately this is far from the case. Recall that ℓ is the smallest integer such that $\mathcal{D}^\ell(U) \not\subseteq U'$.

▶ **Lemma 16.** *We have $\mathcal{H}_\ell \cdots \mathcal{H}_1(U) \not\subseteq U'$.*

Proof. Assume that $\mathcal{D}$ is spanned by the rank one matrices $C_1, \ldots, C_m$. Then there exist indices $k_1, \ldots, k_\ell$ such $C_{k_\ell} \cdots C_{k_1}(U) \not\subseteq U'$. We show that $C_{k_i} \in \mathcal{H}_i$, for $i \in [\ell]$, this implies immediately $\mathcal{H}_\ell \cdots \mathcal{H}_1(U) \not\subseteq U'$. Assume by contradiction that $C_{k_i} \notin \mathcal{H}_i$, for some $i \in [\ell]$. Then $\mathcal{D}^{\ell-j} C_{k_i} \mathcal{D}^{j-1}(U) \not\subseteq U'$, for some $j \neq i$. On the other hand C_{k_i} satisfies $\mathcal{D}^{\ell-i} C_{k_i} \mathcal{D}^{i-1}(U) \not\subseteq U'$. Since C_{k_i} is of rank 1 we have $C_{k_i} \mathcal{D}^{j-1}(U) = C_{k_i} \mathcal{D}^{i-1}(U)$, which yields that neither $\mathcal{D}^{\ell-i} C_{k_i} \mathcal{D}^{j-1}(U)$ nor $\mathcal{D}^{\ell-j} C_{k_i} \mathcal{D}^{i-1}(U)$ is contained in U'. However one of these products is shorter than ℓ, contradicting the minimality of ℓ. ◀

To finish the algorithm, we compute bases for products $\mathcal{H}_i \cdots \mathcal{H}_1$, for $i \in [n]$, in a way similar to computing bases for $\mathcal{D}^i$. Then we search the basis of $\mathcal{H}_\ell$ for an element Z such that $Z\mathcal{H}_{\ell-1} \cdots \mathcal{H}_1(U) \not\subseteq U'$. We put $X_\ell = Z$ and continue searching the basis of $\mathcal{H}_{\ell-1}$ for an element Z such that $X_\ell Z \mathcal{H}_{\ell-2} \cdots \mathcal{H}_1(U) \not\subseteq U'$. Continuing the iteration, Lemma 16 ensures that eventually we find $X_i \in \mathcal{H}_i$, for $i \in [\ell]$, such that $X_\ell \cdots X_1(U) \not\subseteq U'$. We set $D = X_1 + \ldots + X_\ell$, then by Lemma 15 we have $D^\ell(U) \not\subseteq U'$. We return D and ℓ. □

5 The first Wong sequence and triangularizable matrix spaces

Here we only give a proof outline of Theorem 2, and the reader is referred to the full version [18, Section 5] for details. Our task is to determine whether there exists a nonsingular matrix in a triangularizable matrix space, and finding such a matrix if exists. Let $\mathbb{F}'$ be an extension field of $\mathbb{F}$, and recall that $\mathcal{B} \leq M(n,\mathbb{F})$ is triangularizable if there exist nonsingular $C, D \in M(n,\mathbb{F}')$, s.t. $\forall B \in \mathcal{B}$, DBC^{-1} is upper triangular. Our starting point is the following lemma, which connects first Wong sequences with singularity witnesses.

▶ **Lemma 17.** *Let $A \in \mathcal{B} \leq M(n,\mathbb{F})$, and let U^* be the limit of the first Wong sequence of $(A,\mathcal{B})$. Set $d = \dim(U^*)$. Then either U^* is a singularity witness of $\mathcal{B}$, or there exist nonsingular matrices $P, Q \in M(n,\mathbb{F})$, such that $\forall B \in \mathcal{B}$, QBP^{-1} is of the form $\begin{bmatrix} X & Y \\ 0 & Z \end{bmatrix}$, where X is of size $d \times d$, and $\mathcal{B}$ is nonsingular in the X-block.*

Lemma 17 suggests a recursive algorithm: take an arbitrary $A \in \mathcal{B}$ and compute U^*, the limit of the first Wong sequence of $(A,\mathcal{B})$. If we get a singularity witness, we are done. Otherwise, if $U^* \neq 0$, as the X-block is already nonsingular, we only need to focus on the nonsingularity of Z-block which is of smaller size. To make this idea work, we have to satisfy essentially two conditions. We must find some A such that $U^* \neq 0$, and to allow for recursion the specific property of the matrix space $\mathcal{B}$ we are concerned with has to be inherited by the subspace corresponding to the Z-block. It turns out that in the triangularizable case these two problems can be taken care of by the following Lemma.

▶ **Lemma 18.** *Let $\mathcal{B} \leq \mathbb{F}$ be given by a basis $\{B_1, \ldots, B_m\}$, and suppose that there exist nonsingular matrices $C, D \in M(n,\mathbb{F}')$ such that $B_i = DB_i'C^{-1}$ and $B_i' \in M(n,\mathbb{F}')$ is upper triangular for every $i \in [m]$. Then we have the following.*

1. *Either $\cap_{i\in[m]} \ker(B_i) \neq 0$, or there exists $j \in [m]$ and $0 \neq U \leq \mathbb{F}^n$ s.t. $B_j(U) = \mathcal{B}(U)$.*
2. *Suppose there exist $j \in [m]$ and $0 \neq U \leq \mathbb{F}^n$ s.t. $B_j(U) = \mathcal{B}(U)$, and $\dim(U) = \dim(B_j(U))$. Let $B_i^* : \mathbb{F}^n/U \to \mathbb{F}^n/\mathcal{B}(U)$ be the linear map induced by B_i, for $i \in [m]$. Then $\mathcal{B}^* = \langle B_1^*, \ldots, B_m^* \rangle$ is triangularizable over $\mathbb{F}'$.*

Proof. 1. Let $\{e_i \mid i \in [n]\}$ be the standard basis of $\mathbb{F}'^n$, and $c_i = C(e_i)$ and $d_i = D(e_i)$ for $i \in [n]$. If $B_i'(1,1) = 0$ for all $i \in [m]$ then c_1 is in the kernel of every B_i's. If there exists j such that $B_j'(1,1) \neq 0$, we set $U' = \langle c_1 \rangle \leq \mathbb{F}'^n$. Then it is clear that $\langle d_1 \rangle = B_j(U') = \mathcal{B}(U')$. It follows that the first Wong sequence of $(B_j,\mathcal{B})$ over $\mathbb{F}'$ has nonzero limit, and therefore the same holds over $\mathbb{F}$. We can choose for U this limit.

2. First we recall that for a vector space V of dimension n, a complete flag of V is a nested sequence of subspaces $0 = V_0 \subset V_1 \subset \cdots \subset V_n = V$. For $\mathcal{A} \leq \mathrm{Lin}(V,V')$ with $\dim(V) = \dim(V') = n$, the matrix space $\mathcal{A}$ is triangularizable if and only if $\exists$ complete flags $0 = V_0 \subset V_1 \subset \cdots \subset V_n = V$ and $0 = V_0' \subset V_1' \subset \cdots \subset V_n' = V'$ s.t. $\mathcal{A}(V_i) \subseteq V_i'$ for $i \in [n]$.

For $U \leq \mathbb{F}^n$, let $\mathbb{F}'U$ be the linear span of U in $\mathbb{F}'^n$. We think of B_i's and B_i^*'s as linear maps over $\mathbb{F}'$ in a natural way. Let $\ell = \dim(\mathbb{F}'^n/\mathbb{F}'U)$. For $0 \leq i \leq n$ set $S_i = \langle c_1, \ldots, c_i \rangle$ and $T_i = \langle d_1, \ldots, d_i \rangle$. Obviously $\mathcal{B}(S_i) \subseteq T_i$ for $0 \leq i \leq n$. Let $S_i^* = S_i/\mathbb{F}'U$ and $T_i^* = T_i/\mathcal{B}(\mathbb{F}'U)$, and consider $S_0^* \subseteq \cdots \subseteq S_n^*$ and $T_0^* \subseteq \cdots \subseteq T_n^*$. We claim that $\forall i \in [n], \dim(S_i^*) \geq \dim(T_i^*)$. This is because as $T_i \cap \mathcal{B}(\mathbb{F}'U) \supseteq B_j(S_i \cap \mathbb{F}'U)$, by $\dim(\mathbb{F}'U) = \dim(B_j(\mathbb{F}'U))$, $\dim(B_j(S_i \cap \mathbb{F}'U)) \geq \dim(S_i \cap \mathbb{F}'U)$. Thus $\dim(S_i \cap \mathbb{F}'U) \leq \dim(T_i \cap \mathcal{B}(\mathbb{F}'U))$, and $\dim(S_i^*) \geq \dim(T_i^*)$. As $\mathcal{B}^*(S_i^*) \subseteq T_i^*$, $\dim(S_{i+1}^*) - \dim(S_i^*) \leq 1$, and $\dim(T_{i+1}^*) - \dim(T_i^*) \leq 1$, there exist two nested sequences $S_0^* \subset S_{j_1}^* \subset \cdots \subset S_{j_\ell}^* = S_n^*$ and $T_0^* \subset T_{k_1}^* \subset \cdots \subset T_{k_\ell}^* = T_n^*$, s.t. $\dim(S_{j_h}) = \dim(T_{k_h}) = h$. Furthermore, by $\dim(S_i^*) \geq \dim(T_i^*)$, $j_h \leq k_h$, thus

$\mathcal{B}^*(S^*_{j_h}) \subseteq \mathcal{B}^*(S^*_{k_h}) \subseteq T^*_{k_h}$, $\forall h \in [\ell]$. That is, the two nested sequences are complete flags, and $\mathcal{B}^*$ is triangularizable over $\mathbb{F}'$. ◀

Given the above preparation, we can now outline the algorithm for Theorem 2.

Proof of Theorem 2. First we consider finite fields. The algorithm recurses on the size of the matrices, with the base case being the size one. It checks at the beginning whether $\cap_{i\in[m]} \ker(B_i) = 0$. If this is the case then it returns $\cap_{i\in[m]} \ker(B_i)$ which is a singularity witness. Otherwise, for all $i \in [m]$, it computes the limit U_i^* of the first Wong sequence for $(B_i, \mathcal{B})$. By Lemma 18 (1) there exists $j \in [m]$ such that $U_j^* \neq 0$ and $B_j(U_j^*) = \mathcal{B}(U_j^*)$. The algorithm then recurses on the induced actions B_i^*'s of B_i's, which are also triangularizable by Lemma 18 (2). When $\mathcal{B}$ is nonsingular the algorithm should return a nonsingular matrix. This nonsingular matrix is built step by step by the recursive calls, at each step we have to construct a nonsingular linear combination of B_j and the matrix returned by the call. For this we need $n + 1$ field elements.

The case of the rational numbers can be reduced to the case of finite fields. Let b be a bound on the absolute values of entries in B_i's. It can be shown that there exists a prime number p of value polynomially bounded by $\log b$ and n s.t. the following holds: let B_i' be the matrix B_i modulo p. When $\mathcal{B}$ is triangularizable and nonsingular then the matrix space spanned by B_i' is triangularizable over an extension field of $\mathbb{F}_p$ and nonsingular. If $\mathcal{B}$ is singular, modulo any prime the matrix space is singular. So we enumerate all prime numbers up to the given polynomial bound, and for each prime use the algorithm over finite fields. ◀

Acknowledgements. We would like to thank the anonymous reviewers for careful reading and pointing out some gaps in an earlier version of the paper. Most of this work was conducted when G. I., Y. Q. and M. S. were at the Centre for Quantum Technologies (CQT) in Singapore, and partially funded by the Singapore Ministry of Education and the National Research Foundation, also through the Tier 3 Grant "Random numbers from quantum processes". Research partially supported by the European Commission IST STREP project Quantum Algorithms (QALGO) 600700, by the French ANR Blanc program under contract ANR-12-BS02-005 (RDAM project), by the Hungarian Scientific Research Fund (OTKA), Grants NK105645 and K77476, and by the Hausdorff grant EXC59-1/2.

References

1 M. D. Atkinson and N. M. Stephens. Spaces of matrices of bounded rank. *The Quarterly Journal of Mathematics*, 29(2):221–223, 1978.

2 T. Berger and S. Trenn. The quasi-Kronecker form for matrix pencils. *SIAM Journal on Matrix Analysis and Applications*, 33(2):336–368, 2012.

3 Thomas Berger and Stephan Trenn. Addition to "the quasi-Kronecker form for matrix pencils". *SIAM Journal on Matrix Analysis and Applications*, 34(1):94–101, 2013.

4 Stuart J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18(3):147–150, 1984.

5 Jonathan F. Buss, Gudmund S. Frandsen, and Jeffrey O. Shallit. The computational complexity of some problems of linear algebra. *J. Comput. Syst. Sci.*, 58(3):572–596, 1999.

6 Eric Chitambar, Runyao Duan, and Yaoyun Shi. Multipartite-to-bipartite entanglement transformations and polynomial identity testing. *Physical Reveiw A*, 81(5):052310, 2010.

7 Willem A. de Graaf, Gábor Ivanyos, and Lajos Rónyai. Computing Cartan subalgebras of Lie algebras. *Applicable Algebra in Engineering, Communication and Computing*, 7(5):339–349, 1996.

8 Jack Edmonds. Systems of distinct representatives and linear algebra. *J. Res. Nat. Bur. Standards Sect. B*, 71:241–245, 1967.

9 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In N. Sauer R. K. Guy, H. Hanani and J. Schönheim, editors, *Combinatorial Structures and their Appl.*, pages 69–87, New York, 1970. Gordon and Breach.

10 Michael A. Forbes and Amir Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. In *FOCS*, 2013.

11 James Geelen and Satoru Iwata. Matroid matching via mixed skew-symmetric matrices. *Combinatorica*, 25(2):187–215, 2005.

12 James F. Geelen. Maximum rank matrix completion. *Linear Algebra and its Applications*, 288:211–217, 1999.

13 James F. Geelen, Satoru Iwata, and Kazuo Murota. The linear delta-matroid parity problem. *Journal of Combinatorial Theory, Series B*, 88(2):377–398, 2003.

14 Leonid Gurvits. Quantum matching theory (with new complexity theoretic, combinatorial and topological insights on the nature of the quantum entanglement), 2002.

15 Leonid Gurvits. Classical complexity and quantum entanglement. *J. Comput. Syst. Sci.*, 69(3):448–484, 2004.

16 Nicholas J. A. Harvey, David R. Karger, and Kazuo Murota. Deterministic network coding by matrix completion. In *Proceedings of SODA*, pages 489–498. ACM-SIAM, 2005.

17 Nicholas J. A. Harvey, David R. Karger, and Sergey Yekhanin. The complexity of matrix completion. In *Proceedings of SODA*, pages 1103–1111. ACM-SIAM, 2006.

18 Gábor Ivanyos, Marek Karpinski, Youming Qiao, and Miklos Santha. Generalized wong sequences and their applications to edmonds' problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:103, 2013.

19 Gábor Ivanyos, Marek Karpinski, and Nitin Saxena. Deterministic polynomial time algorithms for matrix completion problems. *SIAM J. Comput.*, 39(8):3736–3751, 2010.

20 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

21 László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574, 1979.

22 László Lovász. Singular spaces of matrices and their application in combinatorics. *Boletim da Sociedade Brasileira de Matemática-Bulletin/Brazilian Mathematical Society*, 20(1):87–99, 1989.

23 Kazuo Murota. *Matrices and matroids for systems analysis*. Springer, 2000.

24 Richard Rado. A theorem on independence relations. *The Quarterly Journal of Mathematics, Oxford Ser.*, 13(1):83–89, 1942.

25 Jacob T. Schwartz. Probabilistic algorithms for verification of polynomial identities. In Edward W. Ng, editor, *Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 200–215. Springer Berlin Heidelberg, 1979.

26 Stephan Trenn. Solution concepts for linear DAEs: A survey. In Achim Ilchmann and Timo Reis, editors, *Surveys in Differential-Algebraic Equations I*, Differential-Algebraic Equations Forum, pages 137–172. Springer Berlin Heidelberg, 2013.

27 Leslie G. Valiant. Completeness classes in algebra. In *STOC*, pages 249–261, 1979.

28 D. J. A. Welsh. On matroid theorems of Edmonds and Rado. *Journal of the London Mathematical Society*, 2(2):251–256, 1970.

29 Kai-Tak Wong. The eigenvalue problem $\lambda Tx + Sx$. *Journal of Differential Equations*, 16(2):270 – 280, 1974.

30 Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation*, volume 72 of *LNCS*, pages 216–226. Springer, 1979.

Read-Once Branching Programs for Tree Evaluation Problems

Kazuo Iwama[1] **and Atsuki Nagao**[2]

1 **Graduate School of Informatics, Kyoto University, Kyoto, Japan**
iwama@kuis.kyoto-u.ac.jp
2 **Graduate School of Informatics, Kyoto University, Kyoto, Japan**
a-nagao@kuis.kyoto-u.ac.jp

Abstract

Toward the ultimate goal of separating **L** and **P**, Cook, McKenzie, Wehr, Braverman and Santhanam introduced the *tree evaluation problem* (*TEP*). For fixed $h, k > 0$, $FT_h(k)$ is given as a complete, rooted binary tree of height h, in which each internal node is associated with a function from $[k]^2$ to $[k]$, and each leaf node with a number in $[k]$. The value of an internal node v is defined naturally, i.e., if it has a function f and the values of its two child nodes are a and b, then the value of v is $f(a, b)$. Our task is to compute the value of the root node by sequentially executing this function evaluation in a bottom-up fashion. The problem is obviously in **P** and if we could prove that any branching program solving $FT_h(k)$ needs at least $k^{r(h)}$ states for any unbounded function r, then this problem is not in **L**, thus achieving our goal. The above authors introduced a restriction called *thrifty* against the structure of BP's (i,e., against the algorithm for solving the problem) and proved that any thrifty BP needs $\Omega(k^h)$ states. This paper proves a similar lower bound for *read-once* branching programs, which allows us to get rid of the restriction on the order of nodes read by the BP that is the nature of the thrifty restriction.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Lower bounds, Branching Programs, Read-Once Branching Programs, Space Complexity, Combinatorics

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.409

1 Introduction

Settling the **P** vs. **NP** question is obviously the biggest goal of theoretical computer science, but the fact is that almost nothing is known for separation of other complexity classes, either. For example, separation of **L** (= Log space) and **P**, which has been much less popular than **P** vs. **NP**, should be equally important to make clear the whole view of complexity classes. To this end, Cook, McKenzie, Wehr, Braverman and Santhanam introduced a simple but very general problem called the *tree evaluation problem* (*TEP*) [3]. For fixed $h, k > 0$, $FT_h(k)$ is given as a complete, rooted binary tree of height h in which each internal node is associated with a function from $[k]^2$ to $[k]$, and each leaf node with a number in $[k]$. The value of an internal node v is defined naturally, i.e., if it has a function f and the values of its two child nodes are a and b, then the value of v is $f(a, b)$. Our task is to compute the value of the root node by sequentially executing this function evaluation in a bottom-up fashion. Note that the original definition in [3] is based on a d-ary tree. In this paper, we only consider a binary tree for our TEP.

Our computation model is branching programs (BP's) that are sometimes more useful to discuss complexity bounds rather than Turing machines (TM's) especially for problems


© Kazuo Iwama and Atsuki Nagao;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 409–420

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


having relatively low complexities like TEP. It is known that the size of a branching program (the number of its states) and the space of a TM are closely related, namely a lower bound $s(n)$ for BP's size implies a lower bound $\Theta(\log(s(n)))$ for TM's space. It then turns out that if we can prove that any BP solving $FT_h(k)$ needs at least $k^{r(h)}$ states for any unbounded function r, then this problem is not in $\mathbf{L}$. Since it is obviously in $\mathbf{P}$, we would be able to separate $\mathbf{L}$ and $\mathbf{P}$. For details of these observations, see [3].

It is not hard to construct a branching program that computes $FT_h(k)$ of size $O(k^h)$ (see Fig. 3 given later) and this construction strongly seems optimal. As mentioned above, we only need a much more moderate bound, $k^{r(h)}$, and that is the natural reason why we think this problem would fit our goal. In fact, [3] proves, by using the black pebbling game [10][2], that if our BP's satisfy a certain property, called the *thrifty* restriction, then we do need $\Omega(k^h)$ states. The thrifty restriction roughly means that when the BP reads an internal node v (actually reads its associated function), it has already read all the values of the v's subtree. Thus this algorithmic restriction strongly restricts the order of tree nodes that are read by the BP. (The thrifty restriction also applies to nondeterministic BP's, in which case its meaning is more subtle.) The authors claim that this restriction is "natural," but we can of course think of different kind of BP's that guess (read) function values first and then check the leaf values if they actually realize the function values. In fact our lower bound proof gets messy in this case.

Recall that we have another popular restriction type of BP's, namely the *read-once* restriction, where a read-once BP reads each input value at most once in any computation path. In fact the above $O(k^h)$ construction is not only thrifty but also read-once and [14] proves that if our BP is both thrifty and read-once, then this explicit construction with $(k+1)^h - k$ states is absolutely optimum. Now the natural question is what if we impose only the read-once restriction.

Our contribution. It is shown that if a read-once BP B solves $FT_h(k)$, then B needs $\Omega(k^h)$ states, thus proving a lower bound on the size of read-once BP's similar to that of thrifty BP's. Actually B needs to be read-once only for states reading leaf values, i.e., the result holds for even less restricted BP's such that in every computation path, if the last leaf-reading state s reads a leaf node v, any state appearing before s on the path does not read v. Note that there is no restriction at all on states reading internal nodes (associated with functions). Furthermore, since our main lemma bounds the number of only leaf-reading states, we do not have to care about the number of these non-leaf-reading states.

Since there are no restrictions on the order of nodes visited by the BP any longer, there is no obvious way of directly using the pebbling game for lower bound proof. Instead, we use a similar notion from a slightly different angle, namely we use what we call a *cut configuration*, a set of the values of $h-1$ nodes that "cut" paths between leaf nodes and the root of the given $FT_h(k)$. The key lemma (Lemma 5) is that if a last leaf-reading state accepts two or more inputs having different cut configurations, then the function part in the inputs is severely restricted, which means the number of different inputs whose paths go through this state is very small. Thus there must be a lot of inputs whose function part does not have this restriction, and we can imply that those inputs have only one cut configuration for any of the last leaf-reading states. For such a fixed function part, the number of inputs having that cut configuration for the last leaf-reading state is easily bounded from above. Thus follows the lower bound for the number of such states. Of course there should still exist a big gap between this class of BP's and general ones, but at least we can get rid of the issue of node orders visited by BP's, which was quite annoying for the attempt of generalising our lower bound proofs.

Our proof depends on another important lemma (Lemma 3) that relates the number of leaf-reading states and the number of states reading second-leaves (the leaves located at height $h-1$). This lemma holds for general BP's and gives us a by-product. Namely, as shown in Section 4, we can obtain a k^3 lower bound for general BP's for the height-3 TEP, which is the same as [3] but with a simpler proof.

Related Work. Other than the lower bounds for thrifty BP's, [3] includes several important results, for instance, it gives a lower bound, k^3, for *unrestricted* BP's solving $FT_3(k)$, which is tight up to the constant factor. This is still the best lower bound for general BP's solving TEP. [7] studies mainly nondeterministic BP's for TEP. Its main result is that "bitwise-independent" thrifty nondeterministic BP's for TEP have at least $\frac{1}{2}k^{h/2}$ states, which is tight against the upper bounds shown in [3]. Their main technique is so-called the entropy method developed in [6]. See [3] for several other attempts trying to separate relatively low complexity classes. For instance [4] studies the complexity of BP's solving GEN (known to be P-complete) that asks a certain kind of reachability to a target element repeatedly using a binary operation.

Studies on branching programs have been quite popular since their introduction by Masek [8], and there is a large literature if it is restricted to studies on their size lower bounds (the following is only a small fraction): The best general deterministic lower bound is still $\Omega(n^2/(\log n)^2)$, which was proved almost half a century ago by Nečiporuk [9]. Note that the above lower bound for $FT_3(k)$ is $\Omega(n^{3/2}/(\log n)^{5/2})$ in terms of the binary input length. (For a general d-ary TEP, [3] obtains a stronger $\Omega(n^2/(\log(n))^2)$ lower bound applying the Nečiporuk method.) Against read-once branching programs, we have much better lower bounds. In 1984, Žák [15] first obtained a super-polynomial lower bound, $\Omega(2^{\sqrt{n}-\log n})$, for the half-clique function, which was improved to more than $2^{n/3-o(n)}$ by Wegener [13]. For the triangle parity function, Ajtai [1] gave a 2^{cn} lower bound and the value of c was later improved by Simon(1993) [12]. Jukna [5] relaxed the read-once restriction to the k-read-once restriction (i.e., all variables except k ones are read-once). He obtained a lower bound of $2^{\Omega((\frac{n}{k})^{1/2})}$ for $k = O(n/\log n)$ and this is extended by Žak [11] into a hierarchy theorem based on this value k.

2 Preliminaries

For the Tree Evaluation Problem (TEP), $FT_h(k)$, we are given a complete binary tree T^h of height h with nodes 1 through 2^h-1 (see Fig. 1 for $h=3$). Each internal node $1 \le i \le 2^{h-1}-1$ is associated with some explicit function $f_i : [k]^2 \mapsto [k]$, where $[k] = \{1, 2, \ldots, k\}$. Each leaf node j ($2^{h-1} \le j \le 2^h - 1$) is associated with a number in $[k]$. Our task is to compute the value of the function f_1 at the root node in the natural way: Suppose that we have inputs $f_1, f_2, f_3, a_4, a_5, a_6, a_7$ for the tree of Fig. 1. Then the value we want to obtain is

$$f_1(f_2(a_4, a_5), f_3(a_6, a_7)).$$

Note that each f_i is given as an explicit sequence of values, e.g., $f_i(1,1)$, $f_i(1,2)$, $f_i(1,3)$, $f_i(2,1)$, $f_i(2,2)$, $f_i(2,3)$, $f_i(3,1)$, $f_i(3,2)$, $f_i(3,3)$ for $k=3$. In some cases, it is convenient to use a $k \times k$ matrix instead of the above sequence. For instance Fig. 2 shows an example of f_1,f_2,f_3 for $h=3$. Now if (a_4, a_5, a_6, a_7)=$(3,3,1,2)$, then the solution for this inputs is $f_1(f_2(3,3), f_3(1,2))$= $f_1(3,2)$= 1. In our lower bound proof, the nodes located at height $h-1$ (parents of leaves) play an important role. We call them *second-leaves*.

Our computation model is a (deterministic) *branching program* (*BP*) B, which is a directed, rooted, acyclic graph. Its vertices are called *states* including a unique *initial state* and k *sink*

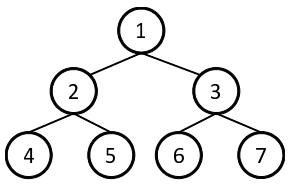


Figure 1 $FT_3(3)$.

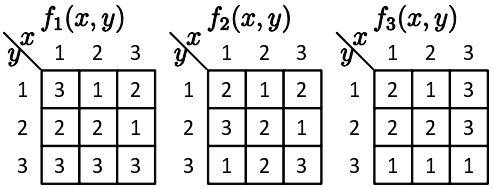


Figure 2 Example of f_1, f_2, f_3.

states. Each non-sink state (or simply a state if no confusion would arise) has k outgoing edges labelled by 1 through k, while each sink state has no outgoing edges. Each state has a *label* of the form (i_1, i_2, i_3) or j, where $1 \leq i_1 \leq 2^{h-1}-1$, $1 \leq i_2, i_3 \leq k$ and $2^{h-1} \leq j \leq 2^h - 1$. Each sink state has a label l where $1 \leq l \leq k$. A BP B computes the solution of TEP in the following way. Suppose that our input is $I = (f_1(1,1), f_1(1,2), \ldots, f_{2^{h-1}-1}(k,k), a_{2^{h-1}}, \ldots, a_{2^h-1})$. Then its *computation path*, P, for input I is defined as follows. P starts from the initial state. If P is now at a state with label (i_1, i_2, i_3), then P is extended by the edge labelled by $f_{i_i}(i_2, i_3)$. If P is at a state with label j, then it is extended by the edge labelled by a_j. We often say that B "reads" the input attached to the node i_1 (a non-leaf node) or j (a leaf) and branches due to its value between 1 and k. P ends with some sink state; if its label is l, then the outcome of the computation is l. If this outcome is equal to the correct solution for all possible inputs I, then we say B *solves* $FT_h(k)$.

Fig. 3 shows an example of a BP that solves $FT_3(3)$. The computation path for the input previously given (f_1, f_2, f_3 in Fig. 2, and $(a_4, a_5, a_6, a_7) = (3, 3, 1, 2)$) is given by a thick line. A BP is called *read-once* if all paths from the root to sinks do not have two or more same labels. The BP in Fig. 3 is read-once.

Our lower bound proof is based on the following simple idea: Suppose that A ($|A| = m_1$) is a carefully selected subset of all the possible inputs for $FT_h(k)$. Let B be any (read-once) BP that solves $FT_h(k)$. Then our proof says that we can always select a set S of states such that each computation path corresponding to each input in A goes through some state in S and any state in S accepts computation paths of at most m_2 inputs in A, concluding that $|S|$ is at least m_1/m_2. To introduce such an input set A, we consider the following constraint for functions f_i: Suppose that

$$X = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1k} \\ \alpha_{21} & \ddots & \cdots & \alpha_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{k1} & \alpha_{12} & \cdots & \alpha_{kk} \end{bmatrix}$$

is the matrix representation of f_i. Then it has to satisfy the following three constraints: (i) $\alpha_{11} \ldots \alpha_{1k}$ (= the first row) is a permutation of $(1, \ldots, k)$ (ii) $\alpha_{11} \ldots \alpha_{k1}$ (= the first column) is a permutation of $(1, \ldots, k)$ (iii) For $\forall j \geq 2$, $\alpha_{j1} \ldots \alpha_{jk}$ is a permutation that can be written as $\delta^l(\alpha_{11} \ldots \alpha_{1k})$ for some $1 \leq l \leq k$ where δ is the cyclic permutation

$$\delta = \begin{pmatrix} 1 & 2 & \cdots & k-1 & k \\ 2 & 3 & \cdots & k & 1 \end{pmatrix}$$

and δ^l is a composition of l δ's. Thus each row is a permutation, and it is not hard to see that each column is also a permutation. X is fixed by determining its first row and the first column, and hence there are $k!(k-1)!$ different f_i's. Let F be the class of functions satisfying these constraints. In this paper, we assume that our function f_i is always selected

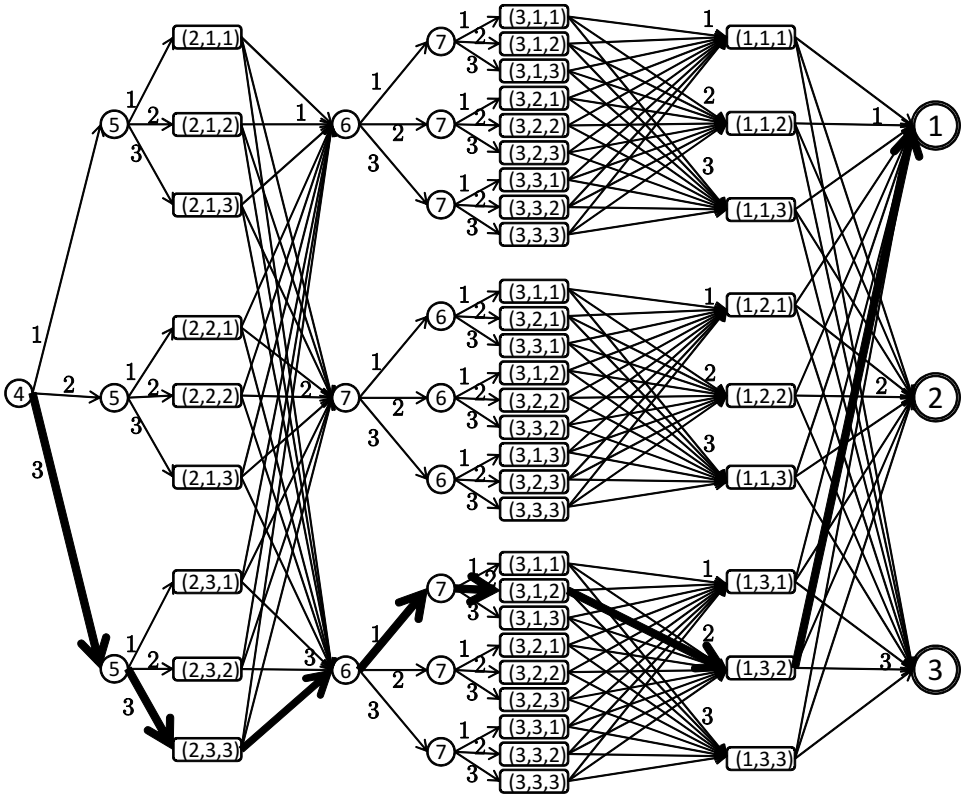


Figure 3 An example of a read-once branching program solving $FT_3(3)$.

from F unless otherwise stated (but of course, our BP's must give correct solutions for all inputs). Now here are easy but important lemmas.

▶ **Lemma 1.** *Suppose that two inputs I and I' (their function parts satisfy the constraint) are exactly the same except only one leaf value at node j. Then the final value of $FT_h(k)$ is different between I and I'.*

Proof. Suppose that the final value is the same and consider the path from the root to j. Since the root value is the same and the leaf value is different, there must be a node i on the path such that the value of i is the same but the value of i's next node i' on the path is different, say, a in I and a' in I'. Let i'' be the sibling of i' (both i' and i'' have i as their parent). Then the value of i'' is the same, say b, in both I and I'. Thus we have $f_i(a, b) = f_i(a', b)$ for $a \neq a'$, which contradicts that $f_i \in F$. ◀

▶ **Lemma 2.** *Suppose that a BP B solves $FT_h(k)$. Then (1) for any internal node i of $FT_h(k)$ and for any $a, b \in [k]$, there must be a state whose label is (i, a, b) in B. (2) If P is a legal computation path, then for any leaf node j, P includes a state that reads j.*

Proof. For (2), suppose that P corresponds to input I and it does not read j. Then consider another input I' which is different from I only in j. Then B obviously outputs the same value for I and I', contradicting the previous lemma. (1) is proved similarly by considering two inputs I and I' that differ only in $f_i(a, b)$ and such that both inputs actually use $f_i(a, b)$ (meaning the values of i's two children are a and b under I and I'). Note that if I satisfies

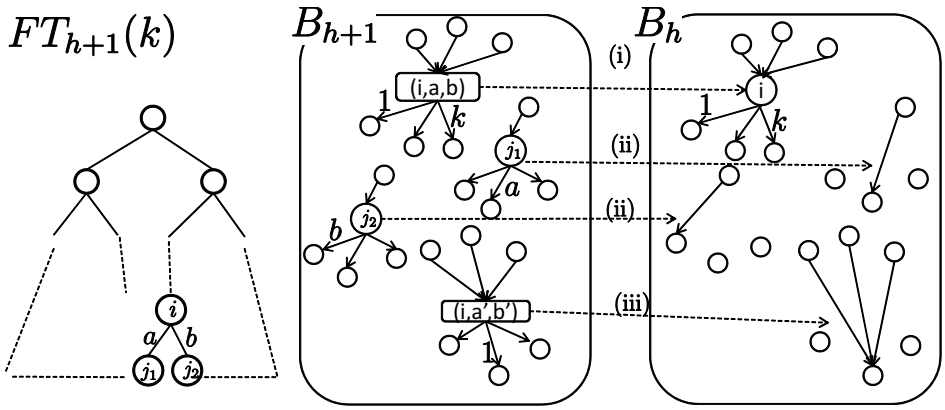


Figure 4 Modification rules(i), (ii) and (iii).

the restriction, then I' does not. Now one can see, exactly as in the proof of the previous lemma, that the final value is different between I and I', but B outputs the same value, a contradiction. ◀

The next lemma (hinted by Th. 5.8 and Th. 5.9 of [3]) relates the number of states reading leaf nodes and the number of state reading second-leaf nodes. By this lemma, we can increase the degree of k by one in the lower bound given in the next section. Note that this lemma holds for general BP's (and see Sec. 4 for its by-product).

▶ **Lemma 3.** *For $h \geq 1$, if there is a BP B_{h+1} solving $FT_{h+1}(k)$ such that the number of states that read second-leaf nodes is n, then there is a BP B_h solving $FT_h(k)$ such that the number of states that read leaf nodes is at most n/k^2. Furthermore, if B_{h+1} is read-once, so is B_h, also.*

Proof. we construct B_h from the given B_{h+1} as follows. Let i be a second-leaf node of $FT_{h+1}(k)$ and (a, b) is a pair of inputs to f_i such that the number of states in B_{h+1} that read $f_i(a, b)$ is less than or equal to the number of states reading $f_i(a', b')$ for any (a', b'). Let m be the number of such state s reading $f_i(a, b)$. By Lemma 2, there is at least one state that reads $f_i(a, b)$ for any $(a, b) \in [k] \times [k]$. So, m is at most $(1/k^2)\times$(the number of states that read f_i). Now we make the following modification against B_{h+1} (see Fig. 4). The basic idea is that we fix the values of the two child (leaf) nodes of i to a and b. Then i looks like a leaf node of $FT_h(k)$ and among the states in B_{h+1} that read i, only $1/k^2$ ones survive by the following construction. This holds for any i and hence the lemma holds. (i) Change the label of the above m states from (i, a, b) to i. (Namely this state reads a leaf node of $FT_h(k)$.) (ii) Suppose that j_1 and j_2 are the two leaf nodes whose parent is i. Then we remove all the states q of B_{h+1} that read j_1 (j_2, respectively) by connecting q's incoming edges to the state to which the edge from q labelled by a (b, respectively) goes. (iii) We remove all the state q of B_{h+1} that read $f_i(a', b')$, $((a', b') \neq (a, b))$, by connecting q's incoming edges to the state to which the edge from q labelled by 1 goes (this "1" is not important or it may be any number in $[k]$).

We repeat this change for all second-leaf nodes of $FT_{h+1}(k)$, obtaining B_h. We omit the proof that this construction is correct, since it is almost obvious from the construction. ◀

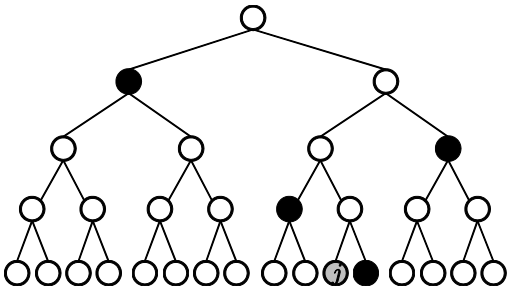


Figure 5 Involved nodes in $CC(I, j)$.

3 Lower Bounds

In this section we obtain a lower bound for the number of states that read leaf nodes of $FT_h(k)$. Then combining it with Lemma 3, we obtain a better lower bound for the number of states that read second-leaf nodes of $FT_h(k)$. Recall that our input satisfies the constraint (its functions belong to F) and all BPs in this section are read-once. Let B be a BP that solves $FT_h(k)$ and P be its arbitrary computation path. (To avoid confusion, we sometimes say that P is a *legal* computation path to emphasise that P is based on an input whose function part satisfies the constraint.) Then by Lemma 2, P reads all leaf values (for any leaf j, there is a state in P that reads j). Let q be the last state on P that reads a leaf value, i.e., there is no state after q on P that reads a leaf. Since B is read-once, q is also the last leaf-reading state on any other legal computation path that includes q. Thus, as far as we are looking at only legal computation paths, we can define a *last leaf-reading state* without specifying a computation path.

Now we define our key tool in the proof in this section. Suppose that $I = (f_1, \ldots, f_{2^{h-1}-1}, a_{2^{h-1}}, \ldots, a_{2^h-1})$ is currently associated with $FT_h(k)$ and let j be a leaf. Then the *cut configuration* (CC) for I with respect j, denoted as $CC(I, j)$, is defined as follows.

$$CC(I, j) = (a_1, a_2, \ldots, a_{h-1})$$

where, (i) a_1 is the value of j's sibling and (ii) if a_i, $1 \leq i \leq h-2$, is the value of node x, a_{i+1} is the value of the sibling of x's parent (see Fig. 5). Suppose that we know functions f_1 to $f_{2^{h-1}-1}$. Then if we further know these $h-1$ values as well as the value of j, then we can compute the solution (= the value of node 1). In fact, it is well-known that we can compute the solution in such a way that we need at most $(h-1)\lceil \log k \rceil$ memory space at any stage of its computation (by recursively obtaining the values of a_1, a_2, and so on, in this order first, then the associated function values from bottom to top). What will be done in the rest of this section is to count the number of legal inputs with a certain restriction on its CC that go through a last leaf-reading node. Our first lemma is an upper bound on the number of inputs having a single CC.

▶ **Lemma 4.** *Fix functions $f_1, \ldots, f_{2^{h-1}-1}$, an arbitrary leaf node, j, and an arbitrary $(a_1, \ldots, a_{h-1})$, $a_i \in [k]$. Then the number of leaf values whose CC with respect to j is $(a_1, \ldots, a_{h-1})$ is at most $k^{2^{h-1}-h+1}$*

Proof. Let T_v be a subtree of $FT_h(k)$ whose root is a node v at height i of $FT_h(k)$. Let $v_1, \ldots, v_{2^{i-1}}$ (we used a simplified numbering) be the leaf nodes of T_v, and $g(v_1, \ldots, v_{2^{i-1}})$ be the value of v. We first calculate the number of different leaf values $(b_1, \ldots, b_{2^{i-1}})$ such

that $g(b_1, \ldots, b_{2^{i-1}}) = a$ for a fixed $a \in [k]$. For fixed $b_1, \ldots, b_{2^{i-1}-1}$, $g(b_1, \ldots, b_{2^{i-1}-1}, x)$ is a function from $[k]$ to $[k]$ (denoted by $g'(x)$). By lemma 1, $g'(x_1) \neq g'(x_2)$ if $x_1 \neq x_2$, in other words, g' is a bijection. Hence, for any $a \in [k]$, value $b \in [k]$ such that $g'(b) = a$ is fixed. Since this holds for any $b_1, \ldots, b_{2^{i-1}-1}$, the number of leaf values $b_1, \ldots, b_{2^{i-1}}$ such that $g(b_1, \ldots, b_{2^{i-1}}) = a$ is $k^{2^{i-1}-1}$.

Now we calculate the number N of leaf values such that their CC with respect to leaf j is $(a_1, \ldots, a_{h-1})$. Let the node taking value a_i be v_i. See Fig. 5 again. First, note that node j can take any of the k values. Next, the value of node v_1 is fixed to a_1; it takes only one value. Since node v_2 is a top node of a subtree with height 2, its leaf nodes can take $k^{2^{2-1}-1} = k$ different values by the above fact. Similarly, v_3's leaf nodes can take $k^{2^{3-1}-1} = k^3$ different values and so on. Therefore,

$$\begin{aligned} N &= k \cdot 1 \cdot k \cdot k^3 \cdot \cdots \cdot k^{2^{h-2}-1} \\ &= k \cdot k^{2^1+2^2+\cdots+2^{h-2}-(h-2)} \\ &= k \cdot k^{2^{h-1}-2-(h-2)} = k^{2^{h-1}-(h-1)} \end{aligned}$$

◀

Now we are ready to prove our main lemma. We divide an input $(f_1, \ldots, f_{2^{h-1}-1}, a_1, \ldots, a_{2^{h-1}})$ into two parts, the function part (f-part) $\mathbf{f} = (f_1, \ldots, f_{2^{h-1}-1})$ and the leaf value part (l-part) $\mathbf{l} = (a_1, \ldots, a_{2^{h-1}})$. Let B be any (read-once) BP solving $FT_h(k)$ and s be any last leaf-reading state, reading a leaf j. Let c_1 and c_2 be two different CC's with respect to j whose inputs have the same f-part, and $G(c_1, c_2, s)$ be the set of such f-parts (i.e., if $\mathbf{f} \in G(c_1, c_2, s)$, then there are two l-parts $\mathbf{a}_1$ and $\mathbf{a}_2$ such that $(\mathbf{f}, \mathbf{a}_1)$ and $(\mathbf{f}, \mathbf{a}_2)$ have CC's c_1 and c_2, respectively). Note that there are $(k!(k-1)!)^{2^{h-1}-1}$ different f-parts in total and we denote this number by N_0.

▶ **Lemma 5.** *Suppose that k is a prime number. Then for any c_1, c_2, s, $|G(c_1, c_2, s)| \leq N_0 \frac{k}{k!}$*

Proof. Let $c_1 = (a_1, \ldots, a_{h-1})$, $c_2 = (b_1, \ldots, b_{h-1})$, and let $v_1, \ldots, v_{h-1}$ be the nodes providing these two CC values. Also let $g_1, \ldots g_{h-1}$ be the functions associated with $v_1, \ldots, v_{h-1}$ producing both c_1 and c_2 (for different leaf values). Recall that s is a last leaf-reading state (reading node j) and our BP is read-once. Hence, the value of j is first read at s, which means two computation paths realizing c_1 and c_2 are not affected by the value of j until the state s. Furthermore, these paths must go to the same sink-node for each fixed value a of the node j, because after the state s our BP reads only function values being the same for c_1 and c_2.

$$g_1(a_1, g_2(a_2, \ldots, g_{h-1}(a_{h-1}, a) \ldots)) = g_1(b_1, g_2(b_2, \ldots, g_{h-1}(b_{h-1}, a) \ldots)) \tag{1}$$

Note that for a fixed a_{h-1}, $g_{h-1}(a_{h-1}, a)$ is a bijection form $[k]$ to $[k]$ and can be represented as a permutation

$$\delta_{h-1} = \begin{pmatrix} 1 & 2 & \cdots & k \\ \alpha_1 & \alpha_2 & \cdots & \alpha_k \end{pmatrix}$$

where $\alpha_1 \ldots \alpha_k$ are the (a_{h-1})th row of the matrix of g_{h-1}. Using similar representations for g_1 to g_{h-2}, (1) can be written as

$$\delta_1 \delta_2 \ldots \delta_{h-1} = \delta'_1 \delta'_2 \ldots \delta'_{h-1} \tag{2}$$

where δ_i is the (a_i)th row of (the matrix of) g_i and δ'_i is the (b_i)th row of g_i. Due to our constraint for g_i, we can write $\delta'_i = \delta^{l_i}\delta_i$ for some $0 \le l_i \le k-1$ (recall that δ is the cyclic permutation).

Now (2) can be rewritten as

$$\delta_1\delta_2\dots\delta_{h-1} = \delta^{l_1}\delta_1\delta^{l_2}\delta_2\dots\delta^{l_{h-1}}\delta_{h-1}$$

Suppose that a_i and b_i are the first different values in the two CC's (i,e., $a_1 = b_1, \dots a_{i-1} = b_{i-1}$). Then $l_1 = l_2 = \dots = l_{i-1} = 0$ and hence

$$\begin{aligned}
&\delta_1\delta_2\dots\delta_{h-2}\delta_{h-1} = \delta^{l_1}\delta_1\delta^{l_2}\delta_2\dots\delta_{h-2}\delta^{l_{h-1}}\delta_{h-1}\\
\Leftrightarrow\quad &\delta_i\delta_{i+1}\dots\delta_{h-2} = \delta^{l_i}\delta_i\delta^{l_{i+1}}\delta_{i+1}\dots\delta^{l_{h-1}}\\
\Leftrightarrow\quad &\delta_i = \delta^{l_i}\delta_i\delta^{l_{i+1}}\delta_{i+1}\dots\delta^{l_{h-1}}\delta_{h-2}^{-1}\dots\delta_{i+1}^{-1}\\
\Leftrightarrow\quad &\delta^* = \delta_i^{-1}\delta^c\delta_i \qquad (3)
\end{aligned}$$

where $\delta^* = (\delta^{l_{i+1}}\dots\delta_{i+1}^{-1})^{-1}$ and $\delta^c = \delta^{l_i}$.

Now let

$$\delta_i = \begin{pmatrix}\alpha_1 & \alpha_2 & \cdots & \alpha_k\\ 1 & 2 & \cdots & k\end{pmatrix} \text{ and } \delta^* = \begin{pmatrix}1 & 2 & \cdots & k\\ \beta_1 & \beta_2 & \cdots & \beta_k\end{pmatrix},$$

Then (3) can be written as

$$\begin{aligned}
&\begin{pmatrix}1 & 2 & \cdots & k\\ \beta_1 & \beta_2 & \cdots & \beta_k\end{pmatrix}\\
&= \begin{pmatrix}1 & 2 & \cdots & k\\ \alpha_1 & \alpha_2 & \cdots & \alpha_k\end{pmatrix}\begin{pmatrix}1 & 2 & \cdots & k\\ 1+c & 2+c & \cdots & k+c\end{pmatrix}\begin{pmatrix}\alpha_1 & \alpha_2 & \cdots & \alpha_k\\ 1 & 2 & \cdots & k\end{pmatrix}\\
&= \begin{pmatrix}\alpha_1 & \alpha_2 & \cdots & \alpha_k\\ \alpha_{1+c} & \alpha_{2+c} & \cdots & \alpha_{k+c}\end{pmatrix}
\end{aligned}$$

where $1+c, \dots, k+c$ are all MOD k. Note that δ^* and δ^c are conjugate and therefore their cycle structures are the same. Since δ is the cyclic permutation, δ^c has a single cycle and therefore δ^* also has a single cycle.

It then turns out that if we fix α_1 to $d \in [k]$, then by the left hand side, d should be mapped to β_d, meaning $\alpha_{1+c} = \beta_d$. Then again by the left hand side, β_d should be mapped to β_{β_d}, meaning $\alpha_{1+2c} = \beta_{\beta_d}$, and so on. Namely once α_1 is fixed, all the other α_i's are sequentially fixed one after another or δ_i itself is fixed. Since k is prime, this sequence of value transfer does not end in the middle. Thus we have at most k different possibilities for δ_i (due to k different values for α_1). Recall that δ_i can take $k!$ different permutation in general. (The whole matrix is determined by fixing the first row and the first column, but it should be noted that it is also determined by fixing any row and then any column). But now there are only k possibilities as shown above. So the number of different g_i is at most $N_0\frac{k}{k!}$ for each combination of other $h-2$ functions. Note that s may have another CC, say c_3, other than c_1 and c_2. Then we have another restriction for functions, which results in even a smaller number of possible functions. Thus it is enough to consider only the case that the state s has two different CC's, for the upper bound of the lemma. ◀

Now we imply a contradiction if the number of leaf-reading states is less than k^{h-1}, through the following two lemmas.

▶ **Lemma 6.** *Suppose that $s_1, s_2, \ldots, s_{k^{h-1}-1}$ are $k^{h-1}-1$ different last leaf-reading states of the BP B. Then there is an f-part $\mathbf{f}_0$ such that for any s_i $(1 \le i \le k^{h-1}-1)$, if inputs $(\mathbf{f}_0, \mathbf{a}_1)$ and $(\mathbf{f}_0, \mathbf{a}_2)$ go through s_i, their CC's are the same.*

Proof. We count the number of f-parts $\mathbf{f}$ that do not satisfy the condition of the lemma. By Lemma 5, there are $N_0 \cdot \frac{k}{k!}$ such $\mathbf{f}$ for each combination of state s_i, CC c_1 and CC c_2. Note that we have $k^{h-1}-1$ s_i's and there are at most k^{h-1} different CC's in general. Therefore the number of such $\mathbf{f}$'s is at most

$$N_0 \cdot \frac{k}{k!} \cdot (k^{h-1}-1) \cdot (k^{h-1})^2 \le N_0 k^{3h-2}/k!,$$

which is strictly less than N_0 for a large (prime) k. Thus an $\mathbf{f}_0$ of the lemma must exists. ◀

▶ **Lemma 7.** *B needs at least k^{h-1} last leaf-reading states.*

Proof. Suppose B has at most $k^{h-1}-1$ last leaf-reading states. Then by Lemma 6, there is an f-part $\mathbf{f}_0$ such that inputs having this $\mathbf{f}_0$ as their f-part show at most one CC for any of these last leaf-reading states. However, Lemma 4 shows each state accepts at most $k^{2^{h-1}-h+1}$ l-parts, meaning these $k^{h-1}-1$ states accept at most $k^{2^{h-1}-h+1} \cdot (k^{h-1}-1) < k^{2^{h-1}}$ l-values in total. Since each of the all $k^{2^{h-1}}$ l-values must be accepted by some last leaf-reading state, this is a contradiction. ◀

▶ **Theorem 8.** *Any read-once BP B_h solving $FT_h(k)$ needs at least k^h states.*

Proof. The contraposition of Lemma 4 claims that if the number of leaf-reading states of B_h is at least m, then the number of second-leaf-reading states of B_{h+1} is at least $k^2 m$. Now the theorem is immediate from Lemma 7. ◀

4 General Branching Programs for Height-3 TEP

Recall that Lemma 3 holds for general BPs. Also it turns out that the TEP of height two is somewhat special. Thus we can obtain the following general lower bound for BPs for the height-3 TEP with a simpler proof than that of [3].

▶ **Theorem 9.** *Any (general) BP solving $FT_3(k)$ needs at least k^3 states.*

Proof. Due to Lemma 3, it suffices to show that any BP solving $FT_2(k)$ needs at least k leaf-reading states. In the following, we show it needs at least $k+1$ leaf-reading states, which is optimal by a construction similar to that of Fig. 3. Recall that $FT_2(k)$ has three nodes, 1, 2 and 3, where node 1 is associated with a function f_1 and nodes 2 and 3 are leaf nodes. Suppose that we have a BP B that solves $FT_2(k)$ and that has at most k leaf-reading states. We fix f_1 to an arbitrary function in F and then B can be modified to the BP that reads only leaf nodes; we also denote this BP by B.

We give a new label (in addition to the original label of B), a set of pairs (a,b), $1 \le a, b \le k$, to each state and each edge of B by the following rule: (i) The initial node of B has label $\{(a,b) \mid 1 \le a, b \le k\}$, i.e., the set of all possible pairs. (ii) Suppose that a state s has a label S and an edge e from s reads node 2 to get value i. Then the label to the edge e is $\{(i,b) \mid 1 \le b \le k\} \cap S$, namely the (possibly empty) set of pairs in S whose first element is i. Similarly for the case that s reads node 3 (we do the same thing with the second element of the pair). (iii) Suppose that all the edges entering state s already have labels. Then the label of s is the union of the labels of those incoming edges. Now it is easy to see that such labels "describe" an execution of B in the following sense: Suppose that the label of an edge

e includes a pair (a, b). Then the computation path of B goes through this edge e if and only if the values of nodes 2 and 3 are a and b, respectively (and f_1 is the current fixed function).

Now suppose for contradiction that B has at most k states other than k sink states and we look at edges that go to these sink states. Note that the number of all edges is k^2 since each of the k states has k edges. Also note that B has to read both of the two leaf states in its computation path, so at least one edge goes to non-sink states. Consequently the number of the above (going to sink states) edges is at most $k^2 - 1$. Since the total number of pairs is k^2, it is impossible to map all those pairs to the edges in a one-to-one fashion, or one of the following two cases must happen:

1. Some pair (a, b) does not appear in any label of these edges. B obviously does not do a correct computation when the values of nodes 2 and 3 are a and b, respectively.
2. Some edge has two (or more) pairs, say (a, b) and (a', b'). Notice that if the state this edge outgoes from reads node 2, then we have $a = a'$. Then the computation of B is not correct again since the output would be the same if the values of node 2 is the same and the values of node 3 are different (recall that our f_1 is in F). Similarly for the case that the state reads node 3 (then $b = b'$).

Thus we can conclude that such B is not a correct BP. ◀

5 Concluding Remarks

The obvious future work is to remove the read-once restriction. Since our main lemma (Lemma 5) heavily depends on the read-once restriction, we do not have any specific approaches to this ultimate goal at this moment. There are a couple of more reasonable sub-goals: One is to prove that if a BP B is thrifty, then B can be converted to a read-once BP without increasing the number of leaf-reading states drastically, or equivalently, to prove that reading a same leaf node twice or more do not help much in thrifty BP's. Another possibility is to attack the case for $h = 4$. This seems more tractable since we can restrict ourselves to the number of leaf-reading states of BP's for $FT_3(k)$ that have several specific properties as shown in [3]. Also this lower bound will outperform the longstanding one by Nečiporuk [9].

References

1 M. Ajtai, L. Babai, P. Hajnal, J. Komlós, P. Pudlák, V. R odl, E. Szemerédi, and G. Turán. Two lower bounds for branching programs. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 30–38. ACM, 1986.

2 S. Cook. An observation on time-storage trade off. *Journal of Computer and System Sciences*, 9(3):308–316, 1974.

3 S. Cook, P. McKenzie, D. Wehr, M. Braverman, and R. Santhanam. Pebbles and branching programs for tree evaluation. *ACM Transactions on Computation Theory (TOCT)*, 3(2):4, 2012.

4 A. Gál and P. McKenzie M. Koucký and. Incremental branching programs. *Theory of Computing Systems*, 43(2):159–184, 2008.

5 S. Jukna and A. Razborov. Neither reading few bits twice nor reading illegally helps much. *Discrete Applied Mathematics*, 85(3):223–238, 1998.

6 S. Jukna and S. Žák. On uncertainty versus size in branching programs. *Theoretical computer science*, 290(3):1851–1867, 2003.

7 B. Komarath and J. Sarma. Pebbling, entropy and branching program size lower bounds. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS)*, page 622, 2013.

8 W. Masek. *A fast algorithm for the string editing problem and decision graph complexity.* PhD thesis, Massachusetts Institute of Technology, 1976.

9 È. Nečiporuk. A boolean function. In *Doklady of the Academy of the USSR*, volume 169, pages 765–766, 1998. English translation in Soviet Mathematics Doklady 7:4, pp.999–1000.

10 M. Paterson and C. Hewitt. Comparative schematology. In *Record of the Project MAC Conference on Concurrent Systems and Parallel Computation*, pages 119–127. ACM, 1970.

11 P. Savický and S. Žák. A read-once lower bound and a (1,+ k)-hierarchy for branching programs. *Theoretical Computer Science*, 238(1):347–362, 2000.

12 J. Simon and M. Szegedy. A new lower bound theorem for read-only-once branching programs and its applications. *Advances in Computational Complexity Theory*, 13:183–193, 1993.

13 I. Wegener. On the complexity of branching programs and decision trees for clique functions. *Journal of the ACM (JACM)*, 35(2):461–471, 1988.

14 D. Wehr. Exact size of smallest minimum-depth deterministic bps solving the tree evaluation problem. Unpublished. http://www.cs.toronto.edu/~wehr/.

15 S. Žák. An exponential lower bound for one-time-only branching programs. In *Mathematical Foundations of Computer Science 1984*, pages 562–566. Springer, 1984.

Computability of the entropy of one-tape Turing machines

Emmanuel Jeandel

LORIA, UMR 7503 – Campus Scientifique, Vandoeuvre-lès-Nancy Cedex, France
emmanuel.jeandel@loria.fr

Abstract

We prove that the maximum speed and the entropy of a one-tape Turing machine are computable, in the sense that we can approximate them to any given precision ϵ. This is counterintuitive, as all dynamical properties are usually undecidable for Turing machines. The result is quite specific to one-tape Turing machines, as it is not true anymore for two-tape Turing machines by the results of Blondel et al., and uses the approach of crossing sequences introduced by Hennie.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Turing Machines, Dynamical Systems, Entropy, Crossing Sequences, Automata

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.421

1 Introduction

The Turing machine is probably the most well known of all models of computation. This particular model has many variations, that all lead to the same notion of computability. The simplest model is the Turing machine with just one tape and one head, that we will consider in this paper.

From the point of view of computability, this model is equivalent to all others. From the point of view of *complexity*, however, the situation is very different. Indeed, it is well known [6, 5, 19] that a language (of finite words) accepted by such a Turing machine in *linear* time is always regular. More precisely, it can be proven that if such a Turing machine is in time $O(n)$ on all inputs, then there is a constant k so that, on any input, the machine passes at most k times in any given position.

We will consider in this paper the Turing machine as a *dynamical system*: The execution is starting from *any* given configuration c, i.e. any initial state, and any initial tape, and we will observe the evolution. While the Turing machine is a model of computation, it is however quite important in the study of dynamical systems. It was intensively studied by Kurka [11], and Moore [14, 15] proved that they can be embedded in various "classical" dynamical systems. As an example, the uncomputability of the entropy of a Turing machine, by Blondel et al. [2] can be used to deduce the uncomputability of the entropy of piecewise-affine maps, proven by Koiran [10] in a different way.

However, these undecidability results are usually obtained for Turing machines with *two tapes*; The basic idea is to use one tape to simulate a given Turing machine M and to control the other tape, that will only move its head without doing any computation or reading any symbol. The *computational complexity* of the new Turing machine will come from the first tape, but the *dynamical complexity* will come from the second tape.

There is a reason why these results use Turing machines with two tapes. We will prove that some dynamical quantities for one-tape Turing machines are actually *computable*, in



© Emmanuel Jeandel;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 421–432

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


the sense that there is an algorithm that, given any $\epsilon > 0$, produces an approximation of the quantity up to ϵ. The two quantities we consider are the *speed* and the *entropy* of a Turing machine. While the most theoretically important quantity is the entropy, we will focus our discussion in the introduction to the speed, which is easier to conceive.

The speed of a Turing machine measures how fast the head goes to infinity. Informally, the speed is greater than α if we can find a configuration c for which the Turing machine is roughly in position αn after n units of time. Note that if α is nonzero, this means that it takes a time $n/\alpha = O(n)$ to be in position n. Now, if we recall a previous result, one-tape Turing machines with running time $O(n)$ on all inputs recognize only regular languages. We will prove, using the same techniques, that this also applies to the maximum speed: If the maximum speed is nonzero, hence the running time on *some* infinite configuration is (asymptotically) *linear*, then there is a *regular* (ultimately periodic) configuration that achieves this maximum speed.

This paper is organized as follows. In the first section, we introduce the formal definitions of the speed and entropy of a Turing machine. In the next section, we proceed to prove the three main theorems: The speed and the entropy are computable, and the speed is actually a rational number, achieved by a ultimately periodic configuration.

2 Definitions

We assume the reader is familiar with Turing machines. A (one-tape) Turing machine M is a (total) map $\delta_M : Q \times \Sigma \mapsto Q \times \Sigma \times \{-1, 0, 1\}$ where Q is a finite set called the set of states, and Σ a finite alphabet.

Now, the best way to see it as a dynamical system might seem unorthodox at first. The idea is to consider the Turing Machine as having a *moving tape* rather than a moving head: A *configuration* is then an element of $\mathcal{C} = Q \times \Sigma^{\mathbb{Z}}$, and the map M on $\mathcal{C}$ is defined as follows: $M((q, c)) = (q', c')$ where $\delta_M(q, c(0)) = (q', a, v)$, $c'(-v) = a$ and $c'(i) = c(i + v)$ for all $i \neq -v$. This distinction is particularly important for the definition of the entropy to be technically correct. However it is more convenient to consider the Turing machines as we are used to, and we will say "the Turing machine is in position i" rather than "the tape has moved i positions to the right".

The speed

Given a configuration $c \in \mathcal{C}$, the *speed* of M on c is the average number of cells that are read per unit of time. Formally, let $s_n(c)$ be the number of *different* cells read during the first n steps of the evolution of the Turing Machine M on input c. Note that s_n is subadditive: $s_{n+m}(c) \leq s_n(c) + s_m(M^n(c))$.

▶ **Definition 2.1.**

$$\overline{s}(c) = \limsup \frac{s_n(c)}{n}, \qquad \underline{s}(c) = \liminf \frac{s_n(c)}{n}.$$

We give two examples.

- Consider a Turing machine with two states q_1, q_2. On q_1, the Turing machine goes to q_2 without changing the position of the head. On q_2 the Turing machine goes right and changes back to q_1. Then $\overline{s}(c) = \underline{s}(c) = 1/2$ for all c.
- Consider a Turing machine with two states $\{L, R\}$ (for Left and Right) and two symbols $\{a, b\}$. In state q, when the machine reads a symbol a, it goes in the direction q. When the machine reads a symbol b, it writes a symbol a instead and changes direction. On

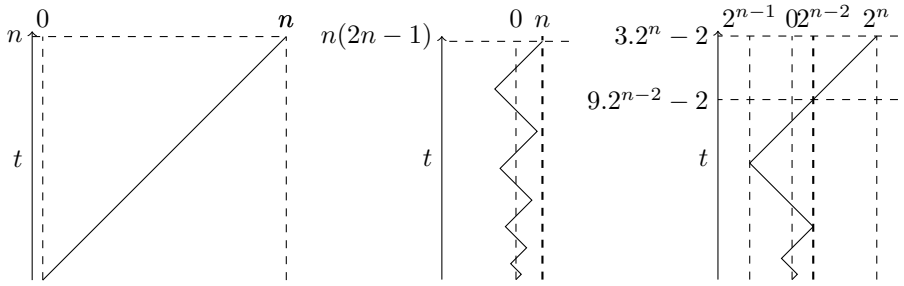


Figure 1 Three different behaviors of the same Turing machine on three different inputs. In the first one, the speed is 1. In the second one the speed is 0. In the third one, the speed is between 1/3 and 1/2. Time goes bottom-up.

input $c = (R, w)$ where w contains only the symbol a, the Turing machine will only go to the right, and $\overline{s}(c) = \underline{s}(c) = 1$. On input $c = (R, w)$ where w contains only the symbol b, the Turing machine will zigzag, and will reach the n-th symbol to the right in time $O(n^2)$, hence will see only $O(\sqrt(n))$ symbols in time n, hence $\overline{s}(c) = \underline{s}(c) = 0$. On input $c = (R, w)$ where w contains b only at all positions $(-2)^i$, the Turing machine will have read (for n even) $2^n + 2^{n-1}$ symbols at time $2^{n+1} + 2^n - 2$ and $\overline{s}(c) = 1/2$, but only $2^{n-1} + 2^{n-2}$ at time $2^{n+1} + 2^{n-2} - 2$, and $\underline{s}(c) = 1/3$. This is illustrated on Figure 1.

Now we define the speed of a Turing machine as the maximum of its average speed on all configurations:

▶ **Definition 2.2.**

$$S(M) = \max_{c\in\mathcal{C}} \underline{s}(c) = \max_{c\in\mathcal{C}} \overline{s}(c) = \lim_n \sup_c \frac{s_n(c)}{n} = \inf_n \sup_c \frac{s_n(c)}{n}\,.$$

The fact that all these definitions are equivalent, and that the maximum speed is indeed a maximum (it is reached by some configuration), is a consequence of the subadditivity of $(s_n)_{n\in\mathbb{N}}$, see [4, Theorem 1.1] or [13] for a more combinatorial proof.

The entropy

The (topological) entropy of a Turing machine is a quantity that measures the complexity of the trajectories. It represents roughly the average number of bits needed to represent the trajectories.

For a configuration c, the *trace* of c is the word $u \in (\Sigma \times Q)^{\mathbb{N}}$ where u_i contains the letter in position 0 of the tape and the state at the i-th step during the execution of M on input c. We note $T(c)$ the trace of c and $T(c)_{|n}$ the first n letters of the trace. Finally, we denote by $T_n = \{T(c)_{|n}, c \in \mathcal{C}\}$

Then the entropy can be defined by

▶ **Definition 2.3.**

$$H(M) = \lim_n \frac{1}{n} \log |T_n| = \inf_n \frac{1}{n} \log |T_n|\,.$$

The limit indeed exists and is equal to the infimum as $(\log |T_n|)_{n\in\mathbb{N}}$ is subadditive. This definition is a specialized version for (moving tape) Turing machines of the general definition of entropy, and was proven equivalent in [16].

We now look again at the examples. In the first case, $T(c)_{|n}$ can take roughly $|\Sigma|^{n/2}$ different values, and $H(M) = 1/2 \log |\Sigma|$. In the second case, the first n letters of $T(c)$ can contain at most $\sqrt{n}$ symbols (b, L) or (b, R) (the maximum is obtained for a configuration with only b). As a consequence, T_n is of size at most $\sum_{i \leq \sqrt{n}} \binom{n}{i} \leq \sqrt{n} \binom{n}{\sqrt{n}}$ so that $H(M) = 0$.

It is possible to give a definition for the entropy that is very similar to the speed. For this, we use *Kolmogorov complexity*. The (prefix-free) Kolmogorov complexity $K(x)$ of a finite word x is roughly speaking the length of the shortest program that outputs x.

A precise definition of Kolmogorov complexity can be found in [3]. Here we just recall a couple of its properties:

- For any alphabet Σ, there exists constants c and c' so that for all words u over Σ, $K(u) \leq |u| \log |\Sigma| + 2 \log |u| + c$ and for all words u, v, $K(uv) \leq K(u) + K(v) + c'$.
- For any computable function f, there exists a constant c so that $K(f(w)) \leq K(w) + c$ whenever $f(w)$ is defined.

For a trace t, define the *lower* and *upper complexity* of t by $\underline{K}(t) = \liminf \frac{K(t_{|n})}{n}$ and $\overline{K}(t) = \limsup \frac{K(t_{|n})}{n}$.

▶ **Theorem 2.4** ([1, 18]). $H(M) = \max_{c \in \mathcal{C}} \underline{K}(T(c)) = \max_{c \in \mathcal{C}} \overline{K}(T(c))$.

From this definition, it will not be surprising that we can obtain results on both speed and entropy using the same arguments.

3 Computability of the speed and the entropy

We will prove in this section that the speed and the entropy of a TM are computable. The proof goes as follows. By the definition of the speed as an infimum, we can compute a sequence s_n so that $S(M) = \inf s_n$. So it is sufficient to find a (computable) sequence s'_n so that $S(M) = \sup s'_n$ to be able to approximate the speed to any given precision ϵ.

To find such a sequence s'_n, it is sufficient to find configurations c_n of near maximal speed. To do that, we need to better understand configurations of maximal speed.

First, we will establish (Propositions 3.1 and 3.2) that a configuration of maximal speed (entropy) cannot do too many zigzags, and must be only finitely many times at any given position. The idea is that revisiting cells that were already visited is a loss of time (and complexity), so the machine should avoid doing it. In the same vein, we can prove that the zigzags must not be too large (Proposition 3.3): the time of the first and last visit of a given cell must be roughly equivalent ($l_n(c) \sim f_n(c)$ in the notation of this proposition).

All this work allows us to redefine the problem as a graph problem: given a weighted (infinite) graph, find the path of minimum average weight (Proposition 3.5). Using the graph approach, we will then prove (Theorem 3.6) that this average minimum weight can be well approximated by considering only *finite* graphs. Finally, the speed and entropy for finite graphs are easy to compute (Theorems 3.7 and 3.9), which ends the proof.

In each section, the proofs will always be done first for the speed, then for the entropy. We deliberately choose to have similar proofs in both cases, to help to understand the proof for the entropy, which is more complex. In particular, some statements about the speed are probably a bit more elaborate than they need to be.

3.1 Biinfinite tapes are no better

The first step in the proof is to simplify the model: we will prove that to achieve the maximum speed (resp. maximum complexity), we only need to consider configurations that never cross the origin, i.e., that stay always on the same side of the tape. This seems quite natural, as changing from a position $i > 0$ to a position $j < 0$ costs at least $i + (-j)$ steps, and might greatly reduce the average speed of the TM on this configuration.

▶ Proposition 3.1. Let c be a configuration for which $S(M) = \lim_n \frac{s_n(c)}{n}$ and suppose $S(M) > 0$. Then, during the computation on input c, the head of M is only finitely many times in any given position i.

Proof. We prove only the result for $i = 0$, the result for all i follows by considering $M^t(c)$ for some suitable t. We suppose by contradiction that the head of M is infinitely often in position 0.

Let k be an integer. As $S(M) > 0$, there must exist a time t for which the head is in position $\pm k$. Let t be the first time when this happens. We may suppose w.l.o.g that at time t the head is in position $+k$. Now let t'_k be the next time the head was in position 0, and finally let t_k be the time at which the head was at its rightmost position in the first t'_k steps.

First, by definition $s_{t_k}(c) = s_{t'_k}(c)$. Furthermore, $t'_k \geq t_k + s_{t_k}(c)/2$. Indeed by definition of t, the leftmost position in the first t_k steps is at most $-(k-1)$ so the TM went further to the right than to the left in the first t_k steps, so that the rightmost position is at least in position $s_{t_k}(c)/2$. Remark also that $t_k \geq k$ (by definition).

From this we obtain

$$\frac{s_{t'_k}}{t'_k} \leq \frac{s_{t_k}}{t_k + s_{t_k}/2} \leq \frac{\frac{s_{t_k}}{t_k}}{1 + \frac{s_{t_k}}{2t_k}}.$$

By taking a limsup on both sides we obtain

$$S(M) \leq \frac{S(M)}{1 + \frac{S(M)}{2}}.$$

A contradiction. ◀

▶ Proposition 3.2. Let c be a configuration for which $H(M) = \lim_n \frac{K(T(c)_{|n})}{n}$ and suppose $H(M) > 0$. Then for any position i, the head of M is only finitely many times in position i.

Proof. It's exactly the same proof. Note that $K(T(c)_{t'_k}) \leq K(T(c)_{t_k}) + O(\log t'_k)$ (The first t'_k bits of $T(c)$ can be recovered if we know only the first t_k bits, and the number of bits we want to recover), and $t'_k \geq t_k + K(T(c)_{t_k})/(2\log|\Sigma|) + O(\log t_k)$ (Indeed $K(T(c)_{t_k}) \leq n\log|\Sigma| + O(\log t_k)$ where $n = s_{t_k}(c)$ is the number of bits read during times $t \leq t_k$, and $t'_k \geq t_k + n/2$), from which we get the same contradiction. ◀

These two propositions state that we only have to deal with configurations that never reach the position $i = 0$ once they leave it at $t = 0$ (replace c by $M^p(c)$ for a suitable p).

If we deal with the disjoint union of the Turing machine M and its mirror (exchange left and right) $\tilde{M}$, we may now assume, and we do in the rest of this section, that the maximum speed and complexity is reached with a configuration that never goes to negative positions $i < 0$ and, if $S(M) > 0$ (resp. $H(M) > 0$), that passes only finitely many times to any given position.

3.2 A reformulation

Recall that we suppose in the following sections that the maximum speed is obtained for a configuration that never goes to negative positions.

Let us call $f_n(c)$ (f for first) the first time the TM reaches position n. Then the average speed on a configuration c (for which the Turing machine never goes in negative positions) can be defined equivalently as $\lim_n \frac{n}{f_n(c)}$. We prove now a stronger statement.

Let us call $l_n(c)$ the *last* time the TM reaches position n. If the TM does not reach position $\pm n$, or if it reaches it infinitely often, let $l_n(c) = \infty$.

▶ Proposition 3.3.

$$S(M) = \max_c \limsup \frac{n}{l_n(c)} = \max_c \liminf \frac{n}{l_n(c)}\,,$$

$$H(M) = \max_c \limsup \frac{K(c_{|n})}{l_n(c)} = \max_c \liminf \frac{K(c_{|n})}{l_n(c)}\,.$$

If the speed (resp. entropy) is nonzero, the maximum is reached for some configuration c for which $l_n(c)$ is never infinite. In particular, for this configuration, $l_n(c) \sim f_n(c)$

Proof. It is clear that $S(M)$ and $H(M)$ are upper bounds, as $n \leq s_{f_n(c)}(c)$ and $K(c_{|n}) \leq K(T(c)_{|f_n(c)}) + O(\log n)$. In particular the result is true if $S(M) = 0$ (resp. $H(M) = 0$).

We first deal with the speed. Let c be a configuration of maximum speed. By the previous subsection, we may suppose that c never reaches negative positions.

Let $t_n = l_n(c)$. Let p be the rightmost position the head reaches before t_n and t'_n the first time this position is reached. Note that $s_{t_n}(c) = s_{t'_n}(c) = p$ (no negative position is ever reached)

From this we get $\lim \frac{t_n}{t'_n} = \lim \frac{t_n}{s_{t_n}(c)} \frac{s_{t'_n}(c)}{t'_n} = 1$.

Note also that $t'_n \geq n$ and $t_n \geq t'_n + s_{t_n} - n$. (The TM is at position $s_{t'_n} = s_{t_n}$ at time t'_n and at position n at time t_n.)

Hence

$$\frac{n}{t_n} \geq \frac{t'_n - t_n}{t_n} + \frac{s_{t_n}}{t_n}\,.$$

From which the result follows.

For the entropy, the proof is almost the same. From $K(T(c)_{t_n}) = K(T(c)_{t'_n}) + O(\log t_n)$, we get again that $\lim_n \frac{t'_n}{t_n} = 1$.

Now $K(T(c)_{t_n}) \leq K(c_n) + (t_n - t'_n)\log|\Sigma| + O(\log t_n)$ (the first t_n bits of $T(c)$ can be recovered if we know t_n and the first p bits of c, hence if we know the first n bits of c and the $p - n \geq t_n - t'_n$ next bits), from which the result follows again. ◀

3.3 Crossing sequences

First denote by $\mathcal{C}^+$ the set of configurations c on which:

- The Turing machine never reaches any positions $i < 0$.
- The Turing machine never reaches the position 0 again once it leaves it at $t = 0$.
- For any $i > 0$, the head of the Turing is only finitely many times in position i.

In the previous section we proved that we only have to deal with configurations in $\mathcal{C}^+$.

The core of the proof is based on *crossing sequences*, introduced by Hennie [6] to obtain complexity lower bounds for one-tape TM.

Let c be a configuration in $\mathcal{C}^+$. The *crossing sequence* at boundary i is the sequence of states of the machine when its head crosses the boundary between the i-th cell and the $i+1$-th cell. We denote by $C_i(c)$ the crossing sequence at boundary i. Note that $C_0(c)$ consists of a single state, which is the initial state of c (the machine never reaches the position 0 anymore) and $C_i(c)$ is finite for $i > 0$.

Crossing sequences have the following property: $C_i(c)$ represents all the exchange of information between the positions $j \leq i$ and the positions $j > i$ of the tape. In particular, if $C_i(c) = C_j(c')$ for two configurations c, c', and if we consider the configuration $\tilde{c}$ that is equal to c up to i then equal to c' (shifted by $i-j$ so that the $j+1$-th cell of c' becomes the $i+1$-th cell of $\tilde{c}$), then the Turing machine on $\tilde{c}$ will behave exactly like c on all positions before i, and as c' (shifted) on positions after i. Hence the crossing sequences capture exactly the behavior of the Turing machine.

The main idea is now that the computation of a Turing machine can be seen as a path on a graph of crossing sequences, where vertices represent crossing sequences and edges link consecutive crossing sequences.

To do this, we now consider the following labeled graph (automaton) G: The vertices of G are all finite words over the alphabet Q (all possible crossing sequences), and there is an edge from w to w' labeled by $a \in \Sigma$ if w and w' are *compatible*, in the sense that it seems possible to find a configuration and a position i so that $C_i(c) = w$, $C_{i+1}(c) = w'$ and a is the letter at position $i+1$ in c (said otherwise, w and w' are two consecutive crossing sequences for some configuration c). The exact definition is as follows. We define recursively two subsets L and R of $Q^* \times Q^* \times \Sigma$ as follows:

- $(\epsilon, \epsilon, a) \in L$, $(\epsilon, \epsilon, a) \in R$
- If $\delta(q_1, a) = (q_2, b, -1)$ then $(q_1 q_2 w, w', a) \in L$ iff $(w, w', b) \in L$
- If $\delta(q_1, a) = (q_2, b, +1)$ then $(q_1 w, q_2 w', a) \in L$ iff $(w, w', b) \in R$
- If $\delta(q_1, a) = (q_2, b, -1)$ then $(q_2 w, q_1 w', a) \in R$ iff $(w, w', b) \in L$
- If $\delta(q_1, a) = (q_2, b, +1)$ then $(w, q_1 q_2 w', a) \in R$ iff $(w, w', b) \in R$

Then there is an edge from w to w' labeled a if and only if $(w, w', a) \in L$.

Note that this echoes a similar definition for two-way finite automata given in [7, 2.6] where $(w, w', a) \in L$ is called "w left-matches w'" (The note in Example 2.15 is particularly relevant). The exact definition above is also hinted at in [17].

Let us explain briefly these conditions. Suppose $\delta(q_1, a) = (q_2, b, +1)$, and suppose that the Turing machine at some point arrives in some cell i from the left, in the state q_1 and sees a. Then by definition, the first symbol from $C_i(c)$ must be q_1. By definition of the local rule δ, the Turing machine will enter state q_2 and go right so that the first symbol in $C_{i+1}(c)$ will be q_2. Now, the next time the Turing machine will come into the cell i, it must be coming from the right, and when it does it will see the symbol b. This explains the rule $(q_1 w, q_2 w', a) \in L$ iff $(w, w', b) \in R$, where w and w' represent the crossing sequences after the second time the Turing machine comes to the cell i.

Now it is clear that a configuration c defines a path in this graph G, and that we can recover the speed of the configuration from the graph, as explained in the following.

A *path* in the graph G is a sequence $p = \{(w_i, u_i)\}_{i<N}$ where w_i is a vertex of G and u_i a letter from Σ so that $(w_i, w_{i+1}, u_i) \in L$ for all $i < N-1$. A *valid* path is an infinite path ($N = \infty$) so that w_0 consists of one single letter (state). We denote by $\mathcal{P}(G)$ the set of valid paths of a graph G.

The following facts are obvious:

▶ Fact 3.4. For any $c \in \mathcal{C}^+$, $\{(C_i(c), c_i)\}_{i \geq 0}$ is a valid path in G.

Furthermore, for any valid path $p = \{(w_i, u_i)\}_{i\geq 0}$, there exists a configuration $c \in \mathcal{C}^+$ so that $u_i = c_i$ and $C_i(c)$ is a prefix of w_i.

Note that it is indeed possible for w_i to be strictly larger than $C_i(c)$.

We are now able to redefine the speed and the complexity based on the graph G. If p is a finite path (N is finite), the *length* of p is $|p| = N$, the *weight* of p is $weight(p) = \sum_{i<N} |w_i|$, and the *complexity* of p is $K(p) = K(u_0 \dots u_{N-1})$

If $p = (u_i, w_i)_{i\geq 0}$ is an infinite path, and $p_{|n} = (u_i, w_i)_{i\leq n}$, the *average speed* of p is $\underline{s}(p) = \liminf \frac{|p_{|n}|}{weight(p_{|n})}$ and the *average complexity* of p is $\underline{K}(p) = \liminf \frac{K(p_{|n})}{weight(p_{|n})}$. We define similarly $\overline{s}(p)$ and $\overline{K}(p)$.

Now note that $\sum_{i<n} |C_i(c)|$ is bounded from below by the first time the TM goes to the position n, and from above by the last time the TM goes to position n. So by the previous section

▶ Proposition 3.5.

$$S(M) = \max_{p\in\mathcal{P}(G)} \underline{s}(p) = \max_{p\in\mathcal{P}(G)} \overline{s}(p)\,,$$

$$H(M) = \max_{p\in\mathcal{P}(G)} \underline{K}(p) = \max_{p\in\mathcal{P}(G)} \overline{K}(p)\,.$$

Now to obtain the main theorems, let G_k be the subgraph of G obtained by taking only the vertices of size $|w_i| \leq k$.

▶ **Theorem 3.6.**

$$S(M) = \sup_k \sup_{p\in\mathcal{P}(G_k)} \overline{s}(p)\,,$$

$$H(M) = \sup_k \sup_{p\in\mathcal{P}(G_k)} \overline{K}(p)\,.$$

This means we only have to consider finite graphs to compute the speed (resp. entropy). We will prove in the next section that the speed and the entropy are computable for finite graphs, which will give the result.

Before going to the proof, some intuition. Let p be a path of maximum speed $S(M) > 0$. For the speed to be nonzero, vertices of large weight cannot be too frequent in p. Now the idea is to *bypass* these vertices (by using other paths in the graph G) to obtain a new path p' with almost the same average speed. For the speed, it's actually possible to obtain a path p' of the *same* speed (this will be done in the next section). However, for the entropy, it is likely that these paths were actually of great complexity so that their removal gives us a path of smaller (yet very near) average complexity.

Proof. First, the speed. One direction is obvious by definition. We suppose that $S(M) > 0$, otherwise the result is trivial. Let p be a path of maximum speed.

Let k be any integer so that $1/k < S(M)$. For any vertex w and w' of size less or equal to k so that p goes through w and w' in that order, choose some finite path $P(w, w')$ from w to w'. Now let K be an upper bound on the weights of all those paths.

The idea is now simple: we will change p so that it will not go through any vertices w of size $|w| > K$: Whenever there is a vertex $\tilde{w}$ of size greater than K, we will look at the last vertex w before it of size less or equal to k, and to the first vertex w' after it of size less or equal to k, and we will replace the portion of this path by $P(w, w')$. Let's call p' this new path. Note that there must exist such a vertex w', otherwise all vertices will be of size greater than k after some time, which means the speed on p is less than $1/k$, a contradiction.

Now we prove this construction works. First, p is of average speed $S(M)$, hence there exists an integer n so that for all $m \geq n$

$$\frac{m}{weight(p_{|m})} \geq S(M)/2\,.$$

Now let m so that the vertex w_m of p is of size less than k. We will look at how the m first positions of p were changed into p'. Let m' be the position of the vertex w_m in p' (w_m still appears in p' as we only change vertices of size greater than k).

By the above inequality, it is clear that in the m first position of the path p, there is at most $2m/(kS(M))$ vertices of size greater than k. All other vertices still appear in p', so that $m' \geq m - 2m/(kS(M))$. Furthermore, at each time, we replace a finite path by a path of smaller weight (each path was of weight at least K, and each new one is of weight at most K).

As a consequence, for this new path p' we have

$$\frac{m'}{weight(p'_{|m'})} \geq \frac{m - 2m/(kS(M))}{weight(p_{|m})}\,.$$

Hence

$$\overline{s}(p') \geq S(M) - 2/k\,.$$

We have proven that some path in G_K is at least $2/k$ to the optimal speed, which proves the result.

The proof for the entropy is, as always, very similar. We start from $1/k < H(M)/(\log|\Sigma|))$, which guarantees that infinitely many vertices are of weight less than k. As before, we will choose K greater than all weights, but now also greater than $k|Q|^{k+1}$.

First, $K(p_{|n}) \leq n\log|\Sigma| + O(\log n)$, so $\underline{K}(p) \leq \underline{s}(p)\log|\Sigma|$, so we may choose n so that for all $m \geq n$

$$\frac{m}{weight(p_{|m})} \geq H(M)/(2\log|\Sigma|)\,.$$

Let $\alpha = 2\log|\Sigma|/H(M)$. With this notation, this implies that for every $m \geq n$, there are at most $m\alpha/k$ (resp. $m\alpha/K$) vertices of size at least k (resp K) in the first m positions of p.

We now have to evaluate $K(p'_{|m'})$. $p_{|m}$ can be recovered from $p'_{|m'}$ by deleting some letters and inserting new ones.

First remark that there are at most $m\alpha/K$ vertices of size at least K in p_m, so we did at most $m\alpha/K$ cuts. In each cut, we inserted at most $|Q|^{k+1}$ letters (the maximal length of a path $P(w,w')$), so we deleted at most $|Q|^{k+1}m\alpha/K \leq m\alpha/k$ letters from $p'_{|m'}$. In particular $m' \leq m(1+\alpha/k)$

We only cut vertices of size at least k, and there are at most $m\alpha/k$ such vertices, so we added at most $m\alpha/k$ letters to $p'_{|m'}$. In particular $m' \geq m(1-\alpha/k)$.

Now the letters we deleted from $p'_{|m'}$ can be encoded into a word over $\{0,1\}$ (specifying which letters we deleted) with at most $m\alpha/k$ symbols "1". For each size $l \leq m\alpha/k$, there are at most $\binom{m'}{m\alpha/k}$ words with l symbols "1", so each such word has complexity at most the logarithm of this number (up to a logarithmic factor to specify l), that is $m'E(\alpha/(k-\alpha)) + o(m)$ where $E(p) = -p\log p - (1-p)\log(1-p)$.

We do the same for the letters we add to p', but we also need to know which letters we had, which can be described by a word of size $m\alpha/k$ and we obtain

$$K(p'_{|m'}) \geq K(p_m) - 2m'E(\alpha/(k-\alpha)) - \lceil m\alpha/k \rceil \lceil \log |\Sigma| \rceil + o(m)\,.$$

Now $weight(p'_{|m'}) \leq weight(p_{|m})$ and $weight(p'_{|m'}) \geq m' \geq m(1-\alpha/k)$:

$$\frac{K(p'_{|m'})}{weight(p'_{|m'})} \geq \frac{K(p_m)}{weight(p_m)} - 2E(\alpha/(k-\alpha)) - \frac{\alpha}{k-\alpha}\lceil \log |\Sigma| \rceil + o(1)\,.$$

Now take the limit (superior) as m tends to infinity:

$$\overline{K}(p') \geq H(M) - 2E(\alpha/(k-\alpha)) - \frac{\alpha}{k-\alpha}\lceil \log |\Sigma| \rceil\,.$$

Now the quantity to the right tends to $H(M)$ when k tends to infinity, which proves the result. ◀

3.4 The main theorems

Now we can explain how to use the last result to prove the main theorems. As hinted above, we only have to be able to compute the speed (and the entropy) from below.

▶ **Theorem 3.7.** *There exists an algorithm that, given a Turing machine M and a precision ϵ, computes $S(M)$ to a precision ϵ.*

Proof. We only have to explain how to compute the maximum speed for a finite graph G. First, we may trim G so that all vertices are reachable from a vertex of size 1. It is then obvious that the maximum speed is obtained by a path that goes to then follow a cycle of minimum average weight, so the maximum speed is exactly the inverse of the minimum average weight. This is easily computable, see [9] for an efficient algorithm. ◀

We can say a bit more

▶ **Theorem 3.8.** *The maximum speed of a Turing machine $S(M)$ is a rational number. It is reached by a configuration which is ultimately periodic.*

Proof. We suppose that $S(M) > 0$ otherwise the result is clear. We will prove that the sequence $\sup_{p \in G_k} \overline{s}(p)$ is stationary. Let $k = 1 + \lceil 1/S(M) \rceil$. Let $K = k(k+1)|Q|^{k+1}$.

Now we look at $\sup_{p \in G_{K'}} \overline{s}(p)$ for some $K' \geq K$. The maximum is reached for some path that reach some cycle of minimum average weight.

Note that this cycle cannot be of length greater than $(k+1)|Q|^{k+1}$. Indeed, denote by m the length of this cycle. As there are at most $|Q|^{k+1}$ vertices in this cycle of length at most k, the average speed on this cycle is less than

$$\frac{m}{(k+1)(m-|Q|^{k+1})} \leq 1/k < S(M)\,.$$

Now, there cannot be any vertices in this cycle of length at least $k(k+1)|Q|^{k+1}$. otherwise the average speed would be less than

$$\frac{(k+1)|Q|^{k+1}}{k(k+1)|Q|^{k+1}} \leq 1/k < S(M)\,.$$

Hence this cycle is already in G_K.

Now if we look at the cycle of minimal average weight in G_K that can be reached in G, hence in G_P from some P, then it is clear that $S(M)$ is exactly the inverse of the average weight of this cycle, and it is reached for some path p in G_P that reaches then follows this cycle. ◀

Note that, while the maximum speed is a rational number, there is no algorithm that actually computes this rational number (we are only able to approximate it up to any given precision). This can be proven by an adaptation of the proof of the undecidability of the existence of a periodic configuration in a Turing machine [8].

Now we do the same for the entropy:

▶ **Theorem 3.9.** *There exists an algorithm that, given a Turing machine M and a precision ϵ, computes $H(M)$ to a precision ϵ.*

Proof. We only have to explain how to compute the maximum complexity for a finite graph G. However, we do not know how to do this in the whole generality. We will only prove how to do it for the graphs G_k, that have an additional property, the *diamond* property: given two vertices w,w' and a word u, there is at most one path from w to w' labeled by u.

First we trim G_k so that any vertex of G_k is reachable from a vertex of size 1.

For a given k, we consider a set B_k of infinite words over the alphabet $(Q \times \Sigma) \cup Q$ defined as follows: A word is in B_k if and only if it does not contain more than $k-1$ consecutive letters in Q, more than 1 consecutive letters in $Q \times \Sigma$, and all factors of the form $(a,q)w(b,q')w'(c,q')$ satisfy than there is a edge from qw to $q'w'$ labeled by b.

Now it it clear that if $p = \{(u_i, q_i w_i)\}_{i \geq 0}$ is an infinite path in G_k, then the word $(u_0,q_0)w_0(u_1,q_1)w_1\dots$ is a word of B_k. Conversely, any word of B_k, up to the deletion of at most $k+1$ letters at its beginning, represents a path in G_k.

Moreover, $K((u_0,q_0)w_0\dots(u_n,q_n)w_n) = K(u_0\dots u_n) + O(1) = K(p_{|n}) + O(1)$. Indeed, we can recover all the states knowing only w_0 and w_n, as the graph has no diamond. Furthermore, the length of $(u_0,q_0)w_0\dots(u_n,q_n)w_n$ is exactly $weight(p_{|n})$.

This means that the maximum complexity on the graph G_k can be computed as:

$$\sup_{w \in B_k} \limsup \frac{K(w_0\dots w_n)}{n}.$$

And we know how to compute this. Indeed, B_k is what is called a subshift of finite type (it is defined by a finite set of forbidden words), for which the above quantity is exactly the entropy (!) of B_k [1, 18], which is easy to compute, see e.g., [12].

To better understand what we did in this theorem, the intuition is as follows: Computing the entropy of the trace is difficult, but the trace can be approximated by taking into account only configurations for which we cross at most k times the frontier between any two consecutive cells. For this approximation T_k of the trace, we can reorder the letters inside the trace so that transitions corresponding to the same position are consecutive, and this does not change the entropy. However, it makes it easier to compute. ◀

Open Problems

From the point of view of dynamical systems, the entropy and the speed (called the maximal Lyapunov exponent) are among the few well known invariants, and thus the result is quite important in this context. However, from the point of view of computer science, it also makes sense to look at the *average* speed, and we are currently trying to compute this number.

An important open problem is to strengthen the last theorem, and actually characterize the exact numbers that can arise as entropies of Turing machines. It cannot be all nonnegative computable numbers, as an enumeration of Turing machines would give us an enumeration of these numbers, which is impossible by an easy diagonalization argument. We have examples showing that the supremum in the theorem is not always reached, which

means it might be possible to obtain different numbers with Turing machines than with finite graphs (which are well known), but the main question remain open.

Finally, the situation for Turing machines with two tapes is not clear. Of course, we know that the speed (resp. entropy) is not computable [2] (there is no algorithm that given a Turing machine and a precision ϵ computes the speed up to ϵ), but we know of no example where the speed (resp. the entropy) is not a computable number.

Acknowledgements. The author thanks the anonymous referees for various comments that led to noticeable improvements to the quality of this article.

References

1 A. A. Brudno. Entropy and the complexity of the trajectories of a dynamical system. *Transactions of the Moscow Mathematical Society*, 44(2):127–151, 1983.

2 Jean-Charles Delvenne and Vincent D. Blondel. Quasi-periodic configurations and undecidable dynamics for tilings, infinite words and Turing machines. *Theoretical Computer Science*, 319:127–143, 2004.

3 Rodney G. Downey and Denis R. Hirschfeldt. *Algorithmic Randomness and Complexity*. Theory and Applications of Computability. Springer, 2010.

4 De-Jun Feng and Weng Hang. Lyapunov Spectrum of Asymptotically Sub-additive Potentials. *Communications in Mathematical Physics*, 297(1):1–43, 2010.

5 J. Hartmanis. Computational Complexity of One-Tape Turing Machine Computations. *Journal of the ACM (JACM)*, 15(2):325–339, 1968.

6 F.C. Hennie. One-Tape, Off-Line Turing Machine Computations. *Information and Control*, 8:553–578, 1965.

7 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

8 Jarkko Kari and Nicolas Ollinger. Periodicity and Immortality in Reversible Computing. In *MFCS 2008*, LNCS 5162, pages 419–430, 2008.

9 Richard M. Karp. A Characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978.

10 Pascal Koiran. The Topological Entropy of Iterated Piecewise Affine Maps is Uncomputable. *Discrete Mathematics and Theoretical Computer Science*, 4(2):351–356, 2001.

11 Petr Kurka. On topological dynamics of Turing machines. *Theoretical Computer Science*, 174:203–216, 1997.

12 Douglas A. Lind and Brian Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, New York, NY, USA, 1995.

13 László Máté. On infinite composition of affine mappings. *Fundamenta Mathematicae*, 159:85–90, 1999.

14 Cristopher Moore. Generalized one-sided shifts and maps of the interval. *Nonlinearity*, 4(3):727–745, 1991.

15 Cristopher Moore. Generalized shifts: unpredictability and undecidability in dynamical systems. *Nonlinearity*, 4(2):199–230, 1991.

16 Piotr Oprocha. On entropy and Turing machine with moving tape dynamical model. *Nonlinearity*, 19(10):2475–2487, 2006.

17 Giovanni Pighizzini. Nondeterministic one-tape off-line Turing machines and their time complexity. *Journal of Automata, Languages and Combinatorics*, 14(1):107–124, 2009.

18 Stephen G. Simpson. Symbolic Dynamics: Entropy = Dimension = Complexity. accepted for publication in *Theory of Computung Systems*.

19 B. A. Trakhtenbrot. Turing Computations with Logarithmic Delay. *Algebra i Logika*, 3(4):33–48, 1964. (In russian).

Computing Optimal Tolls with Arc Restrictions and Heterogeneous Players

Tomas Jelinek[1], Marcus Klaas[2], and Guido Schäfer[3]

1 **VU University Amsterdam, The Netherlands**
`t.jelinek@student.vu.nl`
2 **University of Amsterdam, The Netherlands**
`mail@marcusklaas.nl`
3 **CWI and VU University Amsterdam, The Netherlands**
`g.schaefer@cwi.nl`

Abstract

The problem of computing optimal network tolls that induce a Nash equilibrium of minimum total cost has been studied intensively in the literature, but mostly under the assumption that these tolls are unrestricted. Here we consider this problem under the more realistic assumption that the tolls have to respect some given upper bound restrictions on the arcs. The problem of taxing subnetworks optimally constitutes an important special case of this problem. We study the *restricted network toll problem* for both non-atomic and atomic (unweighted and weighted) players; our studies are the first that also incorporate *heterogeneous* players, i.e., players with different sensitivities to tolls.

For non-atomic and heterogeneous players, we prove that the problem is NP-hard even for single-commodity networks and affine latency functions. We therefore focus on parallel-arc networks and give an algorithm for optimally taxing subnetworks with affine latency functions. For weighted atomic players, the problem is NP-hard already for parallel-arc networks and linear latency functions, even if players are homogeneous. In contrast, for unweighted atomic and homogeneous players, we develop an algorithm to compute optimal restricted tolls for parallel-arc networks and arbitrary (standard) latency functions. Similarly, for unweighted atomic and heterogeneous players, we derive an algorithm for optimally taxing subnetworks for parallel-arc networks and arbitrary (standard) latency functions.

The key to most of our results is to derive (combinatorial) characterizations of flows that are inducible by restricted tolls. These characterizations might be of independent interest.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.1.6 Optimization, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Network routing games, coordination mechanisms, network tolls, taxing subnetworks, heterogeneous players

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.433

1 Introduction

Motivation and Background. It is a well-known fact that selfish route choices in network routing applications result in outcomes that are undesirable for the society as a whole. In urban road traffic, for example, selfish route choices lead to unnecessary traffic jams, thereby causing environmental pollution, waste of natural resources, time and money. The Texas A&M Transportation Institute states in its *2012 Urban Mobility Report* [13, page 1]: "The 2011 data are consistent with one past trend, congestion will not go away by itself – action is needed! [...] The problem is very large. In 2011, congestion caused urban Americans to travel


© Tomas Jelinek, Marcus Klaas, and Guido Schäfer;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 433–444

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


5.5 billion hours more and to purchase an extra 2.9 billion gallons of fuel for a congestion cost of $121 billion."

Road pricing is recognized to be one of the most effective means to reduce congestion in networks. The idea is to let the users pay for the usage of certain segments of the network by imposing tolls. Typically, such tolls incite the users to change their commuting behavior, e.g., by opting for alternative, possibly slightly longer routes, avoiding certain parts of the network at peak hours, etc. As a result, the network becomes less congested because the traffic is better distributed through the network. Currently, road pricing systems are implemented successfully in several large cities across the world (like in Singapore, Kopenhagen, Tel Aviv, London, Dubai, etc.). A fundamental problem in this context is to determine tolls such that the overall congestion of the underlying network is reduced.

In this paper, we study the problem of computing optimal tolls for the arcs of a given network that induce a Nash equilibrium of minimum total cost. This problem has been studied intensively in the literature for decades. One of the earliest articles addressing this problem is due to Beckman, McGuire and Winsten [1], where they show that *marginal cost tolls* induce an optimal flow as Nash equilibrium in non-atomic network routing games. However, most previous studies were conducted under the assumption that the tolls are *unrestricted*. This assumption is too simplistic in many situations and significantly limits the applicability of such tolls in practice. For example, by using marginal cost tolls we lose the ability to control which arcs of the network are tolled and by how much. Clearly, this is undesirable in real-world applications where one can impose tolls only on certain arcs of the network and wants to ensure that they do not exceed predefined amounts.

Only recently, researchers have started to investigate more refined network toll problems like the *taxing subnetworks problem* [8, 10], in which only a subset of the arcs can be tolled, or the *restricted network toll problem* [2], in which tolls have to respect some upper bound restrictions on the arcs. All three studies [2, 8, 10] focus on the case of *non-atomic* players that are *homogeneous*, i.e., all players are assumed to have equal sensitivities to tolls. Here we further advance these investigations.

Our Contributions. We study the *restricted network toll problem* [2] both for non-atomic and atomic (unweighted and weighted) players. In our studies we consider for the first time also the case of *heterogenous* players, i.e., players may have different sensitivities to tolls. Capturing heterogenous players is particularly important if it comes to applications where users experience different disutilities of travel time and monetary cost (due to the tolls). As it turns out, the heterogenous player case gives rise to several new challenges in devising algorithms for the computation of optimal restricted tolls.

The main contributions presented in this paper are as follows:

- In Section 3 we consider the case of non-atomic, heterogeneous players. We prove that the problem of computing optimal restricted tolls is NP-hard even for single-commodity networks with affine latency functions. In light of this negative result, we then focus on parallel-arc networks and derive a combinatorial characterization of flows that are inducible by restricted tolls. Exploiting this characterization, we derive an optimal algorithm for taxing subnetworks with affine latency functions.
- In Section 4 we consider atomic players. We first observe that for weighted players the problem of computing optimal restricted tolls is NP-hard already for parallel-arc networks with linear latency functions (even if the players are homogeneous). We therefore focus on unweighted players and parallel-arc networks with standard latency functions and derive an optimal algorithm to compute restricted tolls for homogeneous players. Further, we obtain an optimal algorithm for taxing subnetworks for heterogeneous players.

As in previous works [2, 8, 10], most of our exact algorithms work only for parallel-arc networks. However, our studies also reveal that this is basically unavoidable, unless one is willing to resort to approximation algorithms (assuming that P $\neq$ NP). Moreover, from a practical point of view the restricted network toll problem for parallel-arc networks is still very well motivated: for example, it captures the problem of pricing fast-lanes (or priority-lanes) of highways that can be used to bypass heavy traffic (like in Tel-Aviv).

Our Techniques. The main difficulty that we face here in designing algorithms to compute optimal restricted tolls is that the underlying problem is a bi-level optimization problem: the feasible tolls constitute a compact set over which we wish to optimize the cost of the corresponding Nash equilibria (which in turn are determined by the tolls). Typically, such bi-level optimization problems are hard to tackle.

The key to most of our algorithmic results is to derive characterizations of flows that are inducible by restricted tolls. For the unrestricted case, several such characterizations can be found in the literature (see, e.g., [4, 6, 9, 15]). Moreover, some of them can be adapted to also incorporate upper bound restrictions on tolls. For example, Fleischer et al. [6] characterize the inducibility of a flow for non-atomic, heterogeneous players by the existence of an optimal solution satisfying certain minimality conditions for a cleverly chosen linear program. The upper bound restrictions can easily be added to this LP formulation such that the same characterization continues to hold. However, the crux is that we cannot simply use this characterization here because it reveals very little about the *structure* of the flows that are inducible by these restricted tolls. In contrast, we derive characterizations that reveal some structural properties of the inducible flows which we then exploit to design our algorithms.

Related Work. Beckman, McGuire and Winsten [1] proved that for non-atomic, homogeneous players marginal cost tolls induce an optimal flow as a Nash equilibrium. The existence of such tolls for non-atomic, heterogeneous players has first been established for single-commodity networks by Cole, Dodis and Roughgarden [4] and then extended to multi-commodity networks by Yang and Huang [15] (see also the independent works by Fleischer, Jain and Mahdian [6] and Karakostas and Kolliopoulos [9]). Fleischer [5] shows that for single-commodity networks linear tolls (in terms of the maximum latency of the optimal flow) are sufficient to enforce an optimal flow as Nash equilibrium.

In the literature one distinguishes between tolls that are *weakly-optimal*, i.e., at least one induced Nash equilibrium is an optimal flow, and *strongly-optimal*, i.e., every induced Nash equilibrium is an optimal flow. Swamy [14] proved the existence of weakly-optimal tolls for atomic, heterogeneous players and splittable flow. For atomic, homogeneous players and unsplittable flow, Caragiannis, Kaklamanis and Kanellopoulos [3] show that for linear latency functions strongly-optimal tolls do not exist for multi-commodity networks or if the players are weighted. They also show that strongly-optimal tolls exist for parallel-arc networks with linear latency functions and unweighted players. Subsequently, Fotakis and Spirakis [7] proved that weakly-optimal tolls can be computed efficiently for single-commodity networks and that these tolls are strongly-optimal for series-parallel networks.

In this paper, we focus on the computation of weakly optimal tolls. Most related to our work are the recent articles [2, 8, 10]. As already mentioned, these studies concentrate on the case of non-atomic players that are homogeneous. Hoefer, Olbrich and Skopalik [8] study the problem of optimally taxing subnetworks. They show that this problem is NP-hard for two-commodity networks and affine latency functions by a non-trivial reduction from partition. We borrow several insights of their proof to establish NP-hardness for single-commodity

networks and affine latency functions in the case of heterogeneous, non-atomic players here. They also derive an algorithm to compute optimal tolls for parallel-arc networks and affine latency functions. Recently, Kleinert et al. [10] extended the algorithm in [8] for optimally taxing subnetworks for parallel-arc networks to more general latency functions. The algorithm guarantees polynomial running time for instances satisfying the *inverse concavity property* (see [10] for details). The restricted network toll problem considered here was introduced by Bonifaci, Salek and Schäfer [2]. The authors show that optimal restricted tolls can be computed efficiently for parallel-arc networks and affine latency functions and also derive bounds on the efficiency of restricted tolls for multi-commodity networks and polynomial latency functions.

2 Preliminaries

We provide formal definitions of the concepts introduced in the Introduction. Suppose we are given an instance $\mathcal{I} = (G = (V, A), (\ell_a)_{a\in A}, (s_i, t_i)_{i\in[k]}, (r_i)_{i\in[k]})$ of the non-atomic network routing game, where G is a directed graph with latency functions $(\ell_a)_{a\in A}$ and k commodities $(s_i, t_i)_{i\in[k]}$ of demand $(r_i)_{i\in[k]}$. The goal of every player is to send his flow along a shortest latency path from its source s_i to its destination t_i. Let $\mathcal{P}_i$ denote the set of all simple directed s_i, t_i-paths in G and define $\mathcal{P} := \cup_{i\in[k]}\mathcal{P}_i$. An outcome of the game is a flow $f : [k] \times \mathcal{P} \to \mathbb{R}_+$ that is feasible, i.e., $\sum_{P\in\mathcal{P}_i} f^i_P = r_i$ for every $i \in [k]$. Given a flow f, the total flow on arc $a \in A$ is defined as $f_a := \sum_{i\in[k]} \sum_{P\in\mathcal{P}:a\in P} f^i_P$. The set of arcs used by commodity i is denoted by A^+_i and we define $A^+ = \cup_{i\in[k]} A^+_i$. We define the latency of a path $P \in \mathcal{P}$ with respect to f as $\ell_P(f) := \sum_{a\in P} \ell_a(f_a)$. The total cost $C(f)$ of f is given by its average latency, i.e., $C(f) := \sum_{P\in\mathcal{P}} f_P \ell_P(f)$. A flow that minimizes $C(\cdot)$ is called *optimal* and denoted by f^*. A feasible flow f is called a *Nash flow* (or *Wardrop flow*) with respect to $\ell := (\ell_a)_{a\in A}$ if

$$\forall i \in [k],\ \forall P \in \mathcal{P}_i,\ f^i_P > 0 : \qquad \ell_P(f) \le \ell_{P'}(f) \quad \forall P' \in \mathcal{P}_i. \tag{1}$$

Atomic network routing games are very similar. The only difference to the non-atomic setting is that the flow for each commodity has to be routed along a single path, i.e., $\forall i \in [k],\ \exists! P \in \mathcal{P}_i$ such that $f^i_P = r_i$. What is refered to as commodity $i \in [k]$ in the non-atomic setting, is considered a player of weight r_i in the atomic setting. (Note that we do not assume that the commodities are distinct. Thus, different players might have the same source and destination.) A flow can be regarded as a mapping from $[k]$ into $\mathcal{P}$. On parallel arc networks, $a(i)$ may be used to denote the arc player i uses in a given flow. Players are said to be unweighted if $r_i = 1$ for all $i \in [k]$. In the atomic setting, a feasible flow is a Nash flow when

$$\forall i \in [k],\ \forall P \in \mathcal{P}_i,\ f_P > 0 : \qquad \sum_{a\in P} \ell_a(f_a) \le \sum_{a\in P'} \ell_a(f_a + r_i) \quad \forall P' \in \mathcal{P}_i. \tag{2}$$

Throughout this paper, we assume that the latency functions are non-negative, non-decreasing, differentiable and semi-convex, i.e., $x \cdot \ell_a(x)$ is convex for every arc $a \in A$; such latency functions are also called *standard* [11]. In the non-atomic setting, the cost of a Nash flow is unique if the latency functions are standard; this property is not guaranteed to hold for atomic players.

We study the *restricted network toll problem* as introduced in [2]: We are given an instance $\mathcal{I}$ of the network routing game and threshold values $\theta := (\theta_a)_{a\in A}$ on the arcs. The goal is to determine non-negative tolls $\tau := (\tau_a)_{a\in A}$ for the arcs of the network that obey the bounds

defined by the threshold functions $(\theta_a)_{a\in A}$. More formally, a toll vector $\tau = (\tau_a)_{a\in A}$ is called *θ-restricted* if for every arc $a \in A$, $0 \leq \tau_a \leq \theta_a$. Additionally, we are given a non-negative vector of player *sensitivities* (also called *types*) $\alpha := (\alpha_i)_{i\in[k]}$. α_i represents the fraction of how a player of type i values the cost of one unit of time (latency) compared to one unit of money (toll).[1] Without loss of generality, we assume that $0 < \alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_k$ throughout this paper. When $\alpha_i = \alpha_j$ for all $i, j \in [k]$, we say the players are *homogeneous* with respect to their sensitivity to toll; otherwise, we call them *heterogeneous*. In the former case, we can assume without loss of generality that α is normalized to 1.

Given a feasible flow f, we define the *combined cost* that a player of type i experiences by traversing arc $a \in A$ as $\phi_a^i(f_a) = \ell_a(f_a) + \alpha_i\tau_a(f_a)$. The goal of a player of type i is to choose a path P that minimizes the combined cost $\ell_P(f) + \alpha_i\tau_P$, where $\tau_P := \sum_{a\in P}\tau_a$. For θ-restricted tolls τ, let f^τ denote a Nash flow that is induced by τ, i.e., f^τ is a Nash flow with respect to the combined costs $(\phi^i)_{i\in[k]}$. Given the restrictions $\theta = (\theta_a)_{a\in A}$ on the arcs, a θ-restricted toll vector τ is *optimal* if there exists a Nash flow f^τ that is inducible by τ whose cost satisfies $C(f^\tau) \leq C(f^{\bar\tau})$ for all Nash flows $f^{\bar\tau}$ that are inducible by θ-restricted tolls $\bar\tau$. The optimization problem that we are considering in this paper is to compute θ-restricted tolls that are optimal.

An important special case of the θ-restricted network toll problem is the so-called *taxing subnetworks problem* [8]. Here we are given a set $T \subseteq A$ of arcs that are taxable arbitrarily while the arcs in $N := A \setminus T$ are non-taxable. This problem is equivalent to setting $\theta_a = \infty$ for every $a \in T$ and $\theta_a = 0$ for every $a \in N$.

3 Non-Atomic Players

We focus on the non-atomic, heterogeneous player case in this section. We first prove that the problem of computing optimal θ-restricted tolls is NP-hard for single-commodity networks.

▶ **Theorem 1.** *The problem of deciding whether there exist θ-restricted tolls that induce a flow of social cost at most K is NP-complete, even for single-commodity networks and affine latency functions with non-atomic and heterogeneous players.*

Our proof is an adaptation of the NP-hardness result for taxing subnetworks for two-commodity instance, affine latency functions and homogeneous players presented in [8]. The idea is to 'mimic' the behavior of their two-commodity instance by a single-commodity instance. To this aim, we have to overcome several difficulties. The proof is involved and due to lack of space deferred to the full version of the paper.

In light of Theorem 1, we subsequently restrict our attention to parallel-arc networks. We first establish a combinatorial characterization of inducible flows which we then use to derive an optimal algorithm for taxing subnetworks.

3.1 Characterization for Parallel-Arc Networks

We present a characterization of flows that are inducible by θ-restricted tolls in parallel-arc networks. Our approach is algorithmic: We first derive an algorithm for computing tolls that induce a given flow *without* any restrictions. We then show that the computed tolls are component-wise minimal and use this insight to derive our final characterization. Our characterization holds for arbitrary latency functions.

[1] Note that players having the same source and destination might still have different sensitivities in our setting because we do not assume that the commodities are distinct.

Algorithm 1: Computation of flow-inducing tolls

Input: flow $f = \sum_{i\in[k]} f^i$
Output: tolls $\tau = (\tau_a)_{a\in A}$ inducing f
1 $\tau \leftarrow \mathbf{0}$
2 **for** $a = 2 \to m^+$ **do**
3 $\quad \Delta_a \leftarrow \frac{\ell_a - \ell_{a-1}}{\hat{\alpha}_{\min}(a)}$
4 $\quad$ **for** $\bar{a} = 1 \to (a-1)$ **do** $\tau_{\bar{a}} \leftarrow \tau_{\bar{a}} + \Delta_a$
5 **end**
6 **return** τ

We assume that we have k different player types. Let $f = (f^i)_{i\in[k]}$ be a given flow, where f^i is the flow of player type i. Note that if f is fixed then all latencies of the arcs become constants. For notational convenience, we will therefore omit the reference to f and write ℓ_a to refer to $\ell_a(f_a)$. Throughout this section we assume that $A = [m]$ and that the arcs are ordered such that $\ell_1 \le \ell_2 \le \cdots \le \ell_m$. This is without loss of generality because we can always relabel the arcs accordingly.

Let $m^+ = \max\{a \in A^+\}$ be the arc of largest latency that is used under f. For every arc $a \in A$ we define $L(a) := \{a' \in A \mid a' \ge a\}$ as the set of arcs that succeed a in the order above (including a). Note that the latency of every arc in $L(a)$ is at least ℓ_a. We use $\hat{\alpha}_{\min}(a)$ to refer to the minimal sensitivity of a player that uses an arc in $L(a)$, i.e., $\hat{\alpha}_{\min}(a) = \min\{\alpha_i \mid A_i^+ \cap L(a) \neq \emptyset\}$; we adopt the convention that $\hat{\alpha}_{\min}(a) = \infty$ if no such player exists.

Algorithm 1 describes the computation of a toll vector τ inducing f. We first establish some properties of this algorithm. It will turn to be convenient to view the algorithm as operating in *phases*, where a phase corresponds to the execution of the outer for-loop (Lines 2–5) for a fixed $a \in \{2, \dots, m^+\}$.

▶ **Lemma 2.** *Let τ be the toll vector computed by Algorithm 1. Then for every two arcs $\hat{a}, \check{a} \in A$ with $\hat{a} \ge \check{a}$ it holds $\tau_{\check{a}} - \tau_{\hat{a}} = \sum_{a=\check{a}+1}^{\hat{a}} \frac{\ell_a - \ell_{a-1}}{\hat{\alpha}_{\min}(a)}$.*

Proof. Note that $\tau_{\check{a}}$ and $\tau_{\hat{a}}$ remain zero during phases $a = 2, \dots, \check{a}$ and are increased by the same amount Δ_a in phases $a = \hat{a}+1, \dots, m^+$. In phase $a \in \{\check{a}+1, \dots, \hat{a}\}$, $\tau_{\check{a}}$ is increased by Δ_a, while $\tau_{\hat{a}}$ remains zero. The claim follows from the definition of $\Delta_a = \frac{\ell_a - \ell_{a-1}}{\hat{\alpha}_{\min}(a)}$. ◀

The following theorem gives a characterization of flows that are inducible by *unrestricted* tolls.

▶ **Theorem 3.** *A flow $f = (f^i)_{i\in[k]}$ of non-atomic, heterogeneous players with sensitivities $0 < \alpha_1 \le \alpha_2 \le \cdots \le \alpha_m$ is inducible by unrestricted tolls if and only if*

$$\forall i, j \in [k] \text{ with } \alpha_i > \alpha_j,\ \forall a \in A_i^+,\ \forall \bar{a} \in A_j^+ :\ \ell_a \ge \ell_{\bar{a}}. \tag{3}$$

Intuitively, the above condition states that a flow is inducible if more sensitive players are routed on arcs with larger latencies.

Proof. Assume for the sake of contradiction that f is inducible and (3) does not hold, i.e., there exist $i, j \in [k]$ with $\alpha_i > \alpha_j$ and arcs $a \in A_i^+$, $\bar{a} \in A_j^+$ such that $\ell_a < \ell_{\bar{a}}$. Note that f is a Nash flow with respect to the combined costs because it is inducible. Since $a \in A_i^+$ and $\bar{a} \in A_j^+$, we have

$$\ell_a + \alpha_i \tau_a \le \ell_{\bar{a}} + \alpha_i \tau_{\bar{a}} \quad \text{or, equivalently,} \quad \ell_{\bar{a}} - \ell_a \ge \alpha_i (\tau_a - \tau_{\bar{a}}). \tag{4}$$

$$\ell_{\bar{a}} + \alpha_j \tau_{\bar{a}} \leq \ell_a + \alpha_j \tau_a \quad \text{or, equivalently,} \quad \ell_{\bar{a}} - \ell_a \leq \alpha_j (\tau_a - \tau_{\bar{a}}). \tag{5}$$

Note that $\ell_{\bar{a}} - \ell_a > 0$ by assumption and $\alpha_j > 0$. Inequality (5) therefore implies that $\tau_a - \tau_{\bar{a}} > 0$. Combining (4) and (5), we obtain $\alpha_j (\tau_a - \tau_{\bar{a}}) \geq \alpha_i (\tau_a - \tau_{\bar{a}})$. Dividing both sides by $\tau_a - \tau_{\bar{a}} > 0$ leads to a contradiction because $\alpha_j < \alpha_i$ by assumption.

Now suppose that (3) holds. We show that f is inducible by the tolls τ computed by Algorithm 1. Assume for the sake of contradiction that f is not a Nash flow with respect to the combined cost. Then for some player i there exist arcs $a \in A_i^+$ and $\bar{a} \in A$ satisfying

$$\ell_a + \alpha_i \tau_a > \ell_{\bar{a}} + \alpha_i \tau_{\bar{a}} \quad \text{or, equivalently,} \quad \ell_a - \ell_{\bar{a}} > \alpha_i(\tau_{\bar{a}} - \tau_a). \tag{6}$$

Let $\check{a} = \min\{a, \bar{a}\}$ and $\hat{a} = \max\{a, \bar{a}\}$. By Lemma 2, the difference in toll is

$$\tau_{\check{a}} - \tau_{\hat{a}} = \sum_{a'=\check{a}+1}^{\hat{a}} \frac{\ell_{a'} - \ell_{a'-1}}{\hat{\alpha}_{\min}(a')}. \tag{7}$$

Recall that $\hat{\alpha}_{\min}(a') = \min\{\alpha_j \mid A_j^+ \cap L(a') \neq \emptyset\}$. We distinguish two cases:

Case 1: $\bar{a} \leq a$. We have $\hat{a} = a \in A_i^+$ and thus $a \in L(a')$ for every $a' \in \{\check{a}+1, \dots, \hat{a}\}$. As a consequence, $\hat{\alpha}_{\min}(a') \leq \alpha_i$. Now (7) implies that $\alpha_i(\tau_{\bar{a}} - \tau_a) \geq \ell_a - \ell_{\bar{a}}$, which is a contradiction to (6).

Case 2: $\bar{a} > a$. We have $\check{a} = a \in A_i^+$. Note that by assumption all arcs in $A_j^+ \cap L(a')$ satisfy $\alpha_j \geq \alpha_i$ for every $a' \in \{\check{a}+1, \dots, \hat{a}\}$. Thus, $\hat{\alpha}_{\min}(a') \geq \alpha_i$ and (7) implies that $\alpha_i(\tau_a - \tau_{\bar{a}}) \leq \ell_{\bar{a}} - \ell_a$, which is a contradiction to (6). ◀

The following lemma is crucial in order to obtain our characterization of flows that are inducible by θ-restricted tolls.

▶ **Lemma 4.** *Let f be an inducible flow. Then the tolls τ computed by Algorithm 1 are component-wise minimal tolls that induce f.*

Proof. Assume for the sake of contradiction that τ is not component-wise minimal, i.e., there exists a toll vector τ' which induces f and $\tau'_a < \tau_a$ for some arc $a \in A$. Choose $\bar{a} \in A$ as the arc with largest latency such that $\tau'_{\bar{a}} < \tau_{\bar{a}}$. Note that the toll that Algorithm 1 imposes on arc $m^+ \in A^+$ is $\tau_{m^+} = 0$. Because $\tau_{\bar{a}} > 0$ there must exist at least one arc in A^+ whose latency is strictly larger than $\ell_{\bar{a}}$. Let $\hat{a} \in A^+$ be an arc which has minimal latency among such arcs, i.e.,

$$\hat{a} = \arg\min\{\ell_a \mid a \in A^+,\ \ell_a > \ell_{\bar{a}}\}.$$

Let i be the least toll sensitive player that uses arc $\hat{a}$. By Lemma 2, $\tau_{\bar{a}} - \tau_{\hat{a}} = \frac{\ell_{\hat{a}} - \ell_{\bar{a}}}{\alpha_i}$. Note that for player i the combined costs of $\hat{a}$ and $\bar{a}$ are equal because

$$\phi^i_{\hat{a}}(\tau) = \ell_{\hat{a}} + \alpha_i \tau_{\hat{a}} = \ell_{\bar{a}} + \alpha_i \tau_{\hat{a}} + \alpha_i \left(\frac{\ell_{\hat{a}} - \ell_{\bar{a}}}{\alpha_i} \right) = \ell_{\bar{a}} + \alpha_i \tau_{\bar{a}} = \phi^i_{\bar{a}}(\tau).$$

Further, $\phi^i_{\bar{a}}(\tau') < \phi^i_{\bar{a}}(\tau)$ because $\tau'_{\bar{a}} < \tau_{\bar{a}}$. Because f is a Nash flow with respect to τ' it follows that $\phi^i_{\hat{a}}(\tau') \leq \phi^i_{\bar{a}}(\tau')$. Therefore, $\phi^i_{\hat{a}}(\tau') \leq \phi^i_{\bar{a}}(\tau') < \phi^i_{\bar{a}}(\tau) = \phi^i_{\hat{a}}(\tau)$. This implies that $\tau'_{\hat{a}} < \tau_{\hat{a}}$ which is a contradiction to the choice of $\bar{a}$. ◀

▶ **Theorem 5.** *A flow $f = (f^i)_{i \in [k]}$ for non-atomic, heterogeneous players with sensitivities $0 < \alpha_1 \leq \alpha_2 \leq \dots \leq \alpha_m$ is inducible by θ-restricted tolls if and only if*

1. *$\forall i, j \in [k]$ with $\alpha_i > \alpha_j$, $\forall a \in A_i^+$, $\forall \bar{a} \in A_j^+$: $\ell_a \geq \ell_{\bar{a}}$.*
2. *$\forall a \in A$: $\theta_a \geq \tau_a$, where τ is the toll vector computed by Algorithm 1 for f.*

Proof. Let f be inducible by θ-restricted tolls τ'. Clearly, f is also inducible by unrestricted tolls and the first condition therefore follows by Theorem 3. Let τ be the toll vector computed by Algorithm 1. Then f is also inducible by τ. By Lemma 4, we have $\tau_a \leq \tau'_a$ for every arc $a \in A$ because τ is component-wise minimal. Since τ' is θ-restricted we conclude that $\tau_a \leq \tau'_a \leq \theta_a$ for every arc $a \in A$, which proves the second condition.

Next, suppose that Conditions 1 and 2 hold. Then f is inducible by unrestricted tolls by Theorem 3. In particular, the proof of Theorem 3 shows that the tolls τ computed by Algorithm 1 induce f. Condition 2 now establishes that τ is θ-restricted, which concludes the proof. ◀

3.2 Taxing Subnetworks

Bonifaci, Salek and Schäfer [2] derive a polynomial-time algorithm for computing optimal θ-restricted tolls for homogeneous, non-atomic players on parallel-arc networks with affine latency functions. We show that this algorithm can be used to determine optimal tolls for taxing subnetworks in the presence of *heterogeneous* players.

Suppose we are given an instance of the taxing subnetworks problem with affine latency functions and k player classes with sensitivities $(\alpha_i)_{i\in[k]}$ and demands $(r_i)_{i\in[k]}$. Let $T \subseteq A$ and $N := A \setminus T$ be the sets taxable and non-taxable arcs, respectively. Set $r := \sum_{i\in[k]} r_i$ and $L = \max_{a\in A} \ell_a(r) - \min_{a\in A} \ell_a(0)$. Note that for homogeneous players and parallel-arc networks the maximum toll needed to induce a given flow is at most L. We can therefore define an instance of the θ-restricted toll problem with $\theta_a = L$ for every $a \in T$ and $\theta_a = 0$ for every $a \in N$. Now, compute an optimal θ-restricted toll vector τ for demand r by running the algorithm in [2]. Let $f^\tau = (f^\tau_a)_{a\in A}$ be the resulting Nash flow. The idea now is to turn the arc flow $f^\tau = (f^\tau_a)_{a\in A}$ into a player flow $f = (f^i)_{i\in[k]}$ such that the properties of Theorem 5 are satisfied. To this aim, we decompose f^τ into k player flows $f = (f^i)_{i\in[k]}$ in such a way that more sensitive players are assigned flow from higher latency arcs. We call this the *canonical decomposition* of f^τ.

▶ **Theorem 6.** *Let $f = (f^i)_{i\in[k]}$ be the canonical decomposition of the flow f^τ as described above. Then f is an optimal θ-restricted flow.*

Proof. Because f^τ is inducible by θ-restricted tolls for homogeneous players, it holds that for every $a \in N$ and $\bar{a} \in A^+$, $\ell_a(f_a) \geq \ell_{\bar{a}}(f_{\bar{a}})$. In particular, all the arcs in $N^+ := N \cap A^+$ have equal latencies and the latencies of arcs in $N \setminus N^+$ are at least as large. Now, the canonical decomposition guarantees that the resulting flow f satisfies (3) of Theorem 3. We can thus use Algorithm 1 to generate tolls which induce f for heterogenous players (as in the proof of Theorem 3). Further, these tolls will not impose any tolls on N because there is no flow-carrying arc with a latency larger than the one in N^+. Since $f^\tau_a = \sum_{i\in[k]} f^i_a$ for every $a \in A$, the total cost is not altered by this decomposition.

We next prove optimality. Let $f = (f_i)_{i\in[k]}$ be an optimal flow inducible by θ-restricted tolls for the game with heterogenous players. Consider the arc flow defined as $f_a := \sum_{i\in[k]} f^i_a$. As before, we know that for every $a \in N$ and $\bar{a} \in A^+$, $\ell_a(f_a) \geq \ell_{\bar{a}}(f_{\bar{a}})$. If $N \cap A^+ \neq \emptyset$ then the maximum latency arc that is used must be in N and all arcs in N have latencies at least as large as this arc. Otherwise, $N \cap A^+ = \emptyset$. In either case, we do not need to impose any toll on the arcs in N to induce f. We conclude that f is also inducible by θ-restricted tolls for homogeneous players. Thus, the total cost of an optimal flow inducible by θ-restricted tolls for homogenous players is at most the cost of one for heterogeneous players. ◀

4 Atomic Players

We turn to the problem of computing optimal θ-restricted tolls for parallel-arc networks with atomic players. Roughgarden [12] proved that it is NP-hard to compute an optimal flow for weighted atomic players in parallel-arc networks with linear latency functions. As a consequence, computing optimal tolls is NP-hard in this setting, even for homogeneous players and without restrictions on the tolls. We therefore assume that the players are homogeneous and unweighted.

4.1 Characterization of flows inducible by θrestricted tolls

We first derive a characterization of inducible flows for unweighted homogeneous players on parallel arc networks.

▶ **Lemma 7.** *A flow f of unweighted homomogeneous players on a parallel arc network $(\{s,t\},A)$ is inducible by θ-restricted tolls if and only if*

$$\ell_a(f_a+1)+\theta_a \geq \ell_{\hat{a}}(f_{\hat{a}}) \quad \forall a \in A, \tag{8}$$

where $\hat{a} := \arg\max_{a\in A^+} \ell_a(f_a)$.

Proof. Suppose that the restriction in Equation (8) holds. Then the tolls τ defined by $\tau_a = \max\{0, \ell_{\hat{a}}(f_{\hat{a}}) - \ell_a(f_a+1)\}$ are clearly non-negative and θ-restricted. Furthermore, for any $a \in A^+$ and $\bar{a} \in A$,

$$\begin{aligned}\ell_a(f_a)+\tau_a &\leq \max\{\ell_a(f_a), \ell_a(f_a)+\ell_{\hat{a}}(f_{\hat{a}})-\ell_a(f_a+1)\}\\ &\leq \max\{\ell_a(f_a), \ell_{\hat{a}}(f_{\hat{a}})\} = \ell_{\hat{a}}(f_{\hat{a}}) \leq \ell_{\bar{a}}(f_{\bar{a}}+1)+\tau_{\bar{a}}.\end{aligned}$$

This means no player has an incentive to change its choice. Suppose Equation (8) is not satisfied. Then there is an arc $a \in A$ such that $\ell_a(f_a+1)+\theta_a < \ell_{\hat{a}}(f_{\hat{a}})$. Then for any θ-restricted toll vector τ, $\ell_a(f_a+1)+\tau_a < \ell_{\hat{a}}(f_{\hat{a}})+\tau_{\hat{a}}$. A player on arc $\hat{a}$ will therefore want to switch to arc a, which means that the flow is not inducible by θ-restricted tolls. ◀

4.2 Optimal θRestricted Tolls on Parallel-Arc Networks

We exploit the above characterization to obtain an optimal algorithm to compute θ-restricted tolls for unweighted homogeneous players. The idea of the algorithm is to first guess the arc $\hat{a} \in A$ which is the maximum latency flow-carrying arc in a optimal solution and then the amount $i \in [k]$ of flow on it. We then compute for every other arc $a \in A \setminus \{\hat{a}\}$ the minimum required flow such that Equation (8) is satisfied; call this flow the *inducible basis* f^0. Every flow that contains the inducible basis f^0 is guaranteed to be inducible by θ-restricted tolls. We can thus disregard the restrictions and complete f^0 in the optimal way (as described in Algorithm 2).

▶ **Lemma 8.** *Algorithm 2 returns a flow which is cost minimal among all flows which contain the inducible basis f^0, ship d extra units and for which the latencies of used arcs do not exceed L, if such a flow exists.*

Proof. Suppose such a flow exists. We show that throughout the algorithm there exists an optimal solution f^* which has at least as much flow on each arc as the current flow f. Consider the ith iteration of the algorithm. In this iteration the algorithm increases the flow on $\bar{a}$ by one. (Note that because an optimal solution exists, such an arc always exists.) By

Algorithm 2: Algorithm for optimal completion of inducible basis f^0

Input: network $(\{s,t\}, A, \ell)$, flow f^0, demand d and latency cap L
Output: optimal flow $f \geq f^0$ such that $\ell_a(f_a) \leq L$, if it exists

$f \leftarrow f^0$
while $d > 0$ **do**
  $\bar{A} \leftarrow \{a \in A \mid \ell_a(f_a + 1) \leq L\}$
  if $\bar{A} = \emptyset$ **then return** failure
  $\bar{a} \leftarrow \arg\min_{a \in \bar{A}} (f_a + 1)\ell_a(f_a + 1) - f_a \ell_a(f_a)$
  $f_{\bar{a}} \leftarrow f_{\bar{a}} + 1$
  $d \leftarrow d - 1$
end

Algorithm 3: Algorithm for finding an optimal flow inducible by θ-restricted tolls

Input: network $(\{s,t\}, A, \ell, \theta)$ and player count k
Output: optimal flow f^* inducible by θ-restricted tolls

$f^* \leftarrow (k, \ldots, k)$
for $\hat{a} \in A$ **do**
  for $i \in [k]$ **do**
    $f^0_{\hat{a}} \leftarrow i$
    $d \leftarrow k - i$
    for $a \in A \setminus \{\hat{a}\}$ **do**
      $f^0_a \leftarrow \min\{n \in [k] \mid \ell_a(n + 1) + \theta_a \geq \ell_{\hat{a}}(i)\}$
      $d \leftarrow d - f^0_a$
    end
    if $d \geq 0$ *and* $\max_{a \in A^+} \ell_a(f^0_a) \leq \ell_{\hat{a}}(i)$ **then**
      $L \leftarrow \ell_{\hat{a}}(i)$
      $f \leftarrow$ optimal completion of f^0 (using Algorithm 2)
      if *flow completion succeeded and* $C(f) < C(f^*)$ **then** $f^* \leftarrow f$
    end
  end
end

our definition of $\bar{A}$, the resulting latency is at most L. Now suppose there is no optimal solution f^* such that $f^*_{\bar{a}} \geq f_{\bar{a}}$. Then after iteration $i - 1$, there exists an optimal solution f^* which 'agreed' with the algorithm's flow and an arc a such that $f^*_a > f_a$. Then in this flow f^*, if we move one unit of flow from a to $\bar{a}$, the change in social cost would be

$$(f^*_{\bar{a}} + 1)\ell_{\bar{a}}(f^*_{\bar{a}} + 1) - f^*_{\bar{a}}\ell_{\bar{a}}(f^*_{\bar{a}}) + (f^*_a - 1)\ell_a(f^*_a - 1) - f^*_a \ell_a(f^*_a). \tag{9}$$

By the choice of $\bar{a}$ and the convexity of $x \cdot \ell_a(x)$,

$$\begin{aligned}(f^*_{\bar{a}} + 1)\ell_{\bar{a}}(f^*_{\bar{a}} + 1) - f^*_{\bar{a}}\ell_{\bar{a}}(f^*_{\bar{a}}) &\leq (f^*_{\bar{a}} + 2)\ell_{\bar{a}}(f^*_{\bar{a}} + 2) - (f^*_{\bar{a}} + 1)\ell_{\bar{a}}(f^*_{\bar{a}} + 1) \\ &= (f_{\bar{a}} + 1)f_{\bar{a}}(f_{\bar{a}} + 1) - f_{\bar{a}}\ell_{\bar{a}}(f_{\bar{a}}) \\ &\leq (f_a + 1)f_a(f_a + 1) - f_a\ell_a(f_a) \\ &\leq f^*_a \ell_a(f^*_a) - (f^*_a - 1)\ell_a(f^*_a - 1).\end{aligned}$$

So the change in the social cost (9) is non-positive. Therefore the optimal solution f^* can be altered without increasing the social cost so that it 'agrees' again with f, a contradiction. ◀

We summarize the algorithm for finding an optimal flow inducible by θ-restricted tolls in Algorithm 3.

▶ **Theorem 9.** *Algorithm 3 finds in polynomial time a flow of minimum total cost which is inducible by θ-restricted tolls.*

Proof. Atomic network routing games with unweighted atomic players admit at least one Nash equilibrium. Since the number of feasible flows is finite, there exists an optimal solution f^*. Let $\hat{a}$ be its maximum latency arc and i the flow on this arc. Consider the iteration of the algorithm with the same choice of $\hat{a}$ and i. The algorithm then puts as much flow on every arc to ensure that it can stand its toll. Because f^* is inducible under θ-restrictions, it must ship at least as much flow on every arc. After this, the algorithm finds a cost-minimal flow of d units on the arcs $A \setminus \{\hat{a}\}$ with the added restriction that $\hat{a}$ remains the maximum-latency arc. Because of this restriction, the resulting flow remains inducible under θ-restrictions. It does this by increasing flow on the arc in such a way that the increase in social cost is minimized. This produces an optimal solution by Lemma 8. It is not hard to see that the algorithm never outputs a flow which is not inducible by θ-restricted tolls. It puts as much flow on every arc a such that it can stand its toll $\ell_{\hat{a}}(f_{\hat{a}}) - \ell_a(f_a + 1)$. If this requires more flow than there is demand, the flow is discarded and will never be returned. ◀

▶ Remark. Note that once an optimal flow inducible by θ-restricted tolls is found we can extract the respective tolls as described in the proof of Lemma 7.

4.3 Optimally Taxing Subnetworks with Heterogeneous Players

With the help of Algorithm 3, we can also compute an optimal solution to the taxing subnetworks problem on parallel-arc networks with heterogeneous players. We compute this optimal flow in polynomial time by the following steps: Run Algorithm 3 on the given network with k players and θ-restrictions as given. This returns a flow $(f_a)_{a \in A}$. We decompose this arc flow into a player flow $(f^i)_{i \in [k]}$ by assigning the most sensitive players to the arcs in N arbitrarily and the remaining players to the arcs in T using the canonical decomposition described in Section 3.2.

▶ **Theorem 10.** *The process described above generates an optimal θ-restricted flow.*

Proof. First we show that the flow is inducible by tolls on T. Run Algorithm 3 on the network $(\{s,t\}, T)$ to define tolls on arcs T. Since the canonical decomposition decomposes the flow on T, these tolls discourage players on T to change to a different arc in T. Furthermore, the maximum latency arc $\hat{a}$ in T^+ has a zero toll. Now note that for the flow f to be inducible in the homogeneous case, for every $a \in N, \bar{a} \in A^+$ it must hold that $\ell_a(f_a + 1) \geq \ell_{\bar{a}}(f_{\bar{a}})$. Since players on T have no incentive to switch to $\hat{a}$, they surely do not have any incentive to switch to an arc in N. Now consider the players on N. They would not change to another arc in N, or f would not be in equilibrium. Let a' be the maximum latency arc in A^+. Suppose $a' \in N$. We impose an additional toll to all arcs in T of $\max\{\ell_{a'}(f_{a'}) - \ell_{\hat{a}}(f_{\hat{a}})/\bar{\alpha}, 0\}$, where $\bar{\alpha} := \min_{i:a(i) \in N} \alpha_i$ denotes the minimum sensitivity amongst players on N. Then the cost that a player on N sees on $\hat{a}$ is at least

$$\ell_{\hat{a}}(f_{\hat{a}}) + \bar{\alpha} \cdot \frac{\ell_{a'}(f_{a'}) - \ell_{\hat{a}}(f_{\hat{a}})}{\bar{\alpha}} = \ell_{a'}(f_{a'}).$$

Since the players on $\hat{a}$ have no incentive to change to other arcs in T, neither do players in N, as their sensitivity to toll is at least as high. Because this extra toll is added to all arcs in T, the players on T are still in equilibrium. Consider an arbitrary user i that uses an arc in T. Then on arc $\hat{a}$ it sees cost of at most

$$\alpha_i \cdot \frac{\ell_{a'}(f_{a'}) - \ell_{\hat{a}}(f_{\hat{a}})}{\bar{\alpha}} + \ell_{\hat{a}}(f_{\hat{a}}) \leq \ell_{a'}(f_{a'}) - \ell_{\hat{a}}(f_{\hat{a}}) + \ell_{\hat{a}}(f_{\hat{a}}) = \ell_{a'}(f_{a'}),$$

so no player on T has an incentive to switch to N. Now let the maximum latency arc $a' \in T$. Then $\hat{a} = a'$. Because the players on T cannot gain by deviating to a different arc in T, for some player $i \in [k]$ such that $a(i) = \hat{a}$,

$$\alpha_i \cdot \tau_a + \ell_a(f_a) \geq \ell_{\hat{a}}(f_{\hat{a}}) \qquad \forall a \in T.$$

Since $\bar{\alpha} > \alpha_i$, for every user on some $\bar{a} \in N$

$$\bar{\alpha} \cdot \tau_a + \ell_a(f_a) \geq \alpha_i \cdot \tau_a + \ell_a(f_a) \geq \ell_{\hat{a}}(f_{\hat{a}}) \geq \ell_{\bar{a}}(f_{\bar{a}}) \qquad \forall a \in T.$$

So the decomposed flow is inducible by tolls.

It remains to prove optimality. Take an optimal solution to the heterogeneous variant of the problem. Again we know that for every $a \in N, \bar{a} \in A^+$ it must hold that $\ell_a(f_a+1) \geq \ell_{\bar{a}}(f_{\bar{a}})$, or it can never be induced by tolls which are zero on N. For arc flow f' defined by $f'_a := \sum_{i=1}^{k} f^i_a$, the component-wise minimal tolls that induce these are equal to $\max\{0, \ell_{\hat{a}}(f_{\hat{a}}) - \ell_a(f_a + 1)\}$ for $a \in A$. Then for $a \in N$, $\tau_a = 0$. So the minimum cost solution to the heterogeneous problem is at most that of the homogeneous variant, which proves optimality. ◀

References

1 M. Beckmann, B. McGuire, and C. Winsten. *Studies in the Economics of Transportation*. Yale University Press, New Haven, 1956.

2 V. Bonifaci, M. Salek, and G. Schäfer. Efficiency of restricted tolls in non-atomic network routing games. In *Proc. 4th Symp. on Algorithmic Game Theory*, pages 302–313, 2011.

3 I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. Taxes for linear atomic congestion games. *ACM Transactions on Algorithms*, 7(1):1–31, 2010.

4 R. Cole, Y. Dodis, and T. Roughgarden. Pricing network edges for heterogeneous selfish users. In *Proc. 35th Symp. on Theory of Computing*, pages 521–530, 2003.

5 L. Fleischer. Linear tolls suffice: New bounds and algorithms for tolls in single source networks. *Theoretical Computer Science*, 348(2-3):217–225, 2005.

6 L. Fleischer, K. Jain, and M. Mahdian. Tolls for heterogeneous selfish users in multicommodity networks and generalized congestion games. In *Proc. 45th Symp. on Foundations of Computer Science*, pages 277–285, 2004.

7 D. Fotakis and P. Spirakis. Cost-balancing tolls for atomic network congestion games. In *Proc. 3rd Workshop on Internet and Network Economics*, pages 179–190, 2007.

8 M. Hoefer, L. Olbrich, and A. Skopalik. Taxing subnetworks. In *Proc. 4th Workshop on Internet and Network Economics*, pages 286–294, 2008.

9 G. Karakostas and S. G. Kolliopoulos. Edge pricing of multicommodity networks for heterogeneous selfish users. In *Proc. 45th Symp. on Foundations of Computer Science*, pages 268–276, 2004.

10 I. Kleinert, M. Klimm, T. Harks, and R. H. Möhring. Computing network tolls with support constraints. Networks, to appear.

11 T. Roughgarden. The price of anarchy is independent of the network topology. *Computer and Systems Sciences*, 67(2):341–364, 2003.

12 T. Roughgarden. On the severity of Braess's paradox: Designing networks for selfish users is hard. *Computer and Systems Sciences*, 72(5):922–953, 2006.

13 D. Schrank, B. Eisele, and T. Lomax. TTI's 2012 urban mobility report, 2012. http://d2dtl5nnlpfr0r.cloudfront.net/tti.tamu.edu/documents/mobility-report-2012.pdf.

14 C. Swamy. The effectiveness of stackelberg strategies and tolls for network congestion games. *ACM Transactions on Algorithms*, 8(4):1–19, 2012.

15 H. Yang and H.-J. Huang. The multi-class, multi-criteria traffic network equilibrium and systems optimum problem. *Transportation Research B*, 38(1):1 – 15, 2004.

Approximation of smallest linear tree grammar*

Artur Jeż[†1] **and Markus Lohrey**[2]

1 MPI Informatik, Saarbrücken, Germany / University of Wrocław, Poland
2 University of Siegen, Germany

Abstract

A simple linear-time algorithm for constructing a linear context-free tree grammar of size $\mathcal{O}(r^2 g \log n)$ for a given input tree T of size n is presented, where g is the size of a minimal linear context-free tree grammar for T, and r is the maximal rank of symbols in T (which is a constant in many applications). This is the first example of a grammar-based tree compression algorithm with an approximation ratio polynomial in g. The analysis of the algorithm uses an extension of the recompression technique (used in the context of grammar-based string compression) from strings to trees.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases Grammar-based compression, Tree compression, Tree-grammars

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.445

1 Introduction

Grammar-based compression has emerged to an active field in string compression during the last 10 years. The principle idea is to represent a given string s by a small context-free grammar that generates only s; such a grammar is also called a *straight-line program* (SLP). For instance, the word $(ab)^{1024}$ can be represented by the SLP with the productions $A_0 \to ab$ and $A_i \to A_{i-1}A_{i-1}$ for $1 \leq i \leq 10$ (A_{10} is the start symbol). The size of this grammar is much smaller than the size (length) of the string $(ab)^{1024}$. In general, an SLP of size n (the size of an SLP is usually defined as the total length of all right-hand sides of productions) can produce a string of length $2^{\Omega(n)}$. Hence, an SLP can be seen indeed as a succinct representation of the generated word. The principle task of grammar-based string compression is to construct from a given input string s a small SLP that produces s. Unfortunately, finding a size-minimal SLP for a given input string is hard: Unless $\mathbb{P} = \mathbb{NP}$ there is no polynomial time grammar-based compressor, whose output SLP has size less than 8569/8568 times the size of a minimal SLP for the input string [4], and so there is no polynomial time grammar-based compressor $\mathcal{G}$ with an approximation ratio of less than 8569/8568. In general the approximation ratio for $\mathcal{G}$ is defined as the function $\alpha_{\mathcal{G}}$ with

$$\alpha_{\mathcal{G}}(n) = \max \frac{\text{size of the SLP produced by } \mathcal{G} \text{ with input } x}{\text{size of a minimal SLP for } x} \ ,$$

where the max is taken over all strings of length n (over an arbitrary alphabet). The best known polynomial time grammar-based compressors [4, 9, 17, 18] have an approximation ratio of $\mathcal{O}(\log(n/g))$, where g is the size of a smallest SLP for the input string (each of them works in linear time).

* The full version of this paper can be found at http://arxiv.org/abs/1309.4958.

† A. Jeż was partially supported by Polish National Science Centre (NCN) grant 2011/01/D/ST6/07164, 2011–2014.



© Artur Jeż Markus Lohrey;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 445–457



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



At this point, the reader might ask, what makes grammar-based compression so attractive. There are actually several reasons: The output of a grammar-based compressor (an SLP) is a clean and simple object, which may simplify the analysis of a compressor or the analysis of algorithms that work on compressed data (see [13] for a survey). Moreover, there are grammar-based compressors which achieve very good compression ratios. For example RePair [12] performs very well in practice and was for instance used for the compression of web graphs [5]. Finally, the idea of grammar-based string compression can be generalized to other data types as long as suitable grammar formalisms are known for them. The last point is the most important one for this work. In [3], grammar-based compression was generalized from strings to trees (a tree in this paper is always a rooted ordered tree over a ranked alphabet, i.e., every node is labelled with a symbol and the rank of this symbol is equal to the number of children of the node). For this, context-free tree grammars were used. Context free tree grammars that produce only a single tree are also known as straight-line context-free tree grammars (SLCF tree grammars). Several papers deal with algorithmic problems on trees that are succinctly represented by SLCF tree grammars, see [13] for a survey. In [14], RePair was generalized from strings to trees, and the resulting algorithm TreeRePair achieved excellent results on real XML data trees. Other grammar-based tree compressors were developed in [15]. But none of these compressors has an approximation ratio polynomial in g: For instance, in [14] a series of trees is constructed, where the n-th tree t_n has size $\Theta(n)$, there exists an SLCF tree grammar for t_n of size $\mathcal{O}(\log n)$, but the grammar produced by TreeRePair for t_n has size $\Omega(n)$ (similar examples can be constructed for the compressors in [3, 15]).

In this paper, we give the first example of a grammar-based tree compressor, called TtoG, with an approximation ratio of $\mathcal{O}(\log n)$ assuming the maximal rank r of symbols is bounded; otherwise the approximation ratio becomes $\mathcal{O}(r^2 \log n)$. TtoG is based on the work [9] of the first author, where grammar-based string compressor with an approximation ratio of $\mathcal{O}(\log n)$ is presented. The crucial fact about this compressor is that in contrast to [4, 17, 18] it does not use the LZ77 factorization of a string (which makes the compressors from [4, 17, 18] not suitable for a generalization to trees, since LZ77 ignores the tree structure and no analogue of LZ77 for trees is known), but is based on the *recompression technique*. This technique was introduced in [7] and successfully applied for a variety of algorithmic problems for SLP-compressed strings [7, 8] and word equations [11, 10]. The basic idea is to compress a string using two operations: (i) block compressions, which replaces every maximal substring of the form a^ℓ for a letter a by a new symbol a_ℓ, and (ii) pair compression, which for a given partition $\Sigma_\ell \uplus \Sigma_r$ of the alphabet replaces every substring $ab \in \Sigma_\ell \Sigma_r$ by a new symbol c. It can be shown that the composition of block compression followed by pair compression (for a suitably chosen partition of the input letters) reduces the length of the string by a constant factor. Hence, the iteration of block compression followed by pair compression yields a string of length one after a logarithmic number of phases. By reversing the single compression steps, one obtains an SLP for the initial string. The term "recompression" refers to the fact, that for a given SLP $\mathbb{G}$, block compression and pair compression can be simulated on the SLP $\mathbb{G}$. More precisely, one can compute from $\mathbb{G}$ a new grammar $\mathbb{G}'$, which is not much larger than $\mathbb{G}$ such that $\mathbb{G}'$ produces the result of block compression (respectively, pair compression) applied to the string produced by $\mathbb{G}$. In [9], the recompression technique is used to bound the approximation ratio of the above compression algorithm based on block and pair compression.

In this work we generalize the recompression technique from strings to trees. The operations of block compression and pair compression can be directly applied to chains of

unary nodes (nodes having only a single child) in a tree. But clearly, these two operations alone cannot reduce the size of the initial tree by a constant factor. Hence we need a third compression operation that we call leaf compression. It merges all children of node that are leafs into the node; the new label of the node determines the old label, the sequence of labels of the children that are leaves, and their positions in the sequence of all children of the node. Then, one can show that a single phase, consisting of block compression (that we call chain compression), followed by pair compression (that we call unary pair compression), followed by leaf compression reduces the size of the initial tree by a constant factor. As for strings, we obtain an SLCF tree grammar for the input tree by basically reversing the sequence of compression operations. The recompression approach again yield an approximation ratio of $\mathcal{O}(\log n)$ for our compression algorithm, but the analysis is technically more subtle.

Related work on grammar-based tree compression. We already mentioned that grammar-based tree compressors were developed in [3, 14, 15], but none of these compressors has a good approximation ratio. Another grammar-based tree compressors was presented in [1]. It is based on the BISECTION algorithm for strings and has an approximation ratio of $\mathcal{O}(n^{5/6})$. But this algorithm used a different form of grammars (elementary ordered tree grammars) and it is not clear whether the results from [1] can be extended to SLCF tree grammars, or whether the good algorithmic results for SLCF-compressed trees [13] can be extended to elementary ordered tree grammars. Let us finally mention [2], where trees are compressed by so called top trees. These are another hierarchical representation of trees. Upper bounds on the size of top trees are derived and compared with the size of the minimal dag (directed acyclic graph). More precisely, it is shown in [2] that the size of the top tree is larger than the size of the minimal dag by a factor of $\mathcal{O}(\log n)$. Since dags can be seen as a special case of SLCF tree grammars, our main result is stronger.

Computational model. To achieve a linear running time we employ RadixSort, see [6, Section 8.3], to obtain a linear-time grouping of symbols. To this end some assumption on the computational model and form of the input are needed: we assume that numbers of $\mathcal{O}(\log n)$ bits (where n is the size of the input tree) can be manipulated in time $\mathcal{O}(1)$ and that the labels of the input tree come from an interval $[1, \ldots, n^c]$, where c is some constant.

1.1 Trees and SLCF tree grammars

Let us fix for every $i \geq 0$ a countably infinite set $\mathbb{F}_i$ of letters of rank i and let $\mathbb{F} = \bigcup_{i \geq 0} \mathbb{F}_i$ be their disjoint union. Symbols in $\mathbb{F}_0$ are called *constants*, while symbols in $\mathbb{F}_1$ are called *unary letters*. We also write $\operatorname{rank}(a) = i$ if $a \in \mathbb{F}_i$. A ranked alphabet F is a finite subset of $\mathbb{F}$. We also write F_i for $F \cap \mathbb{F}_i$ and $F_{\geq i}$ for $\bigcup_{j \geq i} F_i$. An *F-labelled tree* is a rooted, ordered tree whose nodes are labelled with elements from F, satisfying the condition that if a node v is labelled with a then it has exactly $\operatorname{rank}(a)$ children, which are linearly ordered (by the usual left-to-right order). We denote by $\mathcal{T}(F)$ the set of F-labelled trees. In the following we shall simply speak about trees when the ranked alphabet is clear from the context or unimportant. When useful, we identify an F-labelled tree with a term over F in the usual way. The size $|t|$ of the tree t is its number of nodes.

Fix a countable set $\mathbb{Y}$ with $\mathbb{Y} \cap \mathbb{F} = \emptyset$ of *(formal) parameters*, which are denoted by $y, y_1, y_2, \ldots$. For the purposes of building trees with parameters, we treat all parameters as constants, and so F-labelled trees with parameters from $Y \subseteq \mathbb{Y}$ (where Y is finite) are simply $(F \cup Y)$-labelled trees, where the rank of every $y \in Y$ is 0. However to stress the special role of parameters we write $\mathcal{T}(F, Y)$ for the set of F-labelled trees with parameters from Y. We

identify $\mathcal{T}(F)$ with $\mathcal{T}(F, \emptyset)$. In the following we talk about *trees with parameters* (or even trees) when the ranked alphabet and parameter set is clear from the context or unimportant. The idea of parameters is best understood when we represent trees as terms: For instance $f(y_1, a, y_2, y_1)$ with parameters y_1 and y_2 can be seen as a term with variables y_1, y_2 and we can instantiate those variables later on. A *pattern* (or *linear tree*) is a tree $t \in \mathcal{T}(F, Y)$, that contains for every $y \in Y$ at most one y-labelled node. Clearly, a tree without parameters is a pattern. All trees in this paper will be patterns, and we will not mention this assumption explicitly in the following.

When we talk of a *subtree* u of a tree t, we always mean a full subtree in the sense that for every node of u all descendents of that node in t belong to u as well. In contrast, a *subpattern* v of t is obtained from a subtree u of t by replacing some of the subtrees of u by pairwise different parameters. In this way we obtain a pattern $p(y_1, \ldots, y_n)$ and we say that (i) the subpattern v is an *occurrence* of the pattern $p(y_1, \ldots, y_n)$ in t and (ii) $p(y_1, \ldots, y_n)$ is the pattern corresponding to the subpattern v (this pattern is unique up to renaming of parameters). This later terminology applies also to subtrees, since a subtree is a subpattern as well. To make this notions clear, consider for instance the tree $f(a(b(c)), a(b(d)))$ with $f \in \mathbb{F}_2$, $a, b \in \mathbb{F}_1$ and $c, d \in \mathbb{F}_0$. It contains one occurrence of the pattern $a(b(c))$ and two occurrences of the pattern $a(b(y))$.

A *chain pattern* is a pattern of the form $a_1(a_2(\ldots(a_k(y))\ldots))$ with $a_1, a_2, \ldots, a_k \in \mathbb{F}_1$. We write $a_1 a_2 \cdots a_k$ for this pattern and treat it as a string (even though this string still needs an argument on its right to form a proper term). In particular, we write a^ℓ for the chain pattern consisting of ℓ many a-labelled nodes and we write vw (for chain patterns v and w) for what should be $v(w(y))$. A *chain* in a tree t is an occurrence of a chain pattern in t. A chain s in t is *maximal* if there is no chain s' in t with $s \subsetneq s'$. A 2-chain is a chain consisting of only two nodes (which, most of the time, will be labelled with different letters). For $a \in \mathbb{F}_1$, an a-maximal chain is a chain such that (i) all nodes are labelled with a and (ii) there is no chain s' in t such that $s \subsetneq s'$ and all nodes of s' are labelled with a too. Note that an a-maximal chain is not necessarily a maximal chain. Consider for instance the tree $baa(c)$. The unique occurrence of the chain pattern aa is an a-maximal chain, but is not maximal. The only maximal chain is the unique occurrence of the chain pattern baa.

For the further consideration, fix a countable infinite set $\mathbb{N}_i$ of symbols of rank i with $\mathbb{N}_i \cap \mathbb{N}_j = \emptyset$ for $i \neq j$. Let $\mathbb{N} = \bigcup_{i \geq 0} \mathbb{N}_i$. Furthermore, assume that $\mathbb{F} \cap \mathbb{N} = \emptyset$. Hence, every finite subset $N \subseteq \mathbb{N}$ is a ranked alphabet. A *linear context-free tree grammar* (there exist also non-linear CF tree grammars, which we do not need for our purpose) or short *linear CF tree grammar* is a tuple $\mathbb{G} = (N, F, P, S)$ such that $N \subseteq \mathbb{N}$ (resp., $F \subseteq \mathbb{F}$) is a finite set of *nonterminals* (resp., *terminals*), $S \in N$ is the *start nonterminal* of rank 0, and P (the set of *productions*) is a finite set of pairs (A, t) (for which we write $A \to t$), where $A \in N$ and $t \in \mathcal{T}(F \cup N, \{y_1, \ldots, y_{\mathsf{rank}(A)}\})$ is a pattern, which contains exactly one y_i-labelled node for each $1 \leq i \leq \mathrm{rank}(A)$. To stress the dependency of A on its parameters we sometimes write $A(y_1, \ldots, y_{\mathrm{rank}(A)}) \to t$ instead of $A \to t$. Without loss of generality we assume that every nonterminal $B \in N \setminus \{S\}$ occurs in the right-hand side t of some production $(A \to t) \in P$, see [16, Theorem 5]. The derivation relation $\Rightarrow_{\mathbb{G}}$ on $\mathcal{T}(F \cup N, Y)$ is defined as follows: $s \Rightarrow_{\mathbb{G}} s'$ if and only if there is a production $(A(y_1, \ldots, y_\ell) \to t) \in P$ such that s' is obtained from s by replacing some subtree $A(t_1, \ldots, t_\ell)$ of s by t with each y_i replaced by t_i. Intuitively, we replace an A-labelled node by the pattern $t(y_1 \ldots, y_{\mathrm{rank}(A)})$ and thereby identify the j-th child of A with the unique y_j-labelled node of the pattern. Then $L(\mathbb{G}) = \{t \in \mathcal{T}(F) \mid S \Rightarrow_{\mathbb{G}}^* t\}$.

A *straight-line context-free tree grammar* (or *SLCF grammar* for short) is a linear CF tree grammar $\mathbb{G} = (N, F, P, S)$, where (i) for every $A \in N$ there is *exactly one* production

$(A \to t) \in P$ with left-hand side A, (ii) if $(A \to t) \in P$ and B occurs in t then $B < A$, where $<$ is a linear order on N, and (iii) S is the maximal nonterminal with respect to $<$. By (i) and (ii), every $A \in N$ derives exactly one tree from $\mathcal{T}(F, \{y_1, \ldots, y_{\mathsf{rank}(A)}\})$; we denote this tree by $\mathrm{val}(A)$ (like *value*). Moreover, we define $\mathrm{val}(\mathbb{G}) = \mathrm{val}(S)$, which is a tree from $\mathcal{T}(F)$. For an SLCF grammar $\mathbb{G} = (N, F, P, S)$ we can assume without loss of generality that for every production $(A \to t) \in P$ the parameters $y_1, \ldots, y_{\mathrm{rank}(A)}$ occur in t in the order $y_1, y_2, \ldots, y_{\mathrm{rank}(A)}$ from left to right. This can be ensured by a simple bottom-up rearranging procedure.

There is a subtle point, when defining the *size* $|\mathbb{G}|$ of the SLCF grammar $\mathbb{G}$: One possible definition could be $|\mathbb{G}| = \sum_{(A \to t) \in P} |t|$, i.e., the sum of all sizes of right-hand sides. However, consider for instance the rule $A(y_1, \ldots, y_\ell) \to f(y_1, \ldots, y_{i-1}, a, y_i, \ldots, y_\ell)$. It is in fact enough to describe the right-hand side as $(f, (i, a))$, as we have a as the i-th child of f. On the remaining positions we just list the parameters, whose order is known; see the above remark. In general, each right-hand side can be specified by listing for each node its children that are *not* parameters together with their positions in the list of all children. These positions are numbers between 1 and r (it is easy to show that our algorithm TtoG creates only nonterminals of rank at most r, see Lemma 1, and hence every node in a right-hand side has at most r children) and therefore fit into $\mathcal{O}(1)$ machine words. For this reason we define the size $|\mathbb{G}|$ as the total number of non-parameter nodes in all right-hand sides. If the size of a grammar is defined as the total number of all nodes (including parameters) in all right-hand sides, then the approximation ratio of TtoG is multiplied by an additional factor r.

Notational conventions. Our compression algorithm TtoG takes a tree T and applies to it local compression operations, which shrink the size of the tree. With T we always denote the current tree stored by TtoG, whereas n denotes the size of the initial input tree. The algorithm TtoG adds fresh letters to the tree. With F we always denote the set of letters occurring in the current tree T. The ranks of the fresh letters do not exceed the maximal rank of the original letters. To be more precise, if we add a letter a to F_i, then $F_{\geq i}$ was non-empty before this addition. By r we denote the maximal rank of the letters occurring in the input tree. By the above remark, TtoG never introduces letters of rank larger than r.

2 Compression operations

Our compression algorithm TtoG is based on three local replacement rules applied to trees:

- **(i)** a-maximal chain compression: For a unary letter a replace every a-maximal chain consisting of $\ell > 1$ nodes with a fresh unary letter a_ℓ (for all $\ell > 1$).
- **(ii)** (a, b)-pair compression: For two unary letters $a \neq b$ replace every occurrence of ab by a single node labelled with a fresh unary letter c (which identifies the pair (a, b)).
- **(iii)** $(f, i_1, a_1 \ldots, i_\ell, a_\ell)$-leaf compression: For $f \in F_{\geq 1}$, $\ell \geq 1$, $a_1, \ldots, a_\ell \in F_0$ and $0 < i_1 < i_2 < \cdots < i_\ell \leq \mathrm{rank}(f) =: m$ replace every occurrence of $f(t_1, \ldots, t_m)$, where $t_{i_j} = a_j$ for $1 \leq j \leq \ell$ and t_i is a non-constant for $i \notin \{i_1, \ldots, i_\ell\}$, by $f'(t_1, \ldots, t_{i_1-1}, t_{i_1+1}, \ldots, t_{i_\ell-1}, t_{i_\ell+1}, \ldots, t_m)$, where f' is a fresh letter of rank $\mathrm{rank}(f) - \ell$ (which identifies $(f, i_1, a_1 \ldots, i_\ell, a_\ell)$).

Note that each of these operations shrinks the size of the current tree. Operations (i) and (ii) apply only to unary letters and are direct translations of the operations used in the recompression-based algorithm for constructing a grammar for a given string [9]. On the other hand, (iii) is a new and designed specifically to deal with trees.

Every application of one of our compression operations can be seen as the 'backtracking' of a production of the grammar that we construct: When we replace a^ℓ by a_ℓ, we introduce the new nonterminal $a_\ell(y)$ with the production $a_\ell(y) \to a^\ell(y)$. When we replace all occurrences of the chain ab by c, the new production is $c(y) \to a(b(y))$. Finally, for $(f, i_1, a_1 \ldots, i_\ell, a_\ell)$-leaf compression the production is $f'(y_1, \ldots, y_{\mathrm{rank}(f)-\ell}) \to f(t_1, \ldots, t_{\mathrm{rank}(f)})$, where $t_{i_j} = a_j$ for $1 \le j \le \ell$ and every t_i with $i \notin \{i_1, \ldots, i_\ell\}$ is a parameter (and the left-to-right order of the parameters in the right-hand side is $y_1, \ldots, y_{\mathrm{rank}(f)-\ell}$). All these productions are for nonterminals of rank at most r, which implies:

▶ **Lemma 1.** *The rank of nonterminals defined by* TtoG *is at most* r.

During the analysis of the approximation ratio of TtoG we also consider the nonterminals of a smallest grammar generating the given input tree. To avoid confusion between these nonterminals and the nonterminals of the grammar produced by TtoG, we insist on calling the fresh symbols introduced by TtoG (a_ℓ, c, and f' above) *letters* and add them to the set F of current letters, so that F always denotes the set of letters in the current tree T. In particular, whenever we talk about nonterminals, productions, etc. we mean the ones of the smallest grammar we consider. Nevertheless, the above productions for the new letters form the grammar returned by our algorithm TtoG and we need to estimate their size. In order not to mix the notation, we shall call the size of the rule for a new letter a the *representation cost* for a and say that a *represents* the subpattern it replaces in T. For instance, the representation cost of a_ℓ with $a_\ell(y) \to a^\ell(y)$ is ℓ, the representation cost of c with $c(y) \to a(b(y))$ is 2, and the representation cost of f' with $f'(y_1, \ldots, y_{\mathrm{rank}(f)-\ell}) \to f(t_1, \ldots, t_{\mathrm{rank}(f)})$ is $\ell + 1$. A crucial part of the analysis of TtoG is the reduction of the representation cost for letters a_ℓ: Note that instead of representing $a^\ell(y)$ directly by $a_\ell(y) \to a^\ell(y)$, we can introduce new unary letters representing some shorter chains in a^ℓ and build longer chains using the smaller ones as building blocks. For instance, the rule $a_8(y) \to a^8(y)$ can be replaced by the rules $a_8(y) \to a_4(a_4(y))$, $a_4(y) \to a_2(a_2(y))$ and $a_2(y) \to a(a(y))$. This yields a total representation cost of 6 instead of 8. Our algorithm employs a particular strategy for representing a-maximal chains, which yields the total cost stated in the following lemma:

▶ **Lemma 2** (cf. [9, Lemma 2])**.** *Given a list* $\ell_1 < \ell_2 < \cdots < \ell_k$ *we can represent the letters* $a_{\ell_1}, a_{\ell_2}, \ldots, a_{\ell_k}$ *that replace the chain patterns* $a^{\ell_1}, a^{\ell_2}, \ldots, a^{\ell_k}$ *with a total cost of* $\mathcal{O}(k + \sum_{i=1}^{k} \log(\ell_i - \ell_{i-1}))$*, where* $\ell_0 = 0$.

The important property of the compression operations is that we can perform many of them independently in an arbitrary order without influencing the outcome. Since different a-maximal chains and b-maximal chains do not overlap (regardless of whether $a = b$ or not) we can perform a-maximal chain compression for all unary letters a occurring in T in an arbitrary order (assuming that the new letters do not occur in T). We call the resulting tree ChainCmp(T), and denote the corresponding procedure also *chain compression*.

A similar observation applies to leaf compressions: We can perform $(f, i_1, a_1 \ldots, i_\ell, a_\ell)$-leaf compression for all $f \in F_{\ge 1}$, $0 < i_1 < i_2 < \cdots < i_\ell \le \mathrm{rank}(f) =: m$, and $(a_1, a_2, \ldots, a_\ell) \in F_0^\ell$ in an arbitrary order (again assuming that the fresh letters do not occur in the T). We denote the resulting tree with LeafCmp(T) and call the corresponding procedure also *leaf compression*.

The situation is more subtle for unary pair compression: observe that in a chain abc we can compress ab or bc but we cannot do both in parallel (and the outcome depends on the order of the operations). However, as in the case of string compression [9], independent (or parallel) (a, b)-pair compressions are possible when we take a and b from disjoint subalphabets

F_1^{up} and F_1^{down}, respectively. In this case for each unary letter we can tell whether it should be the parent node or the child node in the compression step and the result does not depend on the order of the considered 2-chains, as long as new letters are outside $F_1^{\text{up}} \cup F_1^{\text{down}}$. Hence, we denote with $\textsc{UnaryCmp}(F_1^{\text{up}}, F_1^{\text{down}}, T)$ the result of doing (a,b)-pair compression for all $a \in F_1^{\text{up}}$ and $b \in F_1^{\text{down}}$ (in an arbitrary order). The corresponding procedure is also called $(F_1^{\text{up}}, F_1^{\text{down}})$-*compression*.

3 The algorithm TtoG

In a single phase of the algorithm TtoG, chain compression, $(F_1^{\text{up}}, F_1^{\text{down}})$-compression and leaf compression are executed in this order (for an appropriate choice of the partition $F_1^{\text{up}}, F_1^{\text{down}}$).

Algorithm 1 TtoG: Creating an SLCF tree grammar for the input tree T

```
1: while |T| > 1 do
2:     T ← ChainCmp(T)
3:     compute a partition F_1 = F_1^up ⊎ F_1^down        ▷ Lemma 3
4:     T ← UnaryCmp(F_1^up, F_1^down, T)
5:     T ← LeafCmp(T)
6: return constructed grammar
```

The intuition behind this approach is as follows: If the tree t in question does not have any unary letters, then leaf compression on its own reduces the size of t by half, as it effectively reduces all constant nodes, i.e. leaves of the tree, and more than half of nodes are leaves. On the other end of the spectrum is the situation in which all nodes (except for the unique leaf) are labelled with unary letters. In this case our instance is in fact a string. Chain compression and unary pair compression correspond to the operations of block compression and pair compression, respectively, from the earlier work of the first author on string compression [9], where it is shown that block compression followed by pair compression reduces the size of the string by a constant factor 3/4 (for an appropriate choice of the partition $F_1^{\text{up}}, F_1^{\text{down}}$ of the letters occurring in the string). The in-between cases are a mix of those two extreme scenarios and for each of them the size of the instance drops by a constant factor in one phase as well, see Lemma 4. We need the following lemma, which is a modification of [9, Lemma 4]. Recall that F always denotes the set of letters occurring in T.

▶ **Lemma 3.** *Assume that (i) T does not contain an occurrence of a chain pattern aa for some $a \in F_1$ and (ii) the symbols in T form an interval of numbers. Then, in time $\mathcal{O}(|T|)$ one can find a partition $F_1 = F_1^{up} \uplus F_1^{down}$ such that the number of occurrences of chain patterns from $F_1^{up}F_1^{down}$ in T is at least $(n_1 - 3c + 2)/4$, where n_1 is the number of nodes in T with a unary label and c is the number of maximal chains in T. In the same running time we can provide for each $ab \in F_1^{up}F_1^{down}$ occurring in T a lists of pointers to all occurrences of ab in T.*

A single iteration of the main loop of TtoG is called a *phase*. A single phase can be implemented in time linear to the size of the current T. The main idea is that RadixSort is used for effective grouping in linear time and finding a partition is a simple modification of [9, Lemma 4]. The main property of a single phase is:

▶ **Lemma 4.** *In each phase, $|T|$ is reduced by a constant factor.*

Since each phase needs linear time, the contributions of all phase give a geometric series and we get:

▶ **Theorem 5.** TtoG *runs in linear time.*

4 Size of the grammar produced by TtoG: recompression

4.1 Normal form

We want to compare the size of the grammar produced by TtoG with the size of a smallest SLCF grammar for the input tree T. For this, we first transform the minimal grammar into a so called handle grammar and show that this increases the grammar size by a factor of $\mathcal{O}(r)$, where r is the maximal rank of symbols from $\mathbb{F}$ occurring in T. Then, we compare the size of a minimal handle grammar for T with the size of the output of TtoG.

A *handle* is a pattern $t(y) = f(w_1(\gamma_1), w_2(\gamma_2), \ldots, w_{i-1}(\gamma_{i-1}), y, w_{i+1}(\gamma_{i+1}), \ldots, w_\ell(\gamma_\ell))$, where $\operatorname{rank}(f) = \ell$, every γ_j is either a constant symbol or a nonterminal of rank 0, every w_j is a chain pattern, and y is a parameter. Note that $a(y)$ for a unary letter a is a handle. Since handles have one parameter only, for handles $h_1, h_2, \ldots, h_\ell$ we write $h_1 h_2 \cdots h_\ell$ for the tree $h_1(h_2(\ldots(h_\ell(y))))$ and treat it as a string, similarly to chains patterns. We say that an SLCF grammar $\mathbb{G}$ is a *handle grammar* (or simply "$\mathbb{G}$ is handle") if the following conditions hold:

(H1) $N \subseteq \mathbb{N}_0 \cup \mathbb{N}_1$

(H2) For $A \in N \cap \mathbb{N}_1$ the unique rule for A is of the form $A(y) \to u(B(v(C(w(y)))))$ or $A(y) \to u(B(v(y)))$ or $A(y) \to u(y)$, where u, v, and w are (perhaps empty) sequences of handles and $B, C \in N_1$. We call B the *first* and C the *second* nonterminal in the rule for A.

(H3) For $A \in N \cap \mathbb{N}_0$ the rule for A is of the (similar) form $A \to u(B(v(C)))$ or $A \to u(B(v(c)))$ or $A \to u(C)$ or $A \to u(c)$, where u and v are (perhaps empty) sequences of handles, c is a constant, $B \in N_1$, $C \in N_0$, and $j, k < i$. Again we speak of the first and second nonterminal in the rule for A.

Note that the representation of the rules for nonterminals from $\mathbb{N}_0$ is not unique. Take for instance the rule $A \to f(B, C)$, which can be written as $A \to a(C)$ for the handle $a(y) = f(B, y)$ or as $A \to b(B)$ for the handle $b(y) = f(y, C)$. For nonterminals from $\mathbb{N}_1$ this problem does not occur, since there is a unique occurrence of the parameter y in the right-hand side. For a given SLCF grammar we can find an equivalent handle grammar of similar size:

▶ **Lemma 6.** *Let $\mathbb{G}$ be an SLCF grammar. Then there exists a handle grammar $\mathbb{G}'$ such that $\operatorname{val}(\mathbb{G}') = \operatorname{val}(\mathbb{G})$ and $|\mathbb{G}'| = \mathcal{O}(r|\mathbb{G}|)$, where r is the maximal rank of the letters used in $\mathbb{G}$.*

For the proof one first applies the main result of [16] to make $\mathbb{G}$ monadic (i.e., $N \subseteq \mathbb{N}_0 \cup \mathbb{N}_1$). The resulting grammar can be easily transformed into a handle grammar by considering for each nonterminal $A \in N \cap \mathbb{N}_1$ the path from the root to the unique occurrence of the parameter in the right-hand side of A.

4.2 Intuition and invariants

For a given input tree T we start with a smallest handle grammar $\mathbb{G}$ generating T. In the following, by g we always denote the size of this initial minimal handle grammar. With each occurrence of a letter from $\mathbb{F}$ in $\mathbb{G}$'s rules we associate 2 *credits*. During the run of TtoG we appropriately modify $\mathbb{G}$, so that $\operatorname{val}(\mathbb{G}) = T$ (where T always denotes the current tree in TtoG). In other words, we perform the compression steps of TtoG also on $\mathbb{G}$. We always maintain the invariant that every occurrence of a letter from $\mathbb{F}$ in $\mathbb{G}$'s rules has two credits. To this end, we *issue* some new credits during the modifications, and we have to do a precise bookkeeping on the amount of issued credit. On the other hand, if we do a

compression step in $\mathbb{G}$, then we remove some occurrences of letters. The credit associated with these occurrences is then *released* and can be used to pay for the representation cost of the new letters introduced by the compression step. For unary pair compression and leaf compression, the released credit indeed suffices to pay the representation cost for the fresh letters, but for chain compression the released credit does not suffice. Here we need some extra amount that will be estimated separately. At the end, we bound the size of the grammar produced by TtoG as the sum of the initial credit assigned to $\mathbb{G}$ (at most $2g$) plus the total amount of issued credit plus the extra cost estimated in Section 4.6. We emphasize that the modification of $\mathbb{G}$ is not performed by TtoG, but is only a mental experiment done for the purpose of analyzing TtoG.

An important difference between our algorithm and the string compression algorithm from the earlier paper of the first author [9] is that we add new nonterminals to $\mathbb{G}$ during its modification. All such nonterminals will have rank 0 and we shall denote the set of such currently used nonterminals by $\widetilde{N_0}$. To simplify notation, we denote with m always the number of nonterminals of the current grammar $\mathbb{G}$, and we denote its nonterminals by $A_1, \ldots, A_m$. We assume that $i < j$ if A_i occurs in the right-hand side of A_j, and that A_m is the start nonterminal. With α_i we always denote the current right-hand side of A_i, i.e., the productions of $\mathbb{G}$ are $A_i \to \alpha_i$ for $1 \leq i \leq m$.

Suppose a compression step, for simplicity say an (a, b)-pair compression, is applied to T. We should also reflect it in $\mathbb{G}$. The simplest solution would be to perform the same compression on each of the rules of $\mathbb{G}$, hoping that in this way all occurrences of ab in $\mathrm{val}(\mathbb{G})$ will be replaced by c. However, this is not always the case. For instance, the 2-chain ab may occur 'between' a nonterminal and a unary letter: consider a grammar $A_1(y) \to a(y)$ and $A_2 \to A_1(b(c))$ and a 2-chain ab. Then it it occurs in $\mathrm{val}(A_2)$ but this occurrence is 'between' A_1 and b in the rule for A_2. This intuitions are made precise in Section 4.3. To deal with this problem, we modify the grammar, so that such bad cases no longer occur. Similar problems occur also when we want to replace an a-maximal chain or perform leaf compression. Solutions to those problems are similar and are given in Section 4.4 and Section 4.5, respectively.

To ensure that $\mathbb{G}$ is handle and to estimate the amount of issued credit, we show that the grammar preserves the following invariants, where n_0 (resp. n_1) is the initial number of nonterminals from N_0 (resp., N_1) in $\mathbb{G}$ and g is the initial size of $\mathbb{G}$.

(I1) $\mathbb{G}$ is handle.

(I2) $\mathbb{G}$ has nonterminals $N_0 \cup N_1 \cup \widetilde{N_0}$, where $\widetilde{N_0}, N_0 \subseteq \mathbb{N}_0$, $|N_0| \leq n_0$ and $N_1 \subseteq \mathbb{N}_1$, $|N_1| \leq n_1$.

(I3) The number of occurrences of nonterminals from N_0, N_1 and $\widetilde{N_0}$ in $\mathbb{G}$ are at most g, $n_0 + 2n_1$ and $(n_0 + 2n_1)(r-1)$, respectively

(I4) The rules for $A_i \in \widetilde{N_0}$ are of the form $A_i \to wA_j$ or $A_i \to wc$, where w is a string of unary symbols, $A_j \in N_0 \cup \widetilde{N_0}$ and c is a constant.

It is easy to show that (I1)–(I4) hold for the initial handle grammar $\mathbb{G}$ when we set $\widetilde{N_0} = \emptyset$. The only non-trivial condition is that the number of occurrences of nonterminals from N_1 is at most $n_0 + 2n_1$. However, in a rule for $A_i \in N_0$ there is at most one occurrence of a nonterminal from N_1, namely the first nonterminal in this rule (all other nonterminals are parts of handles and so they are from N_0). Similarly in a rule for $A_i \in N_1$ there are at most two occurrences of nonterminals from N_1.

4.3 $(F_1^{\mathsf{up}}, F_1^{\mathsf{down}})$-compression

We begin with some definitions that help to classify which 2-chains are easy and which hard to compress.

For a non-empty tree or pattern t its *first* letter is the letter that labels the root of t. For a pattern $t(y)$ which is not a parameter its *last* letter is the label of the node above the one labelled with y. A chain pattern ab has a *crossing occurrence* in a nonterminal A_i if one of the following holds:

(C1) $a(A_j)$ is a subpattern of α_i and the first letter of $\mathrm{val}(A_j)$ is b
(C2) $A_j(b)$ is a subpattern of α_i and the last letter of $\mathrm{val}(A_j)$ is a
(C3) $A_j(A_k)$ is a subpattern of α_i, the last letter of $\mathrm{val}(A_j)$ is a and the first letter of $\mathrm{val}(A_k)$ is b.

A chain pattern ab is *crossing* if it has a crossing occurrence in any nonterminal and *non-crossing* otherwise. Unless explicitly written, we use this notion only in case $a \neq b$.

When every chain pattern $ab \in F_1^{\mathrm{up}} F_1^{\mathrm{down}}$ is noncrossing, simulating $(F_1^{\mathrm{up}}, F_1^{\mathrm{down}})$-compression on $\mathbb{G}$ is easy: It is enough to apply $(F_1^{\mathrm{up}}, F_1^{\mathrm{down}})$-compression to each right-hand side of $\mathbb{G}$. We denote the resulting grammar with $\textsc{UnaryCmp}(\mathbb{G})$.

To distinguish between the nonterminals, grammar, etc. before and after the application of UnaryCmp (or, in general, any procedure) we use 'primed' symbols, i.e. A_i', $\mathbb{G}'$, T' for the nonterminals, grammar and tree, respectively, after the compression step and 'unprimed' symbols (i.e. A_i, $\mathbb{G}$, T) for the ones before.

Algorithm 2 $\textsc{Pop}(F_1^{\mathrm{up}}, F_1^{\mathrm{down}}, \mathbb{G})$

```
1: for i ← 1 .. m − 1 do
2:     if the first symbol of α_i is b ∈ F_1^down then
3:         if α_i = b then
4:             replace each A_i G rules by b
5:         else remove this leading b from α_i
6:             replace each A_i in G rules by bA_i
7:     do symmetric actions for the last symbol
```

It is left to assure that indeed all occurrences of chain patterns from $F_1^{\mathrm{up}} F_1^{\mathrm{down}}$ are noncrossing. Consider for instance the grammar with the rules $A_1(y) \to a(y)$ and $A_2 \to A_1(b(c))$. The pattern ab has a crossing occurrence. To deal with crossing occurrences we change the grammar. In our example, we replace A_1 with a, leaving only $A_2 \to ab(c)$, which does not contain a crossing occurrence of ab.

In general, suppose that some $ab \in F_1^{\mathrm{up}} F_1^{\mathrm{down}}$ is crossing because of (C1). Let $a(A_i)$ be a subpattern of some right-hand side and let $\mathrm{val}(A_i) = b(t')$. Then it is enough to modify the rule for A_i so that $\mathrm{val}(A_i) = t'$ and replace each occurrence of A_i in a right-hand side by $b(A_i)$. We call this action *popping-up b from A_i*. The similar operation of popping down a letter a from $A_i \in N \cap \mathbb{N}_1$ is symmetrically defined (note that both pop operations apply only to unary letters). By Lemma 7 below, popping up and down removes all crossing occurrences of ab. Note that the popping up and popping down can be performed for many letters in parallel: The procedure Pop (Algorithm 2) 'uncrosses' all occurrences of patterns from the set $F_1^{\mathrm{up}} F_1^{\mathrm{down}}$, assuming that F_1^{up} and F_1^{down} are disjoint subsets of F_1. Then, $(F_1^{\mathrm{up}}, F_1^{\mathrm{down}})$-compression can be simulated on $\mathbb{G}$ by first uncrossing all 2-chains from $F_1^{\mathrm{up}} F_1^{\mathrm{down}}$ followed by $(F_1^{\mathrm{up}}, F_1^{\mathrm{down}})$-compression.

▶ **Lemma 7.** *Let $\mathbb{G}$ satisfy (I1)–(I4) and $\mathbb{G}' = \textsc{UnaryCmp}(F_1^{up}, F_1^{down}, \textsc{Pop}(F_1^{up}, F_1^{down}, \mathbb{G}))$. Then $\mathrm{val}(\mathbb{G}') = \textsc{UnaryCmp}(F_1^{up}, F_1^{down}, \mathrm{val}(\mathbb{G}))$ and $\mathbb{G}'$ satisfies (I1)–(I4). $\mathcal{O}(g+(n_0+n_1)r)$ credits are issued in the construction of $\mathbb{G}'$, where r is the maximal rank of letters in $\mathbb{G}$. The issued credits and the credits released by UnaryCmp cover the representation cost of fresh letters as well as their credits.*

Since by Lemma 4 we apply $\mathcal{O}(\log n)$ many $(F_1^{\text{up}}, F_1^{\text{down}})$-compressions (for different sets F_1^{up} and F_1^{down}) to $\mathbb{G}$, we obtain:

▶ **Corollary 8.** *(F_1^{up}, F_1^{down})-compression issues in total $\mathcal{O}((g + (n_0 + n_1)r)\log n)$ credits during all modifications of $\mathbb{G}$.*

4.4 Chain compression

Our notations and analysis for chain compression is similar to those for $(F_1^{\text{up}}, F_1^{\text{down}})$-compression. In order to simulate chain compression on $\mathbb{G}$ we want to apply chain compression to the right-hand sides of $\mathbb{G}$. This works as long as there are no crossing chains: A unary letter a *has a crossing chain* in a rule $A_i \to \alpha_i$ if aa has a crossing occurrence in α_i, otherwise it has no crossing chain. As for $(F_1^{\text{up}}, F_1^{\text{down}})$-compression, when there are no crossing chains, we apply chain compression to the right-hand sides of $\mathbb{G}$. We denote with $\textsc{ChainCmp}(\mathbb{G})$ the resulting grammar.

Crossing chains are eliminated by a procedure similar to Pop: Suppose for instance that a has a crossing chain because $a(A_i)$ is a subpattern in a right-hand side and $\operatorname{val}(A_i)$ begins with a. Popping up a does not solve the problem, since after popping, $\operatorname{val}(A_i)$ might still begin with a. Thus, we keep on popping up until the first letter of $\operatorname{val}(A_i)$ is not a. In order to do this in one step we need some notation: We say that a^ℓ is an *a-prefix* of a tree (or pattern) t if $t = a^\ell(t')$ and the first letter of t' is not a (here t' might be the trivial pattern y). Similarly, we say that a^ℓ is an *a-suffix* of a pattern $t(y)$ if $t = t'(a^\ell(y))$ for a pattern $t'(y)$ and the last letter of t' is not a (again, t' might be the trivial pattern y). In this terminology, we have to pop-up (resp. pop-down) the whole a-prefix (resp., a-suffix) of $\operatorname{val}(A_i)$ from of A_i in one step. This is achieved by a procedure RemCrChs, which is similar to Pop. So chain compression is done by first running RemCrChs and then ChainCmp on the right-hand sides of $\mathbb{G}$. We obtain:

▶ **Lemma 9.** *Let $\mathbb{G}$ satisfy (I1)–(I4) and $\mathbb{G}' = \textsc{ChainCmp}(\textsc{RemCrChs}(\mathbb{G}))$. Then $\operatorname{val}(\mathbb{G}') = \textsc{ChainCmp}(\operatorname{val}(\mathbb{G}))$ and $\mathbb{G}'$ satisfies (I1)–(I4). $\mathcal{O}(g + (n_0 + n_1)r)$ credits are issued in the construction of $\mathbb{G}'$ and these credits are used to pay the credits for the fresh letters introduced by ChainCmp (but not their representation cost).*

Since by Lemma 4 we apply $\mathcal{O}(\log n)$ many chain compressions to $\mathbb{G}$, we get:

▶ **Corollary 10.** *Chain compression issues in total $\mathcal{O}((g + (n_0 + n_1)r)\log n)$ credits during all modifications of $\mathbb{G}$.*

The representation cost for the new letters a_ℓ introduced by chain compression is addressed in Section 4.6.

4.5 Leaf compression

In order to simulate leaf compression on $\mathbb{G}$ we perform similar operations as for $(F_1^{\text{up}}, F_1^{\text{down}})$-compression: Ideally we would like to apply leaf compression to each right-hand side of $\mathbb{G}$. However, in some cases this does not return the appropriate result. We say that the pair (f, a) is a *crossing parent-leaf pair* in $\mathbb{G}$, if $f \in F_{\geq 1}$, $a \in F_0$, and one of the following holds:

(L1) $f(t_1, \ldots, t_\ell)$ is a subtree of some right-hand side of $\mathbb{G}$, where for some j we have $t_j = A_k$ and $\operatorname{val}(A_k) = a$.
(L2) For some $A_i \in N_1$, $A_i(a)$ is a subtree of some right-hand side of $\mathbb{G}$ and the last letter of $\operatorname{val}(A_i)$ is f.
(L3) For some $A_i \in N_1$ and $A_k \in N_0 \cup \widetilde{N_0}$, $A_i(A_k)$ is a subtree of some right-hand side of $\mathbb{G}$, the last letter of $\operatorname{val}(A_i)$ is f, and $\operatorname{val}(A_k) = a$.

When there is no crossing parent-leaf pair, we can apply leaf compression to each right-hand side of a rule; denote the resulting grammar with LeafCmp($\mathbb{G}$). If there is a crossing parent-leaf pair, we uncross them all by a generalisation of Pop, called GenPop, which pops up letters from F_0 and pops down letters from $F_{\geq 1}$. The latter requires some generalisation: If we want to pop down a letter of rank > 1, we need to pop a whole handle. This adds new nonterminals to $\mathbb{G}$ as well as a large number of new letters and hence a large amount of credit, so we need to be careful. There are two crucial details:

- When we pop down a whole handle $h = f(t_1, \ldots, t_k, y, t_{k+1}, \ldots, t_\ell)$, we add to the set $\widetilde{N}_0$ fresh nonterminals for all trees t_i that are non-constants, replace these t_i in h by their corresponding nonterminals and then pop down the resulting handle. In this way the issued credit is reduced and no new occurrence of nonterminals from $N_0 \cup N_1$ is created.
- We do not pop down a handle from every nonterminal, but do it only when it is needed, i.e., if for $A_i \in N_1$ one of the cases (L2) or (L3) holds. This allows preserving (I5). Note that when the last symbol in the rule for A_i is not a handle but another nonterminal, this might cause a need for recursive popping. So we perform the whole popping down in a depth-first-search style.

So, for leaf compression we can proceed as in the case of $(F_1^{\mathrm{up}}, F_1^{\mathrm{down}})$-compression and chain compression: We first uncross all parent-leaf pairs and then compress each right-hand side independently.

▶ **Lemma 11.** *Let $\mathbb{G}$ satisfy (I1)–(I4) and $\mathbb{G}' = \mathrm{LeafCmp}(\mathrm{GenPop}(\mathbb{G}))$. Then $\mathrm{val}(\mathbb{G}') = \mathrm{LeafCmp}(\mathrm{val}(\mathbb{G}))$ and $\mathbb{G}'$ satisfies (I1)–(I4). $\mathcal{O}(g + (n_0 + n_1)r)$ credits are issued in the construction of $\mathbb{G}'$. The issued credit and the credit released by* LeafCmp *cover the representation cost of fresh letters as well as their credit.*

Since by Lemma 4 we apply $\mathcal{O}(\log n)$ many leaf compressions to $\mathbb{G}$, we obtain:

▶ **Corollary 12.** *Leaf compression issues in total $\mathcal{O}(((n_0 + n_1)r + g) \log n)$ credits during all modifications of $\mathbb{G}$.*

4.6 Calculating the total cost of representing letters

The issued credit of (which is $\mathcal{O}(((n_0 + n_1)r + g) \log n)$ by Corollaries 8, 10, and 12) is enough to pay the 2 credits for every letter introduced during popping, whereas the released credit covers the representation cost for the new letters introduced by $(F_1^{\mathrm{up}}, F_1^{\mathrm{down}})$-compression and leaf compression. However, the released credit *does not* cover the representation cost for letters created during chain compression. The appropriate analysis is similar to [9]. The idea is as follows: Firstly, we define a scheme of representing letters introduced by chain compression based on the grammar $\mathbb{G}$ and the way $\mathbb{G}$ is changed by chain compression (the *$\mathbb{G}$-based representation*). Then, we show that for this scheme the representation cost is bounded by $\mathcal{O}((g + (n_0 + n_1)r) \log n)$. Lastly, it is proved that the actual representation cost of letters introduced by chain compression during the run of TtoG (the TtoG-*based representation*, whose cost is given by Lemma 2) is smaller than the $\mathbb{G}$-based one. Hence, it is bounded by $\mathcal{O}((g + (n_0 + n_1)r) \log n)$, too. Adding this to the issued credit, we obtain the main result of the paper:

▶ **Corollary 13.** *The total representation cost of the letters introduced by* TtoG *(and hence the size of the grammar produced by* TtoG*) is $\mathcal{O}((g + (n_0 + n_1)r) \log n) \leq \mathcal{O}(g \cdot r \cdot \log n)$, where g is the size of a minimal handle grammar for the input tree T and r the maximal rank of symbols in T.*

Together with Lemma 6 we get:

▶ **Corollary 14.** *The size of the grammar produced by* TtoG *is* $\mathcal{O}(g\, r^2 \log n)$*, where* g *is the size of a minimal SLCF grammar for the input tree* T *and* r *is the maximal rank of symbols in* T*.*

Acknowledgements. The first author would like to thank P. Gawrychowski for introducing him to the topic of compressed data and discussions, as well as S. Maneth and S. Böttcher for the question of applicability of the recompression-based approach to the tree case.

References

1. T. Akutsu. A bisection algorithm for grammar-based compression of ordered trees. *Inf. Process. Lett.*, 110(18-19):815–820, 2010.
2. P. Bille, I. Gørtz, G. Landau, and O. Weimann. Tree compression with top trees. In *Proc. ICALP 2013 (1)*, LNCS 7965, pp.160–171. Springer, 2013.
3. G.Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4–5):456–474, 2008.
4. M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.
5. F. Claude and G. Navarro. Fast and compact web graph representations. *ACM Trans. Web*, 4(4), 2010.
6. T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.
7. A. Jeż. Compressed membership for NFA (DFA) with compressed labels is in NP (P). In *Proc. STACS 2012*, vol. 14 of *LIPIcs*, pp.136–147, Leibniz-Zentrum für Informatik, 2012.
8. A. Jeż. Faster fully compressed pattern matching by recompression. In *Proc. ICALP 2012 (1)*, LNCS 7391, pp.533–544. Springer, 2012.
9. A. Jeż. Approximation of grammar-based compression via recompression. In *Proc. CPM 2013*, LNCS 7922, pp.165–176. Springer, 2013. full version at http://arxiv.org/abs/1301.5842.
10. A. Jeż. One-variable word equations in linear time. In *Proc. ICALP 2013 (2)*, LNCS 7966, pp.324–335. Springer, 2013.
11. A. Jeż. Recompression: a simple and powerful technique for word equations. In *Proc. STACS 2013*, volume 20 of *LIPIcs*, pp.233–244, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2013.
12. N. Jesper Larsson and A. Moffat. Offline dictionary-based compression. In *Proc. DCC 1999*, pp.296–305. IEEE Computer Society Press, 1999.
13. M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.
14. M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Inf. Syst.*, 38(8):1150–1167, 2013.
15. M. Lohrey, S. Maneth, and E. Nöth. XML compression via DAGs. In *Proc. ICDT 2013*, pp.69-80, ACM, 2013.
16. M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *J. Comput. Syst. Sci.*, 78(5):1651–1669, 2012.
17. W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003.
18. H. Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *J. Discrete Algorithms*, 3(2-4):416–430, 2005.

Coloring 3-colorable graphs with $o(n^{1/5})$ colors

Ken-ichi Kawarabayashi[*1] **and Mikkel Thorup**[†2]

1 **National Institute of Informatics, Tokyo Japan / JST ERATO Kawarabayashi Project, Tokyo, Japan**
k_keniti@nii.ac.jp

2 **University of Copenhagen, Copenhagen, Denmark**
mikkel2thorup@gmail.com

Abstract

Recognizing 3-colorable graphs is one of the most famous NP-complete problems [Garey, Johnson, and Stockmeyer STOC'74]. The problem of coloring 3-colorable graphs in polynomial time with as few colors as possible has been intensively studied: $O(n^{1/2})$ colors [Wigderson STOC'82], $\widetilde{O}(n^{2/5})$ colors [Blum STOC'89], $\widetilde{O}(n^{3/8})$ colors [Blum FOCS'90], $O(n^{1/4})$ colors [Karger, Motwani, Sudan FOCS'94], $\widetilde{O}(n^{3/14}) = O(n^{0.2142})$ colors [Blum and Karger IPL'97], $O(n^{0.2111})$ colors [Arora, Chlamtac, and Charikar STOC'06], and $O(n^{0.2072})$ colors [Chlamtac FOCS'07]. Recently the authors got down to $O(n^{0.2049})$ colors [FOCS'12]. In this paper we get down to $O(n^{0.19996}) = o(n^{1/5})$ colors.

Since 1994, the best bounds have all been obtained balancing between combinatorial and semi-definite approaches. We present a new combinatorial recursion that only makes sense in collaboration with semi-definite programming. We specifically target the worst-case for semi-definite programming: high degrees. By focusing on the interplay, we obtained the biggest improvement in the exponent since 1997.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems – Computations on discrete structures, G.2.2 Graph Theory – Graph algorithms

Keywords and phrases Approximation Algorithms, Graph Coloring

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.458

1 Introduction

If ever you want to illustrate the difference between what we consider hard and easy to someone not from computer science, use the example of 2-coloring versus 3-coloring: suppose there is too much fighting in a class, and you want to split it so that no enemies end up in the same group. First you try with a red and a blue group. Put someone in the red group, and everyone he dislikes in the blue group, everyone they dislike in the red group, and so forth. This is an easy systematic approach. Digging a bit deeper, if something goes wrong, you have an odd cycle, and it is easy to see that if you have a necklace with an odd number of red and blue beads, then the colors cannot alternate perfectly. This illustrates both efficient algorithms and the concept of a witness. Knowing that red and blue do not suffice, we might try introducing green, but this is already beyond what we believe computers can do.

* Research partly supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research and by Mitsubishi Foundation.
† Research partly supported by an Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research carrier programme.



© Ken-ichi Kawarabayashi and Mikkel Thorup;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 458–469

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


Formally a k-coloring of an undirected graph assigns k colors to the vertices. The coloring is only valid if no two adjacent vertices get the same color. The validity of coloring is trivially checked in linear time so the deciding if a graph is k-colorable is in NP.

Three-coloring is a classic NP-hard problem. It was proved hard by Garey, Johnson, and Stockmeyer at STOC'74 [9], and was the prime example of NP-hardness mentioned by Karp in 1975 [13]. Bipartite or 2-colorable graphs are very well-understood. How about tripartite or 3-colorable graphs? How can we reason about them if we cannot recognize them? Three-colorable graphs are obvious targets for any approach to NP-hard problems. With the approximation approach, given a 3-colorable graph, that is a graph with an unknown 3-coloring, we try to color it in polynomial time using as few colors as possible. The algorithm is allowed to fail or give up if the input graph was not 3-colorable. If a coloring is produced, we can always check that it is valid even if the input graph is not 3-colorable. This challenge has engaged many researchers. At STOC'82, Wigderson [17] got down to $O(n^{1/2})$ colors for a graph with n vertices. Berger and Rompel [3] improved this to $O((n/(\log n))^{1/2})$. Blum [4] came with the first polynomial improvements, first to $\widetilde{O}(n^{2/5})$ colors at STOC'89, and then to $\widetilde{O}(n^{3/8})$ colors at FOCS'90.

The next big step at FOCS'94 was by Karger, Motwani, Sudan [12] using semi-definite programming (SDP). This came in the wake of Goemans and Williamson's seminal use of SDP for max-cut at STOC'94 [10]. For a graph with maximum degree $\Delta_{\max}$, Karger et al. got down to $O(\Delta_{\max}^{1/3})$ colors. Combining this with Wigderson's algorithm, they got down to $O(n^{1/4})$ colors. Later Blum and Karger [5] combined the SDP from [12] with Blum's [4] algorithm, yielding an improved bound of $\widetilde{O}(n^{3/14}) = \widetilde{O}(n^{0.2142})$. Later improvements on semi-definite programming have also been combined with Blum's algorithm. At STOC'06, Arora, Chlamtac, and Charikar [1] got down to $O(n^{0.2111})$ colors. The proof in [1] is based on the seminal result of Arora, Rao and Vazirani [2] which gives an $O(\sqrt{\log n})$ algorithm for the sparsest cut problem. At FOCS'07 Chlamtac [6] got down to $O(n^{0.2072})$ colors. Recently, at FOCS'12 [14], we presented a purely combinatorial approach (for the first time since Blum [4]), getting down to $\widetilde{O}(n^{4/11})$ colors. Combining it with Chlamtac's SDP [6], we got down to $O(n^{0.2049})$ colors.

Only a few lower bounds are known for the coloring of 3-colorable graphs. We know that it is NP-hard to get down to 5 colors [11, 15]. Recently, Dinur, Mossel and Regev [7] showed that it's hard to color with any constant number of colors (i.e., $O(1)$ colors) based on a variant of the Unique Games Conjecture.

Integrality gap results [8, 12, 16] indicates that our understanding of SDP coloring [2, 6, 12] is close to optimal, and it is therefore natural to go back and see if we can improve things combinatorially.

In this paper we show how to color any 3-colorable n vertex graph in polynomial time using only $O(n^{0.19996})$ colors. This is the biggest single improvement in the exponent since 1997 [5], and in particular, we pass the $n^{1/5}$ milestone. Our approach is combinatorial, but aiming at a better combination with SDP, we specifically target the worst-case for SDP: high degrees.

Technical perspective. To appreciate our result, we have to consider the interplay between combinatorial and semi-definite methods in the above mentioned papers. A parameter Δ is picked. Using Blum's notion of progress, it suffices to work with graphs that either have minimum degree Δ or maximum degree Δ. A high minimum degree is good for combinatorial approaches while a low maximum degree is good for semi-definite approaches. The best bounds are obtained choosing Δ to balance between the best semi-definite and combinatorial approaches.

On the combinatorial side, the coloring bounds have followed, for $i = 1, 2, 3, 4$, the sequence $\widetilde{O}((n/\Delta)^{i/(2i-1)})$. Here $i = 1$ is from Wigderson at STOC'82 [17], $i = 2$ is from Blum at STOC'89 [4], $i = 3$ is from Blum at FOCS'90 [4][1], and $i = 4$ is from Kawarabayashi and Thorup at FOCS'12 [14]. For $i \to \infty$, the sequence approaches its limit $\widetilde{O}((n/\Delta)^{1/2})$. Each of the above steps is based on a new combinatorial coloring idea. It is rather curious (1) that the resulting bounds have all been of the form $\widetilde{O}((n/\Delta)^{i/(2i-1)})$, and (2) that none of these STOC/FOCS papers skipped a step in this sequence of bounds.

For a purely combinatorial algorithm, we balance the above bounds with the trivial Δ-coloring that takes out any vertex v with $< \Delta$ neighbors, colors the rest of the graph inductively, and give v the first color not used in its neighborhood.

The first semi-definite solution of Karger et al. from FOCS'94 [12], got $O(\Delta^{1/3})$ colors. Balancing this with yet to be found $\widetilde{O}((n/\Delta)^{1/2})$ coloring, would yield $\widetilde{O}(n^{1/5})$ colors, which have thus been a natural milestone. Later semi-definite approaches of Arora, Chlamtac, and Charikar at STOC'06 [1] and Chlamtac at FOCS'07 [6], have gotten down to $O(\Delta^{1/3-\varepsilon(n,\Delta)})$ colors where $\varepsilon(n, \Delta) > 0$ is a small value that decreases as complicated function of Δ. The integrality gap from [8] implies that $\varepsilon(n, \Delta) = o(1)$ for $\Delta = n^{o(1)}$.

Chlamtac [personal communication] stated that we would pass the $n^{1/5}$ milestone if the combinatorial side could get down around $\widetilde{O}((n/\Delta)^{12/23})$ colors. This, however, is 8 steps away in the current sequence where the first 4 steps have taken 20 years, each introducing a new combinatorial idea.

Our goal is to improve the overall coloring bound in terms of n, and we will indeed get down to $o(n^{1/5})$ colors. Using our previous combinatorial algorithm [14] as a subroutine, we present a novel recursion that gets us down to $\widetilde{O}((n/\Delta)^{12/23})$ colors, but only for the large values of Δ needed for an optimal combination with SDP. In combination with Chlamtac's SDP [6], we get a polynomial time algorithm that colors any 3-colorable graph on n vertices with $O(n^{0.19996})$ colors.

We note that for smaller values of Δ, our new recursion does not offer any improvement over our previous combinatorial bound $\widetilde{O}((n/\Delta)^{4/7})$ from [14]. Instead of adding another independent dot, we connect the dots, improving the combinatorial side only in the parameter range of relevance for combination with SDP.

Contents. The paper is organized as follows. In a preliminary Section 2, we present notations and basic results needed from [4]. In Section 3, we review our previous algorithm from [14] which we shall use here as a subroutine. In Section 4 we present our novel recursion around this subroutine. This completes the description of our new algorithm. Switching to the analysis, in Section 5 we identify the properties of the subroutine from [14] that we need for our recursion. This properties follow from the analysis from [14], as will be verified in a combined journal version. In Section 6 we use these properties for an inductive analysis of our new recursion.

2 Preliminaries

We hide $\log n$ factors, so we use the notation that $\widetilde{O}(x) \leq x \log^{O(1)}(n)$, $\widetilde{\Omega}(x) \geq x/\log^{O(1)}(n)$, $\widetilde{o}(x)) \leq x/\log^{\omega(1)}(n)$, and $\widetilde{\omega}(x) \geq x \log^{\omega(1)}(n)$.

We are given a 3-colorable graph $G = (V, E)$ with $|V| = n = \omega(1)$ vertices. The (unknown) 3-colorings are with red, green, and blue. For a vertex v, we let $N(v)$ denote its set of

[1] The reference is to the joint journal paper

neighbors. For a vertex set $X \subseteq V$, let $N(X) = \bigcup_{v \in X} N(v)$ be the neighborhood of X. If Y is a vertex set, we use N_Y to denote neighbors in Y, so $N_Y(v) = N(v) \cap Y$ and $N_Y(X) = N(X) \cap Y$. We let $d_Y(v) = |N(v) \cap Y|$ and $d_Y(X) = \{d_Y(v) \mid v \in Y\}$. Then $\min d_Y(X)$, $\max d_Y(X)$, and $\operatorname{avg} d_Y(X)$, denotes the minimum, maximum, and average degree from X to Y.

For some color target k depending on n, we wish to find an $\widetilde{O}(k)$ coloring of G in polynomial time. We reuse several ideas and techniques from Blum's approach [4].

Progress

Blum has a general notion of *progress towards an $\widetilde{O}(k)$ coloring* (or *progress* for short if k is understood). The basic idea is that such progress eventually leads to a full $\widetilde{O}(k)$ coloring of a graph. Blum presents three types of progress towards $\widetilde{O}(k)$ coloring:
Type 0: Same color. Finding vertices u and v that have the same color in every 3-coloring.
Type 1: Large independent set. Finding an independent or 2-colorable vertex set X of size $\widetilde{\Omega}(n/k)$.
Type 2: Small neighborhood. Finding a non-empty independent or 2-colorable vertex set X such that $|N(X)| = \widetilde{O}(k|X|)$.

In order to get from progress to actual coloring, we want k to be bounded by a *near-polynomial* function f of n where near-polynomial means that f is non-decreasing and that there are constants $c, c' > 1$ such that $cf(n) \leq f(2n) \leq c'f(n)$ for all n. As described in [4], this includes any function of the form $f(n) = n^\alpha \log^\beta n$ for constants $\alpha > 0$ and β.

▶ **Lemma 1** ([4, Lemma 1]). *Let f be near-polynomial. If we in time polynomial in n can make progress towards $\widetilde{O}(f(n))$ coloring of either Type 0, 1, or 2, on any 3-colorable graph on n vertices, then in time polynomial in n, we can $\widetilde{O}(f(n))$ color any 3-colorable graph on n vertices.*

The general strategy is to identify a small parameter k for which we can guarantee progress. To apply Lemma 1 and get a coloring, we need a bound f on k where f is near-polynomial in n. As soon as we find one progress of the above types, we are done, so generally, whenever we see a condition that implies progress, we assume that the condition is not satisfied.

Our focus is to find a vertex set X, $|X| > 1$, that is guaranteed to be monochromatic in every 3-coloring. This will happen assuming that we do not get other progress on the way. When we have the vertex set X, we get same-color progress for any pair of vertices in X. We refer to this as *monochromatic progress*.

Most of our progress will be made via results of Blum presented below using a common parameter

$$\Psi = n/k^2. \tag{1}$$

A very useful tool we get from Blum is the following multichromatic (more than one color) test:

▶ **Lemma 2** ([4, Corollary 4]). *Given a vertex set $X \subseteq V$ of size at least $\Psi = n/k^2$, in polynomial time, we can either make progress towards an $\widetilde{O}(k)$-coloring of G, or else guarantee that under* every *legal 3-coloring of G, the set X is multichromatic.*

The following lemma is implicit in [4] and explicit in [14].

▶ **Lemma 3** ([14, Lemma 6]). *If the vertices in a set Z on the average have d neighbors in U, then the whole set Z has at least $\min\{d/\Psi, |Z|\}\, d/2$ distinct neighbors in U (otherwise some progress is made).*

Large minimum degree

Our algorithms will exploit a lower bound Δ on the minimum degree in the graph. It is easily seen that if a vertex v has d neighbors, then we can make progress towards $\widetilde{O}(d)$ coloring since this is a small neighborhood for Type 2 progress. For our color target k, we may therefore assume:

$$k \leq \Delta / \log^a n \text{ for any constant } a. \tag{2}$$

However, combining with semi-definite programming (SDP) as in [5], we can assume a much larger minimum degree. The combination is captured by the following lemma, which is proved in [14, §VIII]:

▶ **Lemma 4** ([5, 14]). *Suppose that for some near-polynomial functions d and f, we for any n can make progress towards $\widetilde{O}(f(n))$ coloring for*

- *any 3-colorable graph on n vertices with minimum degree $\geq d(n)$.*
- *any 3-colorable graph on n vertices with maximum degree $\leq d(n)$.*

Then we can make progress towards $\widetilde{O}(f(n))$-coloring on any 3-colorable graph on n vertices.

Using the SDP from [12], we can make progress towards $d(n)^{1/3}$ for graphs with degrees below $d(n)$, so by Lemma 4, we may assume

$$k \leq \Delta^{1/3}. \tag{3}$$

We can do even better using the strongest SDP result of Chlamtac from [6, Theorem 15]:

▶ **Theorem 5** ([6]). *For any $\tau > \frac{6}{11}$ there is a $c > 0$ such that there is a polynomial time algorithm that for any 3-colorable graph G with n vertices and all degrees below $\Delta = n^\tau$ finds an independent set of size $\widetilde{\Omega}\left(n/\Delta^{1/(3+3c)}\right)$. Hence we can make Type 2 progress towards an $\widetilde{O}\left(\Delta^{1/(3+3c)}\right) = \widetilde{O}\left(n^{\tau/(3+3c)}\right)$-coloring.*

The requirement on τ and c is that $c < 1/2$ and $\lambda_{c,\tau}(\alpha) = 7/3 + c + \alpha^2/(1-\alpha^2) - (1+c)/\tau - (\sqrt{(1+\alpha)/2} + \sqrt{c(1-\alpha)/2})^2$ is positive for all $\alpha \in [0, \frac{c}{1+c}]$.

▶ **Corollary 6.** *In polynomial time, for any 3-colorable graph with n vertices, and all degrees below $\Delta = n^{0.61674333}$, we can make progress towards an $\widetilde{O}(n^{0.19996})$-coloring.*

Proof. We apply Theorem 5 with $\tau = 0.6167433$ and $c = 0.02811113$. Then For $\alpha \in [0, \frac{c}{1+c}]$, it is easily verified that $\lambda_{c,\tau}(\alpha)$ is minimized and positive with $\alpha = 0.0273425$. Then $\tau/(3+3c) = 0.19996$. ◀

By Lemma 4, we may thus assume

$$k = n^{0.19996} \text{ and } \Delta = n^{0.61674333}. \tag{4}$$

With this setting k is slightly smaller than $(n/\Delta)^{12/23}$. Our original algorithm from [14] only assumes the combinatorial bound (2), to make progress towards $\widetilde{O}((n/\Delta)^{4/7})$ coloring. It is only in our new developments that we need the higher degrees that can be assumed via SDP.

Two-level neighborhood structure

The most complex ingredient we get from Blum [4] is a certain regular second neighborhood structure. Let Δ be the smallest degree in the graph G. In fact, we shall use the slightly modified version described in [14].

Unless other progress is made, for some $\Delta_1 = \widetilde{\Omega}(\Delta)$, in polynomial time [4, 14] identifies a 2-level neighborhood structure $H_1 = (r_1, S_1, T_1)$ in G consisting of:

- A root vertex r_1. We assume r_1 is colored red in any 3-coloring.
- A first neighborhood $S_1 \subseteq N(r_1)$ of size at least Δ_1.
- A second neighborhood $T_1 \subseteq N(S_1)$ of size at most n/k. The sets S_1 and T_1 may overlap.
- The edges between vertices in H_1 are the same as those in G.
- The vertices in S_1 all have degrees at least Δ_1 into T_1.
- For some δ_1 the degrees from T_1 to S_1 are all between δ_1 and $5\delta_1$.

3 Review of our FOCS'12 coloring

Our algorithm makes internal use of the coloring algorithm from [14], which we review below. It uses the above 2-level neighborhood structure $H_1 = (r_1, S_1, T_1)$, and works on induced subproblems $(S,T) \subseteq (S_1,T_1)$ defined in terms of a subsets $S \subseteq S_1$ and $T \subseteq T_1$. The edges considered in the subproblem are exactly those between S and T in G. This edge set is denoted $E(S,T)$.

With r_1 red in any 3-coloring, we know that all vertices in $S \subseteq S_1 \subseteq N(r_1)$ are blue or green. We say that a vertex in T has *high S-degree* if its degree to S is bigger than $\delta_1/4$, and we will make sure that any subproblem (S,T) considered satisfies:

(i) We have more than Ψ vertices of high S-degree in T.

In [14, §IV] we implemented a subroutine `cut-or-color`(t,S,T) which for a problem $(S,T) \subseteq (S_1,T_1)$ starts with an arbitrary high S-degree vertex $t \in T$. It has one of the following outcomes:

- Some progress toward a $\widetilde{O}(k)$-coloring. Then we are done, so we assume that this does not happen.
- A guarantee that if r_1 and t have different colors in a 3-coloring C_3 of G, then S is monochromatic in C_3.
- Reporting a "sparse cut around a subproblem $(X,Y) \subseteq (S,T)$" satisfying the following conditions:
  - **(i)** The original high S-degree vertex t has all its neighbors from S in X, that is, $N_S(t) \subseteq X$.
  - **(ii)** All edges from X to T go to Y, so there are no edges between X and $T \setminus Y$.
  - **(iii)** Each vertex $s' \in S \setminus X$ has $|N_Y(s')| < \Psi$.
  - **(iv)** Each vertex $t' \in T \setminus Y$ has $|N_Y(N_S(t'))| < \Psi$.

Assuming `cut-or-color`, we now review the main recursive algorithm, `monochromatic`, from [14]. It takes as input a subproblem (S,T). The pseudo-code is presented in Algorithm 1.

Algorithm 1: `monochromatic`(S,T)

```
let U be the set of high S-degree vertices in T;
check that U is multichromatic in G with Lemma 2;      // if not, progress found and we are done
if there is a t ∈ U such that cut-or-color(S,T,t) returns "sparse cut around (X,Y)" then
    recursively call monochromatic(X,Y)
else
    return "S is monochromatic in every 3-coloring"
```

Let U be the set of high S-degree vertices in T. By 1 we have $|U| \geq \Psi$, so we can apply Blum's multichromatic test from Lemma 2 to U in G. Assuming we did not make progress,

we know that U is multichromatic in every valid 3-coloring. We now apply `cut-or-color` to each $t \in U$, stopping only if a sparse cut is found or progress is made. If we make progress, we are done, so assume that this does not happen. If a sparse cut around a subproblem (X, Y) is found, we recurse on (X, Y).

The most interesting case is when we get neither progress nor a sparse cut.

▶ **Lemma 7.** *If* `cut-or-color` *does not find progress nor a sparse cut for any high S-degree $t \in U$, then S is monochromatic in every 3-coloring of G.*

Proof. Consider any 3-coloring C_3 of G. With Lemma 2 we checked that U is multichromatic in every 3-coloring of G including C_3, so there is some $t \in U$ that has a different color than r_1 in C_3. With this t, `cut-or-color`(S, T, t) guarantees that S is monochromatic in C_3. Note that different 3-colorings may use a different vertex t for the guarantee, and our algorithm does not need to know which t are used. ◀

Thus, unless other progress is made, `monochromatic` ends up with a set S that is monochromatic in every 3-coloring, and then monochromatic progress can be made. However, the correctness demands that we respect 1 and never apply `monochromatic` to a subproblem (S, T) where T has less than Ψ high S-degree vertices (otherwise Lemma 2 cannot be applied to U).

In [14] it is proved that 1 is respected when the recursion `monochromatic`(S_1, T_1) starts in the initial two-level structure (S_1, T_1) from Section 2. In each recursive step, we take the subproblem (X, Y) returned by `cut-or-color`, and recurse on $(S, T) = (X, Y)$. The analysis from [14] has the following points:

- If the average degree from Y to X is at least $\delta_1/2$, then $(S, T) = (X, Y)$ satisfies 1.
- The set Y is of size at least $\Delta_1^2 k^2/(2n)$ and has at least $\delta_1|Y| \geq \delta_1\Delta_1^2 k^2/(2n)$ edges to S_1.
- The set Y has at most $(40\delta_1 n^2)/(\Delta_1^2 k^4) \cdot |T_1|$ edges to $T_1 \setminus X$ (this is the hard part).
- We pick $k = \Theta((n/\Delta_1)^{4/7})$ such that

$$\frac{40\delta_1 n^2}{\Delta_1^2 k^4}|T_1| = \delta_1\Delta_1^2 k^2/(4n) \leq \delta_1|Y|/2.$$

Then the average degree from Y to X is at least $\delta_1/2$, implying 1 for $(S, T) = (X, Y)$.

4 A novel outer loop for high degree graphs

As described above, the first call to Algorithm 1 is with the initial problem (S_1, T_1) that is *regular* in the sense that the degrees from T_1 to S_1 are all between δ_1 and $5\delta_1$ for some δ_1. However, with a color target k below $\Theta((n/\Delta_1)^{4/7})$, we can no longer guarantee that the average degree from Y to X remains above $\delta_1/2$. To preserve the correctness, we will stop our recursive Algorithm 1 if we get to a subproblems (X, Y) where the average degree from Y to X is less than $\delta_1/2$. Inside (X, Y) we find a new regular subproblem (S_2, T_2) where the degrees are between δ_2 and $5\delta_2$ for some δ_2. Again we apply Algorithm 1 until the average drops below $\delta_2/2$. We continue this new outer loop, generating a sequence of regular subproblems $(S_1, T_1) \supset (S_2, T_2) \supset (S_3, T_3) \supset \cdots$, until we somehow end up either making progress, or some error event happens. In combination with SDP, our analysis will show that this outer loop can be used to give error-free progress towards $\tilde{O}(n^{0.19996})$ coloring.

The regularization is described in Algorithm 2, and it is, in itself, fairly standard. Blum [4] used several similar regularizations.

Algorithm 2: `regularize`(S,T)

Let $d_\ell = (4/3)^\ell$;
Partition the vertices of T into sets $U_\ell = \{v \in T \mid d_S(v) \in [d_\ell, d_{\ell+1})\}$;
Subject to $d_\ell \geq \operatorname{avg} d_S(T)/2$ let ℓ maximize $|E(U_\ell, S)|$;
$\delta^r \leftarrow d_\ell/4$; $\Delta^r \leftarrow \operatorname{avg} d_{U_\ell}(S)/4$;
Repeatedly remove vertices $v \in S$ with $d_{U_\ell}(v) \leq \Delta^r$ and $w \in U_\ell$ with $d_S(w) \leq \delta^r$;
$S^r \leftarrow S$; $T^r \leftarrow U_\ell$;
return $(S^r, T^r, \Delta^r, \delta^r)$

▶ **Lemma 8.** *When* `regularize`(S,T) *in Algorithm 2 returns* $(S^r, T^r, \Delta^r, \delta^r)$ *then* $\Delta^r \geq \operatorname{avg} d_T(S)/(30 \lg n)$ *and* $\delta^r \geq \operatorname{avg} d_S(T)/8$. *The sets* S^r *and* T^r *are both non-empty. The degrees from* S^r *to* T^r *are at least* Δ^r *and the degrees from* T^r *to* S^r *are between* δ^r *and* $5\delta^r$.

Proof. Below S and U_ℓ refers to the sets before vertices are removed. We have S^r and T^r denoting the sets after the vertices have been removed.

To prove $\Delta^r \geq \operatorname{avg} d_T(S)/(30 \lg n)$, we first note that the sets U_ℓ with $d_\ell < \operatorname{avg} d_S(T)/2$ only contain vertices of degree below $(4/3)\operatorname{avg} d_S(T)/2 = (2/3)\operatorname{avg} d_S(T)$, so at least $1/3$ of the edges from $E(S,T)$ leave vertices from sets U_ℓ satisfying the condition $d_\ell \leq \operatorname{avg} d_S(T)/2$. There are only $\log_{4/3} n < (5/2)\lg n$ possible values of ℓ, and subject to the condition, we picked ℓ maximizing $E(S, U_\ell)$. Therefore $|E(S,U_\ell)| > (1/3)|E(S,T)|/((5/2)\lg n) = (2/15)|E(S,T)|/\lg n$. It follows that $\Delta^r \geq \operatorname{avg} d_{U_\ell}(S)/4 > \operatorname{avg} d_T(S)/(30 \lg n)$.

The only other slightly non-trivial statement is that the sets S^r and T^r do not end up empty. When we remove vertices from S, we remove at most $|S|\operatorname{avg} d_{U_\ell}(S)/4 \leq |E(S,U_\ell)|/4$ edges, and likewise for the vertices removed from U_ℓ, so these removals take away at most half the edges. It follows that some edges remain hence that $S^r, T^r \neq \emptyset$. ◀

Our new coloring is described in Algorithm 3. Except for the possible regularization, each round j is an iterative version of the recursive Algorithm 1. Moreover, we have made it self-checking in the sense that we report an error if the set U of high degree vertices is too small for 1 ("Error B" below). Also, we report an error if the set S is too small for monochromatic progress which requires at least two same-color vertices ("Error A" below). With $k = \Theta((n/\Delta)^{4/7})$, the analysis from [14] shows that we never get an error and that we never get $|E(S,T)| \leq \delta_1|T|/2$, so the regularization never happens.

The outer loop in Algorithm 3 continues until it either makes an error, or makes progress. The progress can either be explicit with a monochromatic set, or it can happen implicitly as part of the multichromatic test from Lemma 2. Ensuring that we make progress and no errors happen will require a very careful choice of parameters, and we will only gain over [14] when large minimum degree vertices are guaranteed from SDP as in (3) or (4).

5 A good round

When we start round j of Algorithm 3 with a problem $(S_j, T_j, \Delta_j, \delta_j)$, it follows directly from Lemma 8 that the degrees from T_j to S_j are between δ_j and $5\delta_j$, and that the degrees from S_j to T_j are all at least Δ_j.

The journal version of this conference paper will also cover [14] and there we will make a simple generalization of the analysis from [14] so that it applies to an arbitrary round j

Algorithm 3: Seeking progress towards $\widetilde{O}(k)$ coloring

```
let (S_1, T_1, Δ_1, δ_1) be the initial two-level structure from Section 2;
for j ← 1, 2, ... do        // outer loop, round j
    (S, T) ← (S_j, T_j);
    repeat        // iterative version of recursive monochromatic(S_j, T_j)
        if |S| ≤ 1 then return "Error A";
        U ← {v ∈ T | d_S(v) ≥ δ_j/4};
        if |U| < Ψ then return "Error B";
        check U multichromatic with Lemma 2;        // if not, progress was found and we are done
        if ∃t ∈ U such that cut-or-color(S, T, t) returns "sparse cut around (X, Y)" then
            (S, T) ← (X, Y)
        else return "S is monochromatic in every 3-coloring, so monochromatic progress found"
    until |E(S, T)| < δ_j|T|/2;
    (S_j, T_j, Δ_j, δ_j) = regularize(S, T);
```

of the outer loop in Algorithm 3—not just round 1. Below we describe the outcome of the analysis.

The basic requirements for the analysis is that the following *pre-conditions* are satisfied:

$$\Delta_j = \omega(\Psi) \tag{5}$$

$$\delta_j \geq 4\Delta_j/\Psi \tag{6}$$

Based on the pre-conditions, it will follow that no error is made in the round. Also, for any subproblem (X, Y) considered, it will follow that

$$\min d_Y(X) \geq \Delta_j \tag{7}$$

$$|X| \geq \delta_j/4 \tag{8}$$

$$|Y| \geq \Delta_j^2/(2\Psi) \tag{9}$$

If no progress is made in the round, we will get to a subproblem (X, Y) where the average degree from Y to X is smaller than $\delta_j/2$. This is where we terminate the round and regularize. Let (X_j, Y_j) denote this final subproblem of round j. The most interesting part of the analysis is to argue

$$|Y_j| \leq |T_j|(80n^2)/(\Delta_j^2 k^4). \tag{10}$$

Note here that if the upper bound from (10) is smaller than the lower bound in (9), then we can conclude that we never get to the last subproblem (X_j, Y_j), hence progress must have be made in round j. With the parameters from [14], we get this contradiction already for the first round $j = 1$. However, with a smaller k, the upper bound is higher, and then more rounds may happen.

We say round j is *good* if

- the pre-conditions (5) and (6) are satisfied at the beginning of the round.
- No error is made during the round.
- (7)–(10) are satisfied as long as no progress is made.

A simple generalization of the analysis from [14] implies

▶ **Lemma 9** ([14]). *Round j is good if and only if pre-conditions (5) and (6) are satisfied.*

6 Analysis of outer loop

Using Lemma 9 we will prove

▶ **Theorem 10.** *Suppose for some integer $c = O(1)$ that*

$$k = \widetilde{\omega}\left((n/\Delta)^{\frac{2c+2}{4c+3}}\right). \tag{11}$$

and for all $j = 1, ..., c-1$,

$$(\Delta/k)(\Delta k/n)^j(\Delta k^2/n)^{j(j+1)} = \Delta^{j^2+2j+1}k^{2j^2+3j-1}/n^{j^2+2j} = \widetilde{\omega}(1). \tag{12}$$

Then Algorithm 3 will make only good rounds, and make progress towards an $\widetilde{O}(k)$ coloring no later than round c.

If we did not have the j-bound (12), we would just make c very large, with (11) converging to $(n/\Delta)^{1/2}$. The j-bound (12) is rather unattractive, but we need it to make sure that no errors are made when $c > 1$. As an example, our previous bound $k = \widetilde{O}((n/\Delta)^{4/7})$ from [14] corresponds to the case $c = 1$ in (11). To improve this bound, we need $c > 1$. In particular, we need to satisfy (12) for $j = 1$ which becomes $\Delta^4 k^4/n^3 = \widetilde{\omega}(1)$. Now $k \leq (n/\Delta)^{4/7}$ implies $\Delta^4(n/\Delta)^{4/7\cdot 4}/n^3 = \Delta^{12/7}/n^{5/7} = \widetilde{\omega}(1) \iff \Delta = \widetilde{\omega}(n^{5/12})$. Thus we can only make improvements over [14] if we restrict ourselves to sufficiently high degrees, e.g., relying on SDP for lower degrees.

Note that if $\sqrt{n/\Delta} < k < n/\Delta$, then (12) must be minimized for some unique $j \in \mathbb{R}$. However, since $c = O(1)$, we can easily check (12) for all $j = 1, .., c-1$ with a computer.

In the rest of this section, we will prove Theorem 10 by induction assuming (11) and (12). Also, from Lemma 9, we know that round j is good if the pre-conditions (5) and (6) are satisfied. First, for the base case, we will show (a) that the pre-conditions are satisfied for round 1, and hence that round 1 is good. Next, assuming the first j rounds are good, but no progress is made, we will show (b) that $j < c$ and (c) that the pre-conditions of round $j+1$ are satisfied. By induction, (a), (b), and (c) imply Theorem 10.

First round

For the pre-conditions of the first round, we need

▶ **Lemma 11.** $\Psi = n/k^2 = \Delta/n^{\Omega(1)}$

Proof. Since c is constant, (11) implies $k > (n/\Delta)^{1/2+\Omega(1)}$, hence $\Psi = n/k^2 = \Delta/(n/\Delta)^{\Omega(1)}$. Finally we need to argue $(n/\Delta) = n^{\Omega(1)}$. This follows because the neighborhood of any vertex is 2-colorable, hence we always make Type 2 progress towards $\widetilde{O}(n/\Delta)$ coloring. We are only aiming for $k = n^{\Omega(1)}$ coloring, so we would be done if $(n/\Delta) = n^{o(1)}$. ◀

Since $\Delta_1 = \widetilde{\Omega}(\Delta)$, we get $\Psi = o(\Delta_1)$. Pre-condition (5) for round 1 thus follows Lemma 11.

For pre-condition (6), we note that $\Delta_1/\Psi = \Delta_1 k^2/n$ and $5\delta_1 \geq \Delta_1|S_1|/|T_1| = \widetilde{\Omega}(\Delta_1 \Delta k/n)$. Moreover, by (3) we have $k \leq \Delta^{1/3}$. Hence $\delta_1 \gg 4\Delta_1/\Psi$, so pre-condition (6) is also satisfied for round 1. Thus we conclude that both pre-conditions are satisfied for round 1, hence by Lemma 9, round 1 is good.

General rounds

Now for $j \le c$, we assume that the first j rounds are good but that no progress is made. We want to prove that $j < c$ and that the pre-conditions of round $j+1$ are satisfied.

Since no progress is made, round j ends up regularizing. As in Section 5, we let X_j and Y_j denote the last values of X and Y. Then

$$(S_{j+1}, T_{j+1}, \Delta_{j+1}, \delta_{j+1}) = \texttt{regularize}(X_j, Y_j).$$

We will derive inductive bounds on $|T_{j+1}|$, Δ_{j+1}, and δ_{j+1}. From (7) and Lemma 8 with $(S,T) = (X_j, Y_j)$, we get

$$\Delta_{j+1} \ge \operatorname{avg} d_{Y_j}(X_j)/(30 \lg n) \ge \Delta_j/(30 \lg n) \ge \Delta_1/(30 \lg n)^j = \widetilde{\Omega}(\Delta). \tag{13}$$

From (10) we also have

$$\begin{aligned} |T_{j+1}| \;\le\; |Y_j| \;&\le\; |T_j|\,(80n^2)/(\Delta_j^2 k^4) = \widetilde{O}(|T_j|\,n^2/(\Delta^2 k^4)) \\ &= \widetilde{O}\left(|T_1|\left(\frac{n^2}{\Delta^2 k^4}\right)^j\right) = \widetilde{O}\left((n/k)\left(\frac{n^2}{\Delta^2 k^4}\right)^j\right). \end{aligned}$$

From (9), we know that any Y considered, including Y_j, is of size at least $\Delta_j^2/(2\Psi) = \widetilde{\Omega}(\Delta^2/\Psi) = \widetilde{\Omega}(\Delta^2 k^2/n)$, so we must have

$$\Delta^2 k^2/n = \widetilde{O}\left((n/k)\left(\frac{n^2}{\Delta^2 k^4}\right)^j\right) \iff k = \widetilde{O}\left((n/\Delta)^{\frac{2j+2}{4j+3}}\right).$$

By (11) this implies that $j < c$.

Next we need to argue that preconditions (5) and (6) are satisfied for round $j+1$. By (13), we get that (5) follows from Lemma 11.

The critical issue is to make sure that pre-condition (6) is satisfied with $\delta_{j+1} \ge 4\Delta_j/\Psi$. Using (7)–(10), we get that

$$\begin{aligned} \operatorname{avg} d_{X_j}(Y_j) \;&\ge\; \Delta_j |X_j|/|Y_j| = \Delta_j(\delta_j/4)/\widetilde{O}\left((n/k)\left(\frac{n^2}{\Delta^2 k^4}\right)^j\right) \\ &= \widetilde{\Omega}\Big(\Delta_j \delta_j (k/n)\left(\Delta^2 k^4/n^2\right)^j\Big). \end{aligned}$$

By Lemma 8, $\delta_{j+1} \ge \operatorname{avg} d_{X_j}(Y_j)/8$, so $\delta_{j+1} = \delta_j\, \widetilde{\Omega}\Big(\Delta_j (k/n)\left(\Delta^2 k^4/n^2\right)^j\Big)$. Inductively, since $j = O(1)$ and $\Delta_h = \widetilde{\Omega}(\Delta)$ for all $h \le j$, it follows that

$$\delta_j = \widetilde{\Omega}\Big(\delta_1 (\Delta k/n)^{j-1}\left(\Delta^2 k^4/n^2\right)^{j(j-1)/2}\Big).$$

Therefore $\delta_{j+1} = \widetilde{\Omega}\Big(\delta_1 (\Delta_j k/n)(\Delta k/n)^{j-1}\left(\Delta^2 k^4/n^2\right)^{j(j+1)/2}\Big)$. Here $5\delta_1 \ge |S_1|\Delta_1/|T_1| = \widetilde{\Omega}(\Delta^2 k/n)$, so we get

$$\delta_{j+1} = \widetilde{\Omega}\Big(\Delta_j (\Delta k/n)^{j+1}\left(\Delta k^2/n\right)^{j(j+1)}\Big)$$

By (12) we have $(\Delta/k)(\Delta k/n)^j (\Delta k^2/n)^{j(j+1)} = \widetilde{\omega}(1)$, so

$$(\Delta k/n)^{j+1}\left(\Delta k^2/n\right)^{j(j+1)} = \widetilde{\omega}(k^2/n) = \widetilde{\omega}(1/\Psi).$$

Thus $\delta_{j+1} = \widetilde{\omega}(\Delta_j/\Psi) > 4\Delta_j/\Psi$, so pre-condition (6) is indeed satisfied for round $j+1$. This completes our proof of Theorem 10. Our main coloring result follows.

▶ **Theorem 12.** *In polynomial time, we can color any 3-colorable n vertex graph using $\widetilde{O}(n^{0.19996})$ colors.*

Proof. We use Chlamtac's SDP [6] for low degrees, so as stated in (4), for progress towards an $k = \widetilde{O}(n^{0.19996})$ coloring, we may assume the minimum degree is at least $\Delta = n^{0.61674333}$. Then $(n/\Delta)^{14/27} < k < (n/\Delta)^{12/23}$, so to satisfy (11) in Theorem 10, we set $c = 6$. It is easily verified that (12) is satisfied for $j = 1, ..., 5$. By Theorem 10, we conclude that progress is made within the first $c = 6$ rounds.

Incidentally, with our particular values of k and Δ, for an integer j, (12) reaches its minimum with $j = 5$. This implies that our bounds also hold with any larger c. ◀

References

1. S. Arora, E. Chlamtac, and M. Charikar. New approximation guarantee for chromatic number. In *Proc. 38th STOC*, pages 215–224, 2006.
2. S. Arora, S. Rao, and U. Vazirani. Expanders, geometric embeddings and graph partitioning. *J. ACM*, 56(2):1–37, 2009. Announced at STOC'04.
3. B. Berger and J. Rompel. A better performance guarantee for approximate graph coloring. *Algorithmica*, 5(3):459–466, 1990.
4. A. Blum. New approximation algorithms for graph coloring. *J. ACM*, 41(3):470–516, 1994. Announced at STOC'89 and FOCS'90.
5. A. Blum and D.R. Karger. An $\tilde{O}(n^{3/14})$-coloring algorithm for 3-colorable graphs. *Inf. Process. Lett.*, 61(1):49–53, 1997.
6. E. Chlamtac. Approximation algorithms using hierarchies of semidefinite programming relaxations. In *Proc. 48th FOCS*, pages 691–701, 2007.
7. I. Dinur, E. Mossel, and O. Regev. Conditional hardness for approximate coloring. *SIAM J. Comput.*, 39(3):843–873, 2009. Announced at STOC'06.
8. U. Feige, M. Langberg, and G. Schechtman. Graphs with tiny vector chromatic numbers and huge chromatic numbers. *SIAM J. Comput.*, 33(6):1338–1368, 2004. Announced at FOCS'02.
9. M.R. Garey, D.S. Johnson, and L.J. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. Announced at STOC'74.
10. M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995. Announced at STOC'94.
11. V. Guruswami and S. Khanna. On the hardness of 4-coloring a 3-colorable graph. *SIAM Journal on Discrete Mathematics*, 18(1):30–40, 2004.
12. D.R. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45(2):246–265, 1998. Announced at FOCS'94.
13. R. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.
14. K. Kawarabayashi and M. Thorup. Combinatorial coloring of 3-colorable graphs. In *Proc. 53rd FOCS*, pages 68–75, 2012.
15. S. Khanna, N. Linial, and S. Safra. On the hardness of approximating the chromatic number. *Combinatorica*, 20(3):393–415, 2000.
16. M. Szegedy. A note on the θ number of Lovász and the generalized Delsarte bound. In *Proc. 35th FOCS*, pages 36–39, 1994.
17. A. Wigderson. Improving the performance guarantee for approximate graph coloring. *J. ACM*, 30(4):729–735, 1983. Announced at STOC'82.

Randomized Online Algorithms with High Probability Guarantees*

Dennis Komm[1], Rastislav Královič[2], Richard Královič[1,3], and Tobias Mömke[4]

1 **ETH Zurich, Switzerland,** dennis.komm@inf.ethz.ch
2 **Comenius University, Bratislava, Slovakia,** kralovic@dcs.fmph.uniba.sk
3 **Google Inc., Zurich, Switzerland,** richard.kralovic@dcs.fmph.uniba.sk
4 **Saarland University, Germany,** moemke@cs.uni-saarland.de

Abstract

We study the relationship between the competitive ratio and the tail distribution of randomized online problems. To this end, we define a broad class of online problems that includes some of the well-studied problems like paging, k-server and metrical task systems on finite metrics, and show that for these problems it is possible to obtain, given an algorithm with constant expected competitive ratio, another algorithm that achieves the same solution quality up to an arbitrarily small constant error with high probability; the "high probability" statement is in terms of the optimal cost. Furthermore, we show that our assumptions are tight in the sense that removing any of them allows for a counterexample to the theorem.

1998 ACM Subject Classification F.1.2 Modes of Computation, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Online Algorithms, Randomization, High Probability

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.470

1 Introduction

In online computation, we face the challenge of designing algorithms that work in environments where parts of the input are not known while parts of the output are already needed. The standard way of evaluating the quality of online algorithms is by means of *competitive analysis*, where one compares the outcome of an online algorithm to the optimal solution constructed by a hypothetical optimal offline algorithm. Since deterministic strategies are often proven to fail for the most prominent problems, randomization is used as a powerful tool to construct high-quality algorithms that outperform their deterministic counterparts against an oblivious adversary. These algorithms base their computations on the outcome of a random source; for a detailed introduction to online problems we refer the reader to the literature [5].

The most common way to measure the performance of randomized algorithms is to analyze the worst-case expected outcome and to compare it to the optimal solution. With offline algorithms, a statement about the expected outcome is also a statement about the *outcome with high probability* due to Markov's inequality and the fact that the algorithm may be executed many times to amplify the probability of success [11]. However, this amplification is not possible in online settings. As online algorithms only have one attempt to compute a

* The research is partially funded by the SNF grant 200021–146372, Deutsche Forschungsgemeinschaft grant BL511/10-1, and the VEGA 1/0671/11 grant.



© Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke; licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 470–481



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



reasonably good result, a statement with respect to the expected value of their competitive ratio may be rather unsatisfying. As a matter of fact, for a fixed input, it might be the case that such an algorithm produces results of a very high quality in very few cases (i. e., for a rather small number of random choices), but is unacceptably bad for the majority of random computations; still, the expected competitive ratio might suggest a better performance. Thus, if we want to have a certain guarantee that some randomized online algorithm obtains a particular quality, we must have a closer look at its analysis. In such a setting, we would like to state that the algorithm does not only perform well on average, but "almost always."

Besides a theoretical formalization of the above statement, the main contribution of this paper is to show that, for a broad class of problems, the existence of a randomized online algorithm that performs well in expectation immediately implies the existence of a randomized online algorithm that is virtually as good with high probability. Our investigations, however, need to be detailed in order to face the particularities of the framework. First, we show that it is not possible to measure the probability of success with respect to the input size, which might be considered the straightforward approach. Many of the known randomized online algorithms are naturally divided into some kind of *phases* (e. g., the algorithm for metrical task systems from Borodin et al. [6], the marking algorithm for paging from Fiat et al. [8], etc.) where each phase is processed and analyzed separately. Since the phases are independent, a high probability result (i. e., with a probability converging to 1 with an increasing number of phases) can be obtained. However, the definition of these phases is specific to each problem and algorithm. Also, there are other algorithms (e. g., the optimal paging algorithm from Achlioptas et al. [2] and many workfunction-based algorithms) that use other constructions and that are not divided into phases. As we want to establish results with high probability that are independent of the concrete algorithms, we thus have to measure this probability with respect to another parameter; we show that the cost of an optimal solution is a very reasonable quantity for this purpose. Then again it turns out that, if we consider general online problems, the notions of the expected outcome and an outcome with high probability are still not related in any way, i. e., we define problems for which these two measures are incomparable. Hence, we carefully examine both to which parameter the probability should relate and which properties we need the studied problem to fulfill to again allow a division into independent phases; finally, this allows us to construct randomized online algorithms that perform well with a probability tending to 1 with a growing size of the optimal cost. We show that this technique is applicable for a wide range of online problems.

Classically, results concerning randomized online algorithms commonly analyze their expected behavior; there are, however, a few exceptions, e. g., Leonardi et al. [14] analyze the tail distribution of algorithms for call control problems, and Maggs et al. [15] deal with online distributed data management strategies that minimize the congestion in certain network topologies.

Overview

In Section 2, we define the class of symmetric online problems and present the main result (Theorem 8). The theorem states that, for any symmetric problem that fulfills certain natural conditions, it is possible to transform an algorithm with constant expected competitive ratio r to an algorithm having a competitive ratio of $(1 + \varepsilon)r$ with high probability (with respect to the cost of an optimal solution). Section 3 is devoted to proving Theorem 8. We partition the run of the algorithm into phases such that the loss incurred by the phase changes can be amortized; however, to control the variance within one phase, we need to further subdivide the phases. Modelling the cost of single phases as dependent random variables, we obtain a

supermartingale that enables us to apply the Azuma-Hoeffding inequality and thus to obtain the result. After these investigations, we provide applications of the theorem in Section 4 where we show that our result is applicable for task systems. For the k-server problem on unbounded metric spaces and for makespan scheduling, we show that no comparable result can be obtained. We further elaborate the necessity of the conditions in Section 5. The outcome is that, even though the conditions may appear strong, a weakening prohibits a result of the same generality.

Due to space restrictions, many of the proofs are omitted and can be found in the technical report [12].

2 Preliminaries

In this section, we fix the notation for online algorithms that we use throughout the paper. Before we start, we need to briefly discuss the way in which online problems and instances are formally defined. For our investigations, we have to be very careful about these definitions. In particular, in the literature one often refers to "an online problem" when really a class of online problems is meant, which is parameterized by some problem-specific parameters. Let us give a few examples of problems that we study later in the paper. When speaking about the paging problem, we really mean the class of paging problems for, e. g., different cache sizes k. Note that there is some inconsistency in the literature as this problem is usually referred to as "paging" (and not "k-paging") while we speak of the "k-server problem." Here, k denotes the number of servers that are moved in a metric space. However, k alone is neither sufficient to specify a member from the class of paging problems nor of k-server problems. For paging, we also need the number of pages that may be requested in total, say N; for k-server the metric space (M, d) must be known, where M is a set of points and d is a distance function.

To define the above problems entirely, we still need to give even more information by speaking about how problem instances are initialized according to the parameters. For example, we need to specify how the cache is initialized for the paging problem or where the servers are located at the beginning when dealing with the k-server problem. We call this initialization the *initial situation*; for paging, the initial situation is a k-tupel of distinct integers between 1 and N, which formalizes which pages are in the cache at the beginning. Formally, we thus have to speak of an instance of the $((k, N), (s_1, \ldots, s_k))$-paging problem. In general, such a parameterized online problem is given by $(\mathcal{C}, \mathcal{I})$-$P$ where $\mathcal{C}$ is a sequence of problem-specific parameters, $\mathcal{I}$ is a set of valid initial situations, and P is the name of the union of all of theses problems. Formally, $\mathcal{I}$ is a set of valid assignments I to some of the parameters in $\mathcal{C}$ and the competitiveness guarantees of any algorithm for P must be satisfied for any $I \in \mathcal{I}$; note that, sometimes, I is also considered a part of the input instance. To end this discussion, note that in the literature, the initial situation is at times called the initial configuration; in this paper, we choose another name to distinguish it from the configuration of an algorithm (Turing machine). In the following, we will use the notation as used in the literature and omit $\mathcal{C}$; however, the initial situation I plays an important role for us and it is given together with the actual input sequence x. Let us emphasize that, if we say that some algorithm has some specific performance for a problem P, this means that this performance must be guaranteed for *all* feasible choices of $\mathcal{C}$, I, and x.

We are now ready to define online algorithms on initial situations and input instances. To keep the presentation concise, we focus on minimization problems. The same ideas translate to maximization problems by changing the point of view: we show that any optimal solution

has a low profit instead of showing that the algorithmic solution has low cost. An *online algorithm* $\mathtt{A}$ computes the output sequence $\mathtt{A}(I,x) = y = (y_1,\ldots,y_n)$, where I is an initial situation, $x = (x_1,\ldots,x_n)$ is an input sequence, and $y_i = f(I,x_1,\ldots,x_i)$ for some function f. The *cost* of the solution $\mathtt{A}(I,x)$ is denoted by $\mathrm{Cost}(I,x,y) = \mathrm{Cost}(I,x,\mathtt{A}(I,x))$. For the ease of presentation, we refer to the tuple that consists of the initial situation and the input sequence, i. e., (I,x), as the input of the problem; also, we abbreviate $\mathrm{Cost}(I,x,\mathtt{A}(I,x))$ by $\mathrm{Cost}(\mathtt{A}(I,x))$. As already mentioned, the notion of an initial situation plays an important role in the relationship between different variants of the competitive ratio; although it is usually omitted, our definition imposes no restriction on the studied problems and algorithms.

A *randomized online algorithm* $\mathtt{R}$ computes the output sequence $\mathtt{R}^{\phi}(I,x) = y = (y_1,\ldots,y_n)$ such that y_i is computed from $\phi, I, x_1,\ldots,x_i$, where ϕ is the content of a random tape. By $\mathrm{Cost}(\mathtt{R}(I,x))$ we denote the random variable (over the probability space defined by ϕ) expressing the cost of the solution $\mathtt{R}^{\phi}(I,x)$. When dealing with randomized online algorithms we compare the expected outcome to the one of an optimal algorithm. Note that, as usual in such a setting, we only consider computable problems. In the context of this paper we assume an *oblivious adversary* and say that a randomized algorithm is r-competitive if there exists a constant α such that, for every initial situation I and input sequence x, $\mathbb{E}[\mathrm{Cost}(\mathtt{R}(I,x))] \leq r \cdot \mathrm{Cost}(\mathtt{OPT}(I,x)) + \alpha$. For formal reasons, we define the competitive ratio of any (randomized) online algorithm to be 1 if both x and y are empty.

In the sequel, we analyze the notion of *competitive ratio with high probability*. Using paging, it can be shown that it does not make sense to measure the probability with respect to the input length [12]. Then again, for the practical use of paging algorithms, the instances where also the optimal algorithm makes faults are of interest. Hence, it seems reasonable to define the term *high probability* with respect to the cost of an optimal solution. In this paper, we use a strong notion of high probability requiring the error probability to be subpolynomial.

▶ **Definition 1** (Competitive Ratio w.h.p.). A randomized online algorithm $\mathtt{R}$ is r-competitive *with high probability* (w.h.p. for short) if, for any $\beta \geq 1$, there exists a constant α such that for all initial situations and input sequences (I,x) it holds that

$$\Pr[\mathrm{Cost}(\mathtt{R}(I,x)) \geq r \cdot \mathrm{Cost}(\mathtt{OPT}(I,x)) + \alpha] \leq (2 + \mathrm{Cost}(\mathtt{OPT}(I,x)))^{-\beta}.$$

First, note that the purpose of the constant 2 on the right-hand side of the formula is to properly handle inputs with a small (possibly zero) optimum. The choice of the particular constant is somewhat arbitrary (however, it should be greater than 1) since the α term on the left-hand side hides the effects. However, the two notions of the expected and the high-probability competitiveness are incomparable [12]. Nevertheless, many real-world online problems share additional properties that guarantee a close relationship between the expected and high-probability behavior. We now focus on the cost of a solution.

▶ **Definition 2** (Partition Function). A *partition function* of an online problem is a non-negative function $\mathcal{P}$ such that, for any initial situation I, the sequence of requests $x_1,\ldots,x_n$, and the corresponding solutions $y_1,\ldots,y_n$, we have

$$\mathrm{Cost}(I,(x_1,\ldots,x_n),(y_1,\ldots,y_n)) = \textstyle\sum_{i=1}^{n} \mathcal{P}(I,x_1,\ldots,x_i;y_1,\ldots,y_i).$$

In other words, for a problem with a partition function, the cost of a solution is the sum of the costs of particular answers, and the cost of each answer is independent of the future input and output. The partition function allows us to speak of the cost of a subsequence of the outputs. Note that any online problem for which the input instance may stop after each request has either a unique partition function or none, because the overall cost is fixed after

each answer. In what follows, we further restrict the behavior, and it will be convenient to think in terms of the "cost of a particular answer." We may think of online problems that have a partition function as a separate class of problems. However, all further properties depend on specific partition functions and thus requiring a "partitionability" property would be redundant.

▶ **Definition 3** (Request-Boundedness)**.** An online problem P is called *request-bounded* if, for some constant F, it has a partition function $\mathcal{P}$ such that

$$\forall I, x, y, i\colon \mathcal{P}(I, x_1, \ldots, x_i; y_1, \ldots, y_i) \leq F \text{ or } \mathcal{P}(I, x_1, \ldots, x_i;\, y_1, \ldots, y_i) = \infty.$$

We say that P is *request-bounded* according to $\mathcal{P}$.

Note that for any problem with a partition function there is a natural notion of a state; for instance, it is the content of the memory for the paging problem, the position of the servers for the k-server problem, etc. Now we provide a general definition of this notion. By $a \cdot b$, we denote the concatenation of two sequences a and b; λ denotes the empty sequence. An input $(I, x = (x_1, x_2, \ldots, x_n))$ is *feasible* with a solution $y = (y_1, y_2, \ldots, y_n)$ if starting from I, x is a request sequence that is in accord with the problem definition and for each i, y_i is a feasible answer to the request x_i with respect to I, $(x_1, x_2, \ldots, x_{i-1})$, and $(y_1, y_2, \ldots, y_{i-1})$.

▶ **Definition 4** (State)**.** Consider a partition function $\mathcal{P}$, two initial situations I and I', two sequences of requests $x = (x_1, \ldots, x_n)$ and $x' = (x'_1, \ldots, x'_m)$, and two sequences of outputs $y = (y_1, \ldots, y_n)$ and $y' = (y'_1, \ldots, y'_m)$. The triples (I, x, y) and (I', x', y') are equivalent if, for any sequence of requests $x'' = (x''_1, \ldots, x''_p)$ and a sequence of outputs $y'' = (y''_1, \ldots, y''_p)$, the input $(I, x \cdot x'')$ is feasible with a solution $y \cdot y''$ if and only if the input $(I', x' \cdot x'')$ is feasible with a solution $y' \cdot y''$, and the cost of y'' according to $\mathcal{P}$ is the same for both solutions. A *state* s of the problem is an equivalence class over the triples (I, x, y).

Let (I, x, y) be some triple in a state s. By $\texttt{Opt}_s(x')$ we denote an output sequence y' such that $y{\cdot}y'$ is a feasible solution for the input $(I, x{\cdot}x')$ and $\mathrm{Cost}(I, x{\cdot}x', y{\cdot}y') \leq \mathrm{Cost}(I, x{\cdot}x', y{\cdot}y'')$ for any feasible solution $y \cdot y''$. Note that due to the partition function, the definition of $\texttt{Opt}_s(x')$ is independent of the chosen triple (I, x, y). We sometimes simplify notation and write $\mathrm{Cost}(\texttt{Opt}_s(x'))$ instead of $\mathrm{Cost}(I, x \cdot x', y \cdot \texttt{Opt}_s(x')) - \mathrm{Cost}(I, x, y)$, as it is sufficient to know the state s and x' in order to determine the value of the function and the other parameters are clear from the context.

▶ **Definition 5** (Initial State)**.** A state s is called an *initial* state if and only if it contains some triple (I, λ, λ).

We chose this definition of states as it covers best the properties of online computations as we need them in our main theorem. An alternative definition could use task systems with infinitely many states, but the description would become less intuitive; we will return to task systems in Section 4.1.

Intuitively, a state from Definition 4 encapsulates all information about the ongoing computation of the algorithm that is relevant for evaluating the efficiency of the future processing. Usually, the state is naturally described in the problem-specific domain (content of cache, current position of servers, set of jobs accepted so far, etc.). Similar to our discussion on initial situations, we want to emphasize that a state is independent of the concrete algorithm. The internal state of an algorithm (Turing machine), which is a part of its configuration, is a different notion since it may, e. g., behave differently if the starting request had some particular value. The following properties are crucial for our approach to probability amplification.

▶ **Definition 6** (Opt-Boundedness). An online problem is called *opt-bounded* if there exists a constant B and a partition function such that $\forall s, s', x\colon |\mathrm{Cost}(\mathtt{OPT}_s(x)) - \mathrm{Cost}(\mathtt{OPT}_{s'}(x))| \leq B$.

▶ **Definition 7** (Symmetric Problem). An online problem is called *symmetric* if it is has a partition function for which every state is initial.

Note that for symmetric problems, it follows that every sequence of requests is a feasible input sequence. In particular, the input may end after any time step. Formally, any problem with a partition function may be transformed into a symmetric one simply by redefining the set of initial states. However, this transformation may significantly change the properties of the problem. Now we are going to state the main result of this paper, namely that, under certain conditions, the expected competitive ratio of symmetric problems can be achieved w.h.p.

▶ **Theorem 8.** *Consider an online problem P that is opt-bounded and symmetric according to a common partition function. Suppose there is a randomized online algorithm $\mathtt{A}$ for P with constant expected competitive ratio r. Then, for any constant $\varepsilon > 0$, there is a randomized online algorithm $\mathtt{A}'$ with competitive ratio $(1+\varepsilon)r$ w.h.p. (with respect to the optimal cost).*

3 Proof Sketch of Theorem 8

For the ease of presentation, we first provide a proof for a restricted setting where the online problem at hand is also request-bounded.

The algorithm $\mathtt{A}'$ simulates $\mathtt{A}$ and, on some specific places, performs a *reset* operation: if a part x' of the input has been read so far, and a corresponding output y' has been produced, (I, x', y') belongs to the same state as (I', λ, λ), for some initial situation I', because we are dealing with a symmetric problem; hence, $\mathtt{A}$ can be restarted by $\mathtt{A}'$ from I'.

The general idea to boost the probability of acquiring a low cost is to perform a reset each time the algorithm incurs too much cost and to use Markov's inequality to bound the probability of such an event. However, the exact value of how much is "too much" depends on the optimal cost of the input which is not known in advance. Therefore, the input is first partitioned into *phases* of a fixed optimal cost, and then each phase is cut into *subphases* based on the cost incurred so far. A reset may cause an additional expected cost of $r \cdot B$ for the subsequent phase compared to an optimal strategy starting from another state, where B is the constant of the opt-boundedness (Definition 6), i. e., B bounds the different costs between two optimal solutions for a fixed input for different states. We therefore have to ensure that the phases are long enough so as to amortize this overhead.

From now on let us consider ε, r, B, F, and α to be fixed constants; recall that F originates from the request-boundedness property of the online problem at hand (Definition 3) and α is the constant from the definition of competitiveness. The algorithm $\mathtt{A}'$ is parameterized by two parameters C and D that depend on ε, r, B, F, and α. These parameters control the lengths of the phases and subphases, respectively, such that $C + F$ delimits the optimal cost of one phase and $D + F$ delimits the cost of the solution computed by $\mathtt{A}'$ on one subphase; we require that $D > r(C + F + B + \alpha)$.

Consider an input sequence $x = (x_1, \ldots, x_n)$, an initial situation I, and let the optimal cost of the input (I, x) be between $(k-1)C$ and kC for some integer k. Then x can be partitioned into k phases $\tilde{x}_1 = (x_1, \ldots, x_{n_2-1})$, $\tilde{x}_2 = (x_{n_2}, \ldots, x_{n_3-1}), \ldots, \tilde{x}_k = (x_{n_k}, \ldots, x_n)$ in such a way that n_i is the minimal index for which the optimal cost of the input $(I, (x_1, \ldots, x_{n_i}))$ is at least $(i-1)C$. It follows that the optimal cost for one phase is at least $C - F$ and at most $C + F$, with the exception of the last phase which may be cheaper. Note that this partition

can be generated by the online algorithm itself, i. e., $\mathtt{A}'$ can determine when a next phase starts. There are only two reasons for $\mathtt{A}'$ to perform a reset: at the beginning of each phase and after incurring a cost exceeding D since the last reset. Hence, $\mathtt{A}'$ starts each phase with a reset, and the processing of each phase is partitioned into a number of subphases each of cost at least D (with the exception of the possibly cheaper last subphase) and at most $D + F$.

Now we are going to discuss the cost of $\mathtt{A}'$ on a particular input. Let us fix the input (I, x) which subsequently also fixes the indices $1 = n_1, n_2, \ldots, n_k$. Let S_i be a random variable denoting the state of the problem (according to Definition 4) just before processing request x_i, and let $W(i, j)$, $i \leq j$, be a random variable denoting the cost of $\mathtt{A}'$ incurred on the input $x_i, \ldots, x_j$. The following claim is obvious.

▶ Claim 9. If $\mathtt{A}'$ performs a reset just before processing x_i, then S_i captures all the information from the past $W(i, j)$ depends on. In particular, if we fix $S_i = s$, $W(i, j)$ does not depend on $W(l_1, l_2)$, for any $l_1 \leq l_2 \leq i$ and any state s.

The overall structure of the proof is as follows. We first show in Lemma 11 that the expected cost incurred during a phase (conditioned by the state in which the phase was entered) is at most $\mu := r(C + F + B + \alpha)/(1 - p)$, where $p := r(C + F + B + \alpha)/D < 1$. We can then consider random variables $Z_0, Z_1, \ldots, Z_k$ such that $Z_0 := k\mu$ and $Z_i := (k - i)\mu + \sum_{j=1}^{i} \overline{W}_j$ for $i > 0$, where $\overline{W}_i$ is the cost of the ith phase, clipped from above by some logarithmic bound, i. e., $\overline{W}_i := \min\{W(n_i, n_{i+1} - 1), c \log k\}$, for some suitable constant c. We show in Lemma 12 that $Z_0, Z_1, \ldots, Z_k$ form a bounded supermartingale, and then use the Azuma-Hoeffding inequality to conclude that Z_k is unlikely to be much larger than Z_0. By a suitable choice of the free parameters, this implies that Z_k is unlikely to be much larger than the expected cost of $\mathtt{A}$. Finally, we show that w.h.p. Z_k is the cost of the algorithm $\mathtt{A}'$. In order to argue about the expected cost of a given phase in Lemma 11, let us first show that a phase is unlikely to have many subphases. For the rest of the proof, let X_j be the random variable denoting the number of subphases of phase j.

▶ **Lemma 10.** *For any i, s, and any $\delta \in \mathbb{N}$ we have* $\Pr[X_i \geq \delta \mid S_{n_i} = s] \leq p^{\delta - 1}$.

Now we can argue about the expected cost of a phase.

▶ **Lemma 11.** *For any i and s it holds that* $\mathbb{E}[W(n_i, n_{i+1} - 1) \mid S_i = s] \leq \mu$.

Once the expected cost of a phase is established, we can construct the supermartingale as follows.

▶ **Lemma 12.** *For any constant $c > 0$, the sequence $Z_0, \ldots, Z_k$ is a supermartingale.*

Proof. Consider a fixed c. We have to show that for each i, $\mathbb{E}[Z_{i+1} \mid Z_0, \ldots, Z_i] \leq Z_i$. From the definition of the Z_i's it follows that $Z_{i+1} - Z_i = \overline{W}_{i+1} - \mu$. Consider any elementary event ξ from the probability space, and let $Z_i(\xi) = z_i$, for $i = 0, \ldots, k$, be the values of the corresponding random variables. We have

$$\begin{aligned}
&\mathbb{E}[Z_{i+1} \mid Z_0, \ldots, Z_i](\xi) = \mathbb{E}[Z_{i+1} \mid Z_0 = z_0, \ldots, Z_i = z_i] \\
&\quad = \mathbb{E}[Z_i + \overline{W}_{i+1} - \mu \mid Z_0 = z_0, \ldots, Z_i = z_i] \\
&\quad = z_i - \mu + \mathbb{E}[\overline{W}_{i+1} \mid Z_0 = z_0, \ldots, Z_i = z_i] \\
&\quad = z_i - \mu + \textstyle\sum_s \mathbb{E}[\overline{W}_{i+1} \mid Z_0 = z_0, \ldots, Z_i = z_i, S_{n_{i+1}} = s] \\
&\qquad\quad \cdot \Pr[S_{n_{i+1}} = s \mid Z_0 = z_0, \ldots, Z_i = z_i] \\
&\quad \leq z_i - \mu + \textstyle\sum_s \mathbb{E}[W(n_{i+1}, n_{i+2} - 1) \mid S_{n_{i+1}} = s] \cdot \Pr[S_{n_{i+1}} = s \mid Z_0 = z_0, \ldots, Z_i = z_i] \\
&\quad \leq z_i - \mu + \mu \textstyle\sum_s \Pr[S_{n_{i+1}} = s \mid Z_0 = z_0, \ldots, Z_i = z_i] = z_i = Z_i(\xi),
\end{aligned}$$

where the last inequality is a consequence of Lemma 11. ◀

Now we use the following special case of the Azuma-Hoeffding inequality [1, 10].

▶ **Lemma 13** (Azuma, Hoeffding). *Let $Z_0, Z_1, \dots$ be a supermartingale, such that $|Z_{i+1} - Z_i| < \gamma$. Then for any positive real t, $\Pr[Z_k - Z_0 \geq t] \leq \exp\left(-t^2/(2k\gamma^2)\right)$.*

In order to apply Lemma 13, we need the following bound.

▶ Claim 14. Let k be such that $c \log k > \mu$. For any i it holds that $|Z_{i+1} - Z_i| < c \log k$.

We are now ready to prove the subsequent lemma.

▶ **Lemma 15.** *Let k be such that $c \log k > \mu$. There is a constant C (depending on F, B, ε, r, α) such that $\Pr[Z_k \geq (1+\varepsilon)rkC] \leq \exp\left(-\left(k\left((1+\varepsilon)rC - \mu\right)^2\right)/(2c^2 \log^2 k)\right)$.*

Proof. Applying Lemma 13 for any positive t, we get

$$\Pr[Z_k - Z_0 \geq t] \leq \exp\left(-\frac{t^2}{2kc^2 \log^2 k}\right).$$

Noting that $Z_0 = k\mu$ and choosing $t := k((1+\varepsilon)rC - \mu)$ the statement follows. The only remaining task is to verify that $t > 0$, which can be shown by some simple calculations [12]. ◀

To prove the claim of the main theorem, we show the following bound.

▶ **Lemma 16.** *For any c and $\beta > 1$ there is a k_0 such that for any $k > k_0$*

$$\exp\left(-\frac{k\left((1+\varepsilon)rC - \mu\right)^2}{2c^2 \log^2 k}\right) \leq \frac{1}{2(2+kC)^\beta}.$$

Proof. Note that the left-hand side is of the form $\exp(-\eta k/\log^2 k)$ for some positive constant η. Clearly, for any $\beta > 1$ and large enough k, it holds that $\exp(\eta k/\log^2 k) \geq 2(2+kC)^\beta$. ◀

Combining Lemmata 15 and 16, we get the following result.

▶ **Corollary 17.** *There is a constant C (depending on F, B, ε, r, α) such that for any $\beta > 1$ there is a k_0 such that for any $k > k_0$ we have*

$$\Pr[Z_k \geq (1+\varepsilon)rkC] \leq 1/(2(2+kC)^\beta).$$

To finish the proof of the theorem we show that w.h.p. Z_k is actually the cost of the algorithm $\mathtt{A}'$.

▶ **Lemma 18.** *For any $\beta > 1$ there is a c and a k_1 such that for any $k > k_1$ $\Pr[Z_k \neq \mathrm{Cost}(\mathtt{A}'(I,x))] \leq 1/(2(2+kC)^\beta)$.*

Proof. Since $Z_k = \sum_{j=1}^{k} \min\{W(n_j, n_{j+1}-1), c\log k\}$ the event that $Z_k \neq \mathrm{Cost}(\mathtt{A}'(I,x))$ happens exactly when there is some j such that $W(n_j, n_{j+1}-1) > c\log k$. Consider any fixed j. Since the cost of a subphase is at most $D+F$, it holds that $W(n_j, n_{j+1}-1) \leq X_j(F+D)$. From Lemma 10 it follows that for any c,

$$\Pr[W(n_j, n_{j+1}-1) > c\log k] \leq \Pr\left[X_j \geq \left\lceil \frac{c\log k}{F+D} \right\rceil\right] \leq p^{\frac{c\log k}{F+D}-1}.$$

Consider the function

$$g(k) := \frac{\log\left(\frac{2k}{p}(2+kC)^\beta\right)}{\log k}.$$

It is decreasing, and $\lim_{k\mapsto\infty} g(k) = 1+\beta$. Hence, it is possible to find a constant c and a k_1 such that for any $k > k_1$ it holds that

$$c \geq \frac{F+D}{\log(1/p)} \cdot g(k).$$

From that we obtain

$$(1/p)^{\frac{c\log k}{F+D}-1} \geq 2k(2+kC)^{\beta}.$$

Thus, for this choice of c and k_1, it holds that $\Pr[W(n_j, n_{j+1}-1) > c\log k] \leq p^{\frac{c\log k}{F+D}-1} \leq 1/(2k(2+kC)^{\beta})$. Using the union bound, we conclude that the probability that the cost of any phase exceeds $c\log k$ is at most $1/(2(2+kC)^{\beta})$. ◀

Using the union bound, combining Lemma 18 and Corollary 17, and noting that the cost of the optimum is at most kC, we get the following statement.

▶ **Corollary 19.** *There is a constant C such that for any $\beta > 1$ there is a k_2 such that for any $k > k_2$ we have*

$$\Pr[\mathrm{Cost}(\mathtt{A}'(I,x)) \geq (1+\varepsilon)r\mathrm{Cost}(\mathtt{OPT}(I,x))] \leq (2+kC)^{-\beta}.$$

To conclude the proof by showing that for any $\beta > 1$ there is some α' such that

$$\Pr[\mathrm{Cost}(\mathtt{A}'(I,x)) > (1+\varepsilon)r\mathrm{Cost}(\mathtt{OPT}(I,x)) + \alpha'] \leq (2+kC)^{-\beta}$$

holds for all k, we have to choose α' large enough to cover the cases of $k < k_2$. For these cases, $\mathrm{Cost}(\mathtt{OPT}(I,x)) < k_2 C$, and hence the expected cost of $\mathtt{A}$ is at most rk_2C, and due to Lemma 11, the expected cost of $\mathtt{A}'$ is constant. The right-hand side $(2+kC)^{-\beta}$ is decreasing in k, so it is at least $(2+k_2C)^{-\beta}$, which is again a constant. From Markov's inequality it follows that there exists a constant α' such that $\Pr[\mathrm{Cost}(\mathtt{A}'(I,x)) > \alpha'] < (2+k_2C)^{-\beta}$ finishing the proof of the restricted setting.

3.1 Avoiding Request-Boundedness

All that is left to do is to show how to handle problems that are not request-bounded [12]. The main idea is to apply the restricted Theorem 8 to a modified request-bounded version of the given problem. We show that there is a modified version of the algorithm such that the computed solution has an expected competitive ratio matching the original one for the modified problem. By ensuring that *any* solution to the modified problem translates to a solution of the original problem with at most the same competitive ratio, it is enough to apply our theorem to the modified problem to obtain an analogous result for the original problem.

4 Applications and Lower Bounds

We now discuss the impact of Theorem 8 on task systems, the k-server problem, and paging. Despite being related, these problems have different flavors when analyzing them in the context of high probability results. We show that makespan scheduling does not allow for similar results.

4.1 Task Systems

The properties of online problems needed for Theorem 8 are related to the definition of task systems. There are, however, some important differences.

To analyze the relation, let us recall the definition of task systems as introduced by Borodin et al. [6]. We are given a finite state space S and a function $d\colon S \times S \to \mathbb{R}_+$ that specifies the (finite) cost to move from one state to another. The requests given as input to a task system are a sequence of $|S|$-vectors that specify, for each state, the cost to process the current task if the system resides in that state. An online algorithm for task systems aims to find a schedule such that the overall cost for transitions and processing is minimized. From now on we will call states in S *system states* to distinguish them from the states of Definition 4. The main difference between states of Definition 4 and system states is that states depend on the sequence of requests and answers; this way there may be infinitely many states. States are also more general than system states in that specific state transitions may be impossible.

▶ **Theorem 20.** *Let* $\mathtt{A}$ *be a randomized online algorithm with expected competitive ratio* r *for task systems. Then, for any* $\varepsilon > 0$*, there is a randomized online algorithm* $\mathtt{A}'$ *for task systems with competitive ratio* $(1+\varepsilon)r$ *w.h.p. (with respect to the optimal cost).*

4.2 The *k*-Server Problem

The k-server problem, introduced by Manasse et al. [16], is concerned with the movement of k servers in a metric space. Each request is a location and the algorithm has to move one of the servers to that location. If the metric space is finite, this problem is well known to be a special metrical task system. Recent progress by Bansal et al. [3] suggests that randomization might lead to an expected competitive ratio exponentially better than the deterministic lower bound.

Theorem 20 directly implies that all algorithms with a constant expected competitive ratio for the k-server problem in a finite metric space can be transformed into algorithms that have almost the same competitive ratio w.h.p.

If the metric space is infinite, an analogous result is still valid except that we have to bound the maximum transition cost by a constant. This is the case, because the proof of Theorem 20 uses the finiteness of the state space only to ensure bounded transition costs. Without the restriction to bounded distances, in general we cannot obtain a competitive ratio much better than the deterministic one w.h.p.

▶ **Theorem 21.** *Let* (M,d) *be a metric space with* $|M| = N$ *constant,* $s \in M$ *be the initial position of all servers,* ℓ *a constant and let* r *be the infimum over the competitive ratios of all deterministic online algorithms for the* k*-server problem in* (M,d) *for instances with at most* ℓ *requests. For every* $\varepsilon > 0$*, there is a metric space* (M',d') *where for any randomized online algorithm* $\mathtt{R}$ *for the* k*-server problem there is an oblivious adversary against which the solution of* $\mathtt{R}$ *has a competitive ratio of at least* $r - \varepsilon$ *with constant probability.*

▶ **Corollary 22.** *If we allow the metric to be infinite, then there is no* $(k-\varepsilon)$*-competitive online algorithm w.h.p. for the* k*-server problem for any constant* ε*.*

We simply use that the lower bound of Manesse et al. [16] satisfies the properties of Theorem 21.

4.3 Paging

Analogous to the k-server problem also the paging problem allows for the application of Theorem 8. Thus for any paging algorithm with expected competitive ratio r there is an algorithm with competitive ratio $r(1+\varepsilon)$ w.h.p.

Note that the marking algorithm is analyzed based on phases that correspond to $k+1$ distinct requests, and hence the analysis of the expected competitive ratio immediately gives the $2H_k - 1$ competitive ratio also w.h.p. However, e. g., the optimal algorithm with competitive ratio H_k due to Achlioptas et al. [2] is a distribution-based algorithm where the high probability analysis is not immediate; Theorem 8 gives an algorithm with competitive ratio $H_k(1+\varepsilon)$ w.h.p. also in this case.

4.4 Makespan Scheduling

Let us consider the classical online makespan scheduling problem $\mathrm{P}||\mathrm{C}_{\max}$ where jobs arrive one by one. It is well known that there is a tight deterministic bound $2 - 1/m$ for $m \in \{2, 3\}$ on the competitive ratio, where m is the number of machines [9, 7]. Similar to Theorem 21, we can show the following.

▶ **Theorem 23.** *For any m, let ℓ, k be constants depending on α and let r be the infimum over the competitive ratios of all deterministic online algorithms for the online makespan scheduling problem with m machines for instances with at most ℓ requests such that each request is a job with an integer processing time at most k. Then for any constant $\varepsilon > 0$, the lower bound on the competitive ratio w.h.p. is at least $r - \varepsilon$.*

The restriction to integers does not weaken the result, as we may choose a suitable scaling factor of the processing costs that allows to hide the deviation in the ε. In particular, for $m = 2$ we already obtain the tight bound on the competitive ratio for $\ell = 3$ [7] and thus there is no $(3/2 - \varepsilon)$-competitive algorithm w.h.p. for $m = 2$ and any constant ε whereas there is an online algorithm with an expected competitive ratio of $4/3$ [4].

5 Necessity of Requirements

As mentioned above, our result holds with large generality as many well-studied online problems meet the requirements we imposed. However, the assumptions of Theorem 8 require that for the problem at hand (1) every state is initial, and (2) $\forall s, s', x\colon |\mathrm{Cost}(\mathtt{OPT}_s(x)) - \mathrm{Cost}(\mathtt{OPT}_{s'}(x))| \leq B$. We can show that removing any of the conditions (1) and (2) allows for counterexamples to the theorem [12].

We would like to emphasize that the applicability of Theorem 8 is a property of problems and not of algorithms. There are problems that do not fit the assumptions of the theorem and still can be solved almost optimally by specific randomized online algorithms with high probability; for instance, albeit using a weaker notion of high probability than in the previous sections, online flow shop scheduling with unit-length tasks, $\mathrm{F}|p_{ij} = 1|\mathrm{C}_{\max}$, allows for such algorithms with respect to the number of machines [12].

Acknowledgment. The authors want to express their deepest thanks to Georg Schnitger who gave some very important impulses that contributed to the results of this paper.

References

1 K. Azuma. Weighted sums of certain dependent random variables. *Tôhoku Mathematical Journal*, 19(3):357–367, 1967.
2 D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1-2):203–218, 2000.
3 N. Bansal, N. Buchbinder, A. Mądry, and J. Naor. A polylogarithmic-competitive algorithm for the k-server problem (extended abstract). In *Proc. of FOCS 2011*, pages 267–276, 2011.
4 Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New Algorithms for an Ancient Scheduling Problem. *J. Comput. Syst. Sci.*, 51(3):359–366, 1995.
5 A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
6 A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM*, 39(4):745–763, 1992.
7 U. Faigle, W. Kern, and G. Turán. On the performance of on-line problems for partition problems. *Acta Cybern.*, 9(2):107–119, 1989.
8 A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.
9 R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Tech. J.*, 45:1563–1581, 1966.
10 W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
11 J. Hromkovič. *Design and analysis of randomized algorithms*. Springer-Verlag, Berlin, 2005.
12 D. Komm, R. Královič, R. Královič, and T. Mömke: Randomized online computation with high probability guarantees. CoRR abs/1302.2805, 2013.
13 E. Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.
14 S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosén. On-line randomized call control revisited. *SIAM Journal on Computing*, 31(1):86–112, 2001.
15 B. M. Maggs, F. Meyer auf der Heide, B. Voecking, and M. Westermann. Exploiting locality for networks of limited bandwidth. In *Proc. of FOCS 1997*, pages 284–293, 1997.
16 M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive Algorithms for On-line Problems. *Journal of Algorithms*, 11(2):208–230, 1990.
17 D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

An optimal quantum algorithm for the oracle identification problem

Robin Kothari

David R. Cheriton School of Computer Science and Institute for Quantum Computing, University of Waterloo, Waterloo, Canada
rkothari@cs.uwaterloo.ca

Abstract

In the oracle identification problem, we are given oracle access to an unknown N-bit string x promised to belong to a known set $\mathcal{C}$ of size M and our task is to identify x. We present a quantum algorithm for the problem that is optimal in its dependence on N and M. Our algorithm considerably simplifies and improves the previous best algorithm due to Ambainis et al. Our algorithm also has applications in quantum learning theory, where it improves the complexity of exact learning with membership queries, resolving a conjecture of Hunziker et al.

The algorithm is based on ideas from classical learning theory and a new composition theorem for solutions of the filtered γ_2-norm semidefinite program, which characterizes quantum query complexity. Our composition theorem is quite general and allows us to compose quantum algorithms with input-dependent query complexities without incurring a logarithmic overhead for error reduction. As an application of the composition theorem, we remove all log factors from the best known quantum algorithm for Boolean matrix multiplication.

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

Keywords and phrases quantum algorithms, quantum query complexity, oracle identification

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.482

1 Introduction

Query complexity is a model of computation where quantum computers are provably better than classical computers. Some of the great breakthroughs of quantum algorithms have been conceived in this model (e.g., Grover's algorithm [11]). In this paper we study the query complexity of the oracle identification problem, the very basic problem of completely determining a string given oracle access to it.

In the oracle identification problem, we are given an oracle for an unknown N-bit string x, promised to belong to a known set $\mathcal{C} \subseteq \{0,1\}^N$, and our task is to identify x while minimizing the number of oracle queries. For a set $\mathcal{C}$, we denote this problem $\textsc{oip}(\mathcal{C})$. As usual, classical algorithms have access to an oracle that outputs x_i on input i, while quantum algorithms have access to a unitary O_x that maps $|i,b\rangle$ to $|i,b \oplus x_i\rangle$ for $b \in \{0,1\}$. For a function $f : D \to E$, where $D \subseteq \{0,1\}^N$, let $Q(f)$ denote the bounded-error quantum query complexity of computing $f(x)$. Then $\textsc{oip}(\mathcal{C})$ corresponds to computing the identity function $f(x) = x$ with $D = E = \mathcal{C}$.

For example, let $\mathcal{C}_N := \{0,1\}^N$. Then the classical query complexity of $\textsc{oip}(\mathcal{C}_N)$ is N, since every bit needs to be queried to learn x, even with bounded error. A surprising result of van Dam shows that $Q(\textsc{oip}(\mathcal{C}_N)) = N/2 + O(\sqrt{N})$ [19]. As another example, consider the set $\mathcal{C}_{\mathrm{H1}} = \{x : |x| = 1\}$, where $|x|$ is the Hamming weight of x. This is the search problem with 1 marked item and thus $Q(\textsc{oip}(\mathcal{C}_{\mathrm{H1}})) = \Theta(\sqrt{N})$ [6, 11].



© Robin Kothari;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 482–493

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


Due to the generality of the problem, it has been studied in contexts such as quantum query complexity [1, 2], quantum machine learning [18, 5, 13] and post-quantum cryptography [8]. Several well-known problems are special cases of oracle identification, e.g., the search problem with one marked item [11], the Bernstein-Vazirani problem [7], the oracle interrogation problem [19] and hidden shift problems [20]. For some applications, generic oracle identification algorithms are almost as good as algorithms tailored to the specific application [9]. Consequently, this result improves some of the upper bounds stated in [9].

Ambainis et al. [1, 2] studied the oracle identification problem in terms of N and $M := |\mathcal{C}|$. They exhibited algorithms whose query complexity is close to optimal in its dependence on N and M. For a given N and M, we say an oracle identification algorithm is optimal in terms of N and M if it solves all N-bit oracle identification problems with $|\mathcal{C}| = M$ making at most Q queries and there exists some N-bit oracle identification problem with $|\mathcal{C}| = M$ that requires $\Omega(Q)$ queries. This does not, however, mean that the algorithm is optimal for each set $\mathcal{C}$ individually, since these two parameters do not completely determine the query complexity of the problem. For example, all oracle identification problems with $M = N$ can be solved with $O(\sqrt{N})$ queries, and this is optimal since this class includes the search problem with 1 marked item ($\mathcal{C}_{\mathrm{H1}}$ above). However there exists a set $\mathcal{C}$ of size $M = N$ with query complexity $\Theta(\log N)$, such as the set of all strings with arbitrary entries in the first $\log N$ bits and zeroes elsewhere.

Let $\textsc{oip}(M, N)$ denote the set of oracle identification problems with $\mathcal{C} \subseteq \{0,1\}^N$ and $|\mathcal{C}| = M$. Let the query complexity of $\textsc{oip}(M, N)$ be the maximum query complexity of any problem in that set. Then the classical query complexity of $\textsc{oip}(M, N)$ is easy to characterize:

▶ Proposition 1. The classical (bounded-error) query complexity of $\textsc{oip}(M, N)$ is $\Theta(\min\{M, N\})$.

For $M \leq N$, the upper bound follows from the observation that we can always eliminate at least one potential string in $\mathcal{C}$ with one query. For the lower bound, consider any subset of $\mathcal{C}_{\mathrm{H1}}$ of size M. For $M > N$, the lower bound follows from any set $\mathcal{C} \supseteq \mathcal{C}_{\mathrm{H1}}$ and the upper bound is trivial since any query problem can be solved with N queries.

Now that the classical query complexity is settled, we can move to quantum query complexity. When quantum queries are permitted, the $M \leq N$ case is fully understood. For a lower bound, we consider (as before) any subset of $\mathcal{C}_{\mathrm{H1}}$ of size M, which is as hard as the search problem on M bits and requires $\Omega(\sqrt{M})$ queries. For an upper bound, we can reduce this to the case of $M = N$ by selecting M bits such that the strings in $\mathcal{C}$ are distinct when restricted to these bits. (A proof of this fact appears in [9, Theorem 11].) Thus $Q(\textsc{oip}(M, N)) \leq Q(\textsc{oip}(M, M))$, which is $O(\sqrt{M})$ [1, Theorem 3].

▶ Proposition 2. For $M \leq N$, $Q(\textsc{oip}(M, N)) = \Theta(\sqrt{M})$.

For the hard regime, where $M > N$, the best known lower and upper bounds are the following, from [1, Theorem 2] and [2, Theorem 2] respectively.

▶ **Theorem 1** ([1, 2]). *If $N < M \leq 2^{N^d}$ for some constant $d < 1$, then $Q(\textsc{oip}(M, N)) = O(\sqrt{N \log M / \log N})$ and for all $M > N$, $Q(\textsc{oip}(M, N)) = \Omega(\sqrt{N \log M / \log N})$.*

When M gets closer to 2^N, their algorithm no longer gives nontrivial upper bounds. For example, if $M \geq 2^{N/\log N}$, their algorithm makes $O(N)$ queries. While not stated explicitly, an improved algorithm follows from the techniques of [3, Theorem 6], but the improved algorithm also does not yield a nontrivial upper bound when $M \geq 2^{N/\log N}$. Ambainis et al. [2] left open two problems, in increasing order of difficulty: to determine whether it is always possible to solve the oracle identification problem for $M = 2^{o(N)}$ using $o(N)$ queries and to design a single algorithm that is optimal in the entire range of M.

In this paper we resolve both open problems by completely characterizing the quantum query complexity of the oracle identification problem in the full range $N < M \leq 2^N$.

▶ **Theorem 2.** *For $N < M \leq 2^N$, $Q(\text{OIP}(M,N)) = \Theta\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$.*

The lower bound follows from the ideas in [1], but needs additional calculation. We provide a proof in the full version of this paper [15]. The lower bound also appears in an unpublished manuscript [3, Remark 1]. The $+1$ term in the denominator is relevant only when M gets close to 2^N; it ensures that the complexity is $\Theta(N)$ in that regime.

Our main result is the algorithm, which is quite different from and simpler than that of [2]. It is also optimal in the full range of M as it makes $O\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$ queries when $M \geq N$ and $O(\sqrt{M})$ queries when $M \leq N$. Our algorithm has two main ingredients:

First, we use ideas from classical learning theory, where the oracle identification problem is studied as the problem of exact learning with membership queries [4]. In particular, our quantum algorithm is based on Hegedűs' implementation of the halving algorithm [12]. Hegedűs characterizes the number of queries needed to solve the classical oracle identification problem in terms of the "extended teaching dimension" of $\mathcal{C}$. While we do not use that notion, we borrow some of the main ideas. This is further explained in Section 2.

We now present a high-level overview of the algorithm. Say we know that the string in the black box, x, belongs to a set S. We can construct from S a string s, known as the "majority string," which is 1 at position i if at least half the strings in S are 1 at position i. Importantly, for any i, the set of strings in S that disagree with s at position i is at most half the size of S. Now we search for a disagreement between x and s using Grover's algorithm. If the algorithm finds no disagreement, then $x = s$. If it does, we have reduced the size of S by a factor of 2. This gives a suboptimal algorithm with query complexity $O(\sqrt{N} \log M)$. We improve the algorithm by taking advantage of two facts: first, that Grover's algorithm can find a disagreement faster if there are many disagreements to be found, and second, that there exists an order in which to find disagreements that reduces the size of S as much as possible in each iteration. The existence of such an order was shown by Hegedűs [12].

The second ingredient of our upper bound is a composition theorem for solutions of the filtered γ_2-norm semidefinite program (SDP) introduced by Lee et al. [16] that preserves input-dependent query complexities. We need such a result to resolve the following problem: Our algorithm consists of k bounded-error quantum algorithms that must be run sequentially because each algorithm requires as input the output of the previous algorithm. Let the query complexities of the algorithms be $Q_1(x), Q_2(x), \ldots, Q_k(x)$ on input x. If these were exact algorithms, we could merely run them one after the other, giving one algorithm's output to the next as input, to obtain an algorithm with worst-case query complexity $O(\max_x \sum_i Q_i(x))$. However, since these are bounded-error algorithms, we cannot guarantee that all k algorithms will give the correct output with high probability. One option is to apply standard error reduction, but this would yield an algorithm making $O(\max_x \sum_i Q_i(x) \log k)$ queries. Instead, we prove a general composition theorem for the filtered γ_2-norm SDP that gives an algorithm making $O(\max_x \sum_i Q_i(x))$ queries, as if the algorithms had no error. A similar result is known for worst-case query complexity, but that gives a suboptimal upper bound of $O(\sum_i \max_x Q_i(x))$ queries. We prove this result in Section 3.

The oracle identification problem was also studied by Atıcı and Servedio [5], who studied algorithms that are optimal for a given set $\mathcal{C}$. The query complexity of their algorithm depends on a combinatorial parameter of $\mathcal{C}$, $\hat{\gamma}^{\mathcal{C}}$, which satisfies $2 \leq 1/\hat{\gamma}^{\mathcal{C}} \leq N+1$. They prove $Q(\text{OIP}(\mathcal{C})) = O(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} \log M \log\log M)$. Our algorithm for oracle identification, without modification, makes fewer queries than this. Our algorithm makes $O\left(\sqrt{\frac{1/\hat{\gamma}^{\mathcal{C}}}{\log 1/\hat{\gamma}^{\mathcal{C}}}} \log M\right)$

queries, which resolves a conjecture of Hunziker et al. [13]. We show this in Section 4.1. Our composition theorem can also be used to remove unneeded log factors from existing quantum query algorithms. As an example, we show how to improve the almost optimal Boolean matrix multiplication algorithm that makes $O(n\sqrt{l}\,\mathrm{poly}(\log n))$ queries [14], where n is the size of the matrices and l is the output sparsity, to an algorithm with query complexity $O(n\sqrt{l})$. We show this in Section 4.2. We conclude with open questions in Section 5. Proofs omitted due to space constraints appear in the full version of this paper [15].

2 Oracle identification algorithm

In this section we explain the ideas that go into our algorithm and prove its correctness. We also prove the query upper bound assuming we can compose bounded-error quantum algorithms without incurring log factors, which we justify in Section 3.

Throughout this section, let $x \in \mathcal{C}$ be the string we are trying to identify. For any set $S \in \{0,1\}^N$, let $\textsc{maj}(S)$ be an N-bit string such that $\textsc{maj}(S)_i$ is 1 if $|\{y \in S : y_i = 1\}| \geq |\{y \in S : y_i = 0\}|$ and 0 otherwise. In words, $\textsc{maj}(S)_i$ is b if the majority of strings in S have bit i equal to b. Note that the string $\textsc{maj}(S)$ need not be a member of S. In this paper, all logarithms are base 2 and for any positive integer k, we define $[k] := \{1, 2, \ldots, k\}$.

2.1 Basic halving algorithm

We begin by describing a general learning strategy called the halving algorithm, attributed to Littlestone [17]. Say we currently know that the oracle contains a string $x \in S \subseteq \mathcal{C}$. The halving algorithm tests if the oracle string x is equal to $\textsc{maj}(S)$. If it is equal, we have identified x; if not, we look for a bit at which they disagree. Having found such a bit i, we know that $x_i \neq \textsc{maj}(S)_i$, and we may delete all strings in S that are inconsistent with this. Since at most half the strings in S disagree with $\textsc{maj}(S)$ at any position, we have at least halved the number of potential strings.

To convert this into a quantum algorithm, we need a subroutine that tests if a given string $\textsc{maj}(S)$ is equal to the oracle string x and finds a disagreement otherwise. This can be done by running Grover's algorithm on the bitwise $\textsc{xor}$ of x and $\textsc{maj}(S)$.

Algorithm 1 Basic halving algorithm

1: $S \leftarrow \mathcal{C}$
2: **repeat**
3: Search for a disagreement between x and $\textsc{maj}(S)$. If we find a disagreement, delete all inconsistent strings from S. If not, let $S \leftarrow \{\textsc{maj}(S)\}$.
4: **until** $|S| = 1$

This algorithm always finds the unknown string x, since S always contains x. The loop can run at most $\log M$ times, since each iteration cuts down the size of S by a factor of 2. Grover's algorithm needs $O(\sqrt{N})$ queries, but it is a bounded-error algorithm. For this section, let us assume that bounded-error algorithms can be treated like exact algorithms and need no error reduction. Assuming this, Algorithm 1 makes $O(\sqrt{N}\log M)$ queries.

2.2 Improved halving algorithm

Even assuming free error reduction, Algorithm 1 is not optimal. Primarily, this is because Grover's algorithm can find an index i such that $x_i \neq \textsc{maj}(S)_i$ faster if there are many such

indices to be found, and Algorithm 1 does not exploit this fact. Given an N-bit binary string, we can find a 1 with $O(\sqrt{N/K})$ queries in expectation, where $K > 0$ is the number of 1s in the string. Alternately, there is a variant of Grover's algorithm that finds the first 1 (from left to right, say) in the string in $O(\sqrt{p})$ queries in expectation where p is the position of the first 1. This follows from the known $O(\sqrt{N})$ algorithm for finding the first 1 in a string of size N [10], by running that algorithm on the first 2^k bits, for $k = 1, 2, \ldots, \log N$. We can now modify the previous algorithm to look for the first disagreement between x and $\textsc{maj}(S)$ instead of any disagreement.

Algorithm 2 Improved halving algorithm

1: $S \leftarrow \mathcal{C}$
2: **repeat**
3: Search for the first disagreement between x and $\textsc{maj}(S)$. If we find a disagreement, delete all inconsistent strings from S. If not, let $S \leftarrow \{\textsc{maj}(S)\}$.
4: **until** $|S| = 1$

As before, the algorithm always finds the unknown string. Let r be the number of times the loop repeats and $p_1, p_2, \ldots, p_r$ be the positions of disagreement found. After the first run of the loop, since a disagreement is found at position p_1, we have learned the first p_1 bits of x; the first $p_1 - 1$ bits agree with $\textsc{maj}(S)$, while bit p_1 disagrees with $\textsc{maj}(S)$. Thus we are left with a set S in which all strings agree on these p_1 bits. For convenience, we can treat S as a set of strings of length $N - p_1$ (instead of length N). Each iteration reduces the effective length of strings in S by p_i, which gives $\sum_i p_i \leq N$, since there are at most N bits to be learned. As before, the loop can run at most $\log M$ times, thus $r \leq \log M$. Finally, if we assume again that these bounded-error search subroutines are exact, this algorithm requires $O(\sum_i \sqrt{p_i})$ queries, which is $O(\sqrt{N \log M})$, by the Cauchy–Schwarz inequality.

2.3 Final algorithm

While Algorithm 2 is an improvement over Algorithm 1, it is still not optimal. One reason is that sometimes a disagreement between the majority string and x may eliminate more than half the possible strings. This observation can be exploited by finding disagreements in such a way as to maximize the reduction in size when a disagreement is found. This idea is due to Hegedűs [12].

To understand the basic idea, consider searching for a disagreement between x and $\textsc{maj}(S)$ classically. The most obvious strategy is to check if $x_1 = \textsc{maj}(S)_1$, $x_2 = \textsc{maj}(S)_2$, and so on until a disagreement is found. This strategy makes more queries if the disagreement is found at a later position. However, we could have chosen to examine the bits in any order. We would like the order to be such that if a disagreement is found at a later position, it cuts down the size of S by a larger factor. Such an ordering would ensure that either we spend very few queries and achieve a factor-2 reduction right away, or we spend more queries but the size of S goes down significantly. Hegedűs shows that there is always a reordering of the bits that achieves this. The following lemma is similar to [12, Lemma 3.2], but we provide a proof for completeness.

▶ **Lemma 3.** *For any $S \subseteq \{0,1\}^N$, there exists a string $s \in \{0,1\}^N$ and a permutation σ on N, such that for any $p \in [N]$, $|S_p| \leq \frac{|S|}{\max\{2,p\}}$, where $S_p = \{y \in S : y_{\sigma(i)} = s_{\sigma(i)}$ for $1 \leq i \leq p-1$ and $y_{\sigma(p)} \neq s_{\sigma(p)}\}$, the set of strings in S that agree with s at $\sigma(1), \ldots, \sigma(p-1)$ and disagree with it at $\sigma(p)$.*

Proof. We will construct the permutation σ and string s greedily, starting with the first position, $\sigma(1)$. We choose this bit to be one that intuitively contains the most information, i.e., a bit for which the fraction of strings that agree with the majority is closest to $1/2$. This choice will make $|S_1|$ as large as possible. More precisely, we choose $\sigma(1)$ to be any j that maximizes $|\{y \in S : y_j \neq \textsc{maj}(S)_j\}|$. Then let $s_{\sigma(1)}$ be $\textsc{maj}(S)_{\sigma(1)}$.

In general, after having chosen $\sigma(1), \ldots, \sigma(k-1)$ and having defined s on those bits, we choose $\sigma(k)$ to be the most informative bit assuming all previous bits have agreed with string s on positions $\sigma(1), \ldots, \sigma(k-1)$. This choice makes $|S_k|$ as large as possible. More precisely, define $\bar{S}_p = \{y \in S : y_{\sigma(i)} = s_{\sigma(i)} \text{ for all } 1 \leq i \leq p\}$. We choose $\sigma(k)$ to be any bit j that maximizes $|\{y \in \bar{S}_{k-1} : y_j \neq \textsc{maj}(\bar{S}_{k-1})_j\}|$. Then let $s_{\sigma(k)}$ be $\textsc{maj}(\bar{S}_{k-1})_{\sigma(k)}$.

This construction ensures that $|S_1| \geq |S_2| \geq \ldots \geq |S_N|$. Since $\sigma(k)$ was chosen to maximize $|\{y \in \bar{S}_{k-1} : y_j \neq \textsc{maj}(\bar{S}_{k-1})_j\}|$, we have $|S_k| = |\{y \in \bar{S}_{k-1} : y_{\sigma(k)} \neq \textsc{maj}(\bar{S}_{k-1})_{\sigma(k)}\}| \geq |\{y \in \bar{S}_{k-1} : y_{\sigma(k+1)} \neq \textsc{maj}(\bar{S}_{k-1})_{\sigma(k+1)}\}|$. The size of this set is at least $|\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq \textsc{maj}(\bar{S}_{k-1})_{\sigma(k+1)}\}|$, since $\bar{S}_k \subseteq \bar{S}_{k-1}$. We do not know the value of $\textsc{maj}(\bar{S}_{k-1})_{\sigma(k+1)}$ (e.g., it need not be equal to $s_{\sigma(k+1)}$), but we do know that it is either 0 or 1. So this term is at least $\min\{|\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq 0\}|, |\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq 1\}|\} = \min\{|\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq s_{\sigma(k+1)}\}|, |\{y \in \bar{S}_k : y_{\sigma(k+1)} = s_{\sigma(k+1)}\}|\} = \min\{|S_{k+1}|, |\bar{S}_{k+1}|\} = |S_{k+1}|$, where the last equality uses $|S_k| \leq |\bar{S}_k|$ for all k. Finally, combining $|S_1| + \ldots + |S_p| \leq |S|$ with $|S_1| \geq |S_2| \geq \ldots \geq |S_p|$ gives $|S_p| \leq |S|/p$. Combining this with $|S_1| \leq |S|/2$, which follows from the definition of S_1, yields the result. ◀

We can now state our final oracle identification algorithm.

Algorithm 3 Final algorithm

1: $S \leftarrow \mathcal{C}$
2: **repeat**
3: Let σ and s be as in Lemma 3. Search for the first (according to σ) disagreement between x and s. If we find a disagreement, delete all inconsistent strings from S. If not, let $S \leftarrow \{s\}$.
4: **until** $|S| = 1$

As before, it is clear that this algorithm solves the problem. Let us analyze the query complexity. To compute the query complexity, let r be the number of times the loop repeats. Let $p_1, p_2, \ldots, p_r$ be the positions of disagreement. We have $\sum_{i=1}^{r} p_i \leq N$, as in Algorithm 2.

Unlike the previous analysis, the bound $r \leq \log M$ can be loose, since the size of S may reduce by a larger factor due to Lemma 3. Instead, we know that each iteration reduces the set S by a factor of $\max\{2, p_i\}$, which gives us $\prod_{i=1}^{r} \max\{2, p_i\} \leq M$. As before, we will assume the search subroutine is exact, which gives us a query upper bound of $O(\sum_{i=1}^{r} \sqrt{p_i})$, subject to the constraints $\sum_{i=1}^{r} p_i \leq N$ and $\prod_{i=1}^{r} \max\{2, p_i\} \leq M$. We solve this optimization problem in the full version [15] to obtain the following lemma.

▶ **Lemma 4.** *Let $C(M, N)$ be the maximum value attained by $\sum_{i=1}^{r} \sqrt{p_i}$, subject to the constraints $\sum_{i=1}^{r} p_i \leq N$, $\prod_{i=1}^{r} \max\{2, p_i\} \leq M$, $r \in [N]$ and $p_i \in [N]$ for all $i \in [r]$. Then $C(M, N) = O\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$ and $C(M, N) = O(\sqrt{M})$.*

Thus Algorithm 3 achieves the upper bound claimed in Theorem 2, under our assumption that the search subroutine is exact. Since it is not exact, we could reduce the error with logarithmic overhead, but it is usually unnecessary to incur this loss in quantum query algorithms. In the next section we prove this and establish the complexity of Algorithm 3.

3 Composition theorem for input-dependent query complexity

The primary aim of this section is to rigorously establish the query complexity of Algorithm 3. Along the way, we will develop techniques that can be used more generally. Let us begin by describing what we would like to prove. Algorithm 3 essentially consists of a loop repeated $r(x)$ times. We write $r(x)$ to make explicit its dependence on the input x. The loop itself consists of running a variant of Grover's algorithm on x, based on information we have collected thus far about x. Call these algorithms $A_1, A_2, \ldots, A_{r(x)}$. To be clear, A_1 is the algorithm that is run the first time the loop is executed, i.e., it looks for a disagreement under the assumption that $S = \mathcal{C}$. It produces an output $p_1(x)$, which is then used by A_2. A_2 looks for a disagreement assuming a modified set S, which is smaller than $\mathcal{C}$. Let us say that in addition to $p_2(x)$, A_2 also outputs $p_1(x)$. This ensures that the output of A_i completely describes all the information we have collected about x. Thus algorithm A_{i+1} now only needs the output of A_i to work correctly.

We can now view Algorithm 3 as a composition of $r(x)$ algorithms, $A_1, A_2, \ldots, A_{r(x)}$. It is a composition in the sense that the output of one is required as the input of the next algorithm. We know that the expected query complexity of A_i is $O(\sqrt{p_i(x)})$. If these algorithms were exact, then running them one after the other would yield an algorithm with expected query complexity $O(\sum_i \sqrt{p_i(x)})$. But since they are bounded error, this does not work. However, if we consider their worst-case complexities, we can achieve this complexity. If we have r algorithms $A_1, A_2, \ldots, A_r$ with worst-case query complexities Q_i, then there is a quantum algorithm that solves the composed problem with $O(\sum_i Q_i)$ queries. This is a remarkable property of quantum algorithms, which follows from the work of Lee et al. [16]. We first discuss this simpler result before moving on to input-dependent complexities.

3.1 Composition theorem for worst-case query complexity

We now show a composition theorem for solutions of the filtered γ_2-norm SDP, which implies a similar result for worst-case quantum query complexity. This follows from the work of Lee et al. [16], which we generalize in the next section. As discussed in the introduction, let $D \subseteq \{0,1\}^N$, and consider functions that map D to E. For any matrix A indexed by D, we define a quantity $\gamma(A)$. (To readers familiar with the notation of [16], this is their $\gamma_2(A|\Delta)$.)

▶ **Definition 5.** Let A be a square matrix indexed by D. We define $\gamma(A)$ as the following:

$$\gamma(A) := \min_{\{|u_{xj}\rangle, |v_{yj}\rangle\}} \max_{x \in D} \quad c(x) \tag{1}$$

$$\text{subject to:} \qquad \forall x \in D, \quad c(x) = \max\left\{ \sum_j \||u_{xj}\rangle\|^2, \sum_j \||v_{xj}\rangle\|^2 \right\} \tag{2}$$

$$\forall x, y \in D, \quad \sum_{j : x_j \neq y_j} \langle u_{xj} | v_{yj} \rangle = A_{xy} \tag{3}$$

We use $\gamma(A)$ to refer to both the SDP above and its optimum value. For a function $f : D \to E$, let F be its Gram matrix, defined as $F_{xy} = 1$ if $f(x) \neq f(y)$ and $F_{xy} = 0$ otherwise. Lee et al. showed that $Q(f) = \Theta(\gamma(J - F))$, where J is the all-ones matrix.

More generally, they showed that this SDP also upper bounds the quantum query complexity of state conversion. In the state conversion problem, we have to convert a given state $|s_x\rangle$ to $|t_x\rangle$. An explicit description of the states $|s_x\rangle$ and $|t_x\rangle$ is known for all $x \in D$, but we do not know the value of x. Since the query complexity of this task depends only on the Gram matrices of the starting and target states, define S and T by $S_{xy} = \langle s_x | s_y \rangle$ and $T_{xy} = \langle t_x | t_y \rangle$ for all $x, y \in D$. Let $S \mapsto T$ denote the problem of converting states with

Gram matrix S to those with Gram matrix T. If F is the Gram matrix of a function f, then $J \mapsto F$ is the function evaluation problem. Lee et al. showed that $Q(S \mapsto T) = O(\gamma(S - T))$, which generalizes $Q(f) = O(\gamma(J - F))$.

We now have the tools to prove the composition theorem for the filtered γ_2-norm SDP.

▶ **Theorem 6** ([16]). *Let $f_0, f_1, \ldots, f_k$ be functions with Gram matrices $F_0, F_1, \ldots, F_k$. Let $C_1, C_2, \ldots, C_k$ be the optimum value of the SDPs for the state conversion problems $F_0 \mapsto F_1, \ldots, F_{k-1} \mapsto F_k$, i.e., for $i \in [k]$, $C_i = \gamma(F_{i-1} - F_i)$. Then, $\gamma(F_0 - F_k) \leq \sum_{i=1}^{k} C_i$.*

This does not appear explicitly in [16], but simply follows from the triangle inequality $\gamma(A+B) \leq \gamma(A)+\gamma(B)$ [16, Lemma A.2]. From this we can also show an analogous theorem for quantum query complexity, which states $Q(F_0 \mapsto F_k) = O(\sum_{i=1}^{k} Q(F_{i-1} \mapsto F_i))$. We do not prove this claim as we do not need it in this paper.

For our application, we require a composition theorem similar to Theorem 6, but for input-dependent query complexity. However, it is not even clear what this means a priori, since the value $\gamma(J - F)$ does not contain information about input-dependent complexities. Indeed, the value is a single number and cannot contain such information. However, the SDP does contain this information and we modify this framework to be able to access this.

For example, let f be the find-first-one function, which outputs the smallest i such that $x_i = 1$ and outputs $N + 1$ if $x = 0^N$. There is a quantum algorithm that solves this with $O(\sqrt{f(x)})$ queries in expectation. Furthermore, there is a feasible solution for the $\gamma(J - F)$ SDP with $c(x) = O(\sqrt{f(x)})$, where $c(x)$ is the function that appears in (2). This suggests that $c(x)$ gives us information about the x-dependent query complexity. The same situation occurs when we consider the search problem with multiple marked items. There is a feasible solution with $c(x) = O(\sqrt{N/K})$ for inputs with K ones. This function $c(x)$ will serve as our input-dependent cost measure.

3.2 Cost functions

▶ **Definition 7** (Cost function). Let A be a square matrix indexed by D. We say $c : D \to \mathbb{R}$ is a feasible cost function for $\gamma(A)$ if there is a feasible solution of $\gamma(A)$ with values $c(x)$ in eq. (2). Let the set of all feasible cost functions for $\gamma(A)$ be denoted $\Gamma(A)$.

Note that if c is a feasible cost function for $\gamma(J - F)$, then $\max_x c(x)$ is an upper bound on the worst-case cost, $\gamma(J - F)$, which is exactly what we expect from an input-dependent cost. We can now prove an input-dependent analogue of Theorem 6 with $c(x)$ playing the role of $\gamma(J - F)$.

▶ **Theorem 8.** *Let $f_0, f_1, \ldots, f_k$ be functions with Gram matrices $F_0, F_1, \ldots, F_k$. Let $c_1, \ldots, c_k$ be feasible cost functions for $\gamma(F_0 - F_1), \ldots, \gamma(F_{k-1} - F_k)$, i.e., for $i \in [k]$, $c_i \in \Gamma(F_{i-1} - F_i)$. Then there is a $c \in \Gamma(F_0 - F_k)$ satisfying $c(x) \leq \sum_i c_i(x)$ for all $x \in D$.*

As in the case of Theorem 6, this follows from an analogous triangle inequality.

▶ **Lemma 9.** *Let A and B be square matrices indexed by D. If $c_A \in \Gamma(A)$ and $c_B \in \Gamma(B)$, there exists a $c \in \Gamma(A + B)$ satisfying $c(x) \leq c_A(x) + c_B(x)$ for all $x \in D$.*

This is shown by constructing a feasible solution for $\gamma(A + B)$ by taking the direct sum of vectors in a solution of $\gamma(A)$ and $\gamma(B)$. A proof appears in the full version [15].

In our applications, we will encounter algorithms that also output their input, i.e., accept as input $f(x)$ and output $(f(x), g(x))$. Note that the Gram matrix of the function $h(x) = (f(x), g(x))$ is merely $H = F \circ G$, defined as $H_{xy} = F_{xy}G_{xy}$.

Such an algorithm can either be thought of as a single quantum algorithm that accepts $f(x) \in E$ as input and outputs $(f(x), g(x))$ or as a collection of algorithms A_e for each $e \in E$, such that algorithm $A_{f(x)}$ requires no input and outputs $(f(x), g(x))$ on oracle input x. These are equivalent viewpoints, since in one direction you can construct the algorithms A_e from A by hardcoding the value of e and in the other direction, we can read the input e and call the appropriate A_e as a subroutine and output $(e, A_e(x))$. Additionally, if the algorithm $A_{f(x)}$ makes $q(x)$ queries on oracle input x, the algorithm A we constructed accepts $f(x)$ as input, outputs $(f(x), g(x))$, and makes $q(x)$ queries on oracle input x. While intuitive for quantum algorithms, we establish this rigorously for cost functions in the full version [15]:

▶ **Theorem 10.** *Let $f, g : D \to E$ be functions with Gram matrices F and G. For any $e \in E$, let $f^{-1}(e) = \{x : f(x) = e\}$. For every $e \in E$, let $c_e : f^{-1}(e) \to \mathbb{R}$ be a feasible cost function for $\gamma(J - G_e)$, where G_e denotes the matrix G restricted to those x that satisfy $f(x) = e$. Then there exists a $c \in \Gamma(F - F \circ G)$, such that $c(x) = c_{f(x)}(x)$.*

3.3 Algorithm analysis

We can now return to computing the query complexity of Algorithm 3. Using the same notation as in the beginning of this section, for any $x \in \mathcal{C}$, we define $r(x)$ to be the number of times the repeat loop is run in Algorithm 3 for oracle input x assuming all subroutines have no error. Similarly, let $p_1(x), p_2(x), \ldots p_{r(x)}(x)$ be the first positions of disagreement found in each run of the loop. Note that $p_1(x), p_2(x), \ldots p_{r(x)}(x)$ together uniquely specify x. Let $r = \max_x r(x)$.

We now define r functions $f_1, \ldots, f_r$ as $f_1(x) = p_1(x), f_2(x) = (p_1(x), p_2(x)), \ldots, f_r(x) = (p_1(x), \ldots, p_r(x))$, where $p_k(x) = 0$ if $k > r(x)$. Thus if P_i are the Gram matrices of the functions p_i, then $F_1 = P_1, F_2 = P_1 \circ P_2, \ldots, F_r = P_1 \circ P_2 \circ \cdots \circ P_r$.

We will now construct a solution for $\gamma(J - F_r)$, using solutions for the intermediate functions f_i. From Theorem 8 we know that we only need to construct solutions for $\gamma(J - F_1), \gamma(F_1 - F_2), \ldots, \gamma(F_{r-1} - F_r)$. From Theorem 10 we know that instead of constructing a solution for $\gamma(F_k - F_{k+1})$, which is $\gamma(F_k - F_k \circ P_{k+1})$, we can construct several solutions, one for each value of $f_k(x)$. More precisely, let $f_k : D \to E_k$; then we can construct solutions for $\gamma(J - P^e_{k+1})$ for all $e \in E_k$, where P^e_{k+1} is the matrix P_{k+1} restricted to x that satisfy $f_k(x) = e$.

For any k, the problem corresponding to $\gamma(J - P^e_{k+1})$ is just the problem of finding the first disagreement between x and a known string, which is the essentially the find-first-one function. This has a solution with cost function $O(\sqrt{f(x)})$, which in this case is $O(\sqrt{p_{k+1}(x)})$.

▶ **Theorem 11.** *Let f be the function that outputs the smallest i such that $x_i = 1$ and outputs $N + 1$ if $x = 0^N$ and let F be its Gram matrix. Then there is a $c \in \Gamma(J - F)$ such that $c(x) = O(\sqrt{f(x)})$.*

Proof. Let $a_k = k^{-1/4}$ and $b_k = 1/a_k = k^{1/4}$. Define $|u_{xj}\rangle = |v_{xj}\rangle$ as the following.

$$|u_{xj}\rangle = |v_{xj}\rangle = \begin{cases} a_j, & \text{if } j < f(x) \\ b_{f(x)}, & \text{if } j = f(x) \\ 0, & \text{if } j > f(x). \end{cases}$$

This is a feasible solution for $\gamma(J - F)$. Since the constraints are symmetric in x and y, there are two cases: either $f(x) < f(y)$ or $f(x) = f(y)$. In the first case, $\sum_{j:x_j \neq y_j} \langle u_{xj} | v_{yj} \rangle = \sum_{j=f(x)} \langle u_{xj} | v_{yj} \rangle = a_{f(x)} b_{f(x)} = 1$, since x and y agree on all positions before $f(x)$. In the

second case, $\sum_{j:x_j \neq y_j} \langle u_{xj}|v_{yj}\rangle = 0$, since x and y only disagree after position $f(x) = f(y)$. To compute the cost function, note that $c(0^N) = \sum_{k=1}^{N} a_k^2 = O(\sqrt{N}) = O(\sqrt{f(0^N)})$. For $x \neq 0^N$, $c(x) = \sum_{k=1}^{f(x)-1} a_k^2 + b_{f(x)}^2 = \sum_{k=1}^{f(x)-1} k^{-1/2} + \sqrt{f(x)} = O(\sqrt{f(x)})$. ◀

Our function is different from this one in two ways. First, we wish to find the first disagreement with a fixed string s instead of the first 1. This change does not affect the Gram matrix or the SDP. Second, we are looking for a disagreement according to an order σ, not from left to right. This is easy to fix, since we can replace j with $\sigma(j)$ in the definition of the vectors in the proof above.

This shows that for any k, there is a feasible cost function for $\gamma(J - P_{k+1}^e)$ with cost $c(x) = O(\sqrt{p_{k+1}(x)})$ for any x that satisfies $f_k(x) = e$. Using Theorem 10, we get that for any k there is a $c_k \in \Gamma(F_k - F_k \circ P_{k+1})$ with $c_k(x) = O(\sqrt{p_{k+1}(x)})$ for all $x \in D$. Finally, using Theorem 8, we have a $c \in \Gamma(J - F_r)$ with cost $c(x) = O(\sum_{i=1}^{r} \sqrt{p_i(x)}) = O(\sum_{i=1}^{r(x)} \sqrt{p_i(x)})$.

Since the function $f_r(x)$ uniquely determines x, we have a feasible cost function for oracle identification with cost $O(\sum_{i=1}^{r(x)} \sqrt{p_i(x)})$, subject to the constraints of Lemma 4, which we have already solved. Along with the lower bound, this yields the main result.

▶ **Theorem 2.** *For $N < M \leq 2^N$, $Q(\text{OIP}(M, N)) = \Theta\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$.*

4 Other applications

4.1 Quantum learning theory

The oracle identification problem has also been studied in quantum learning theory with the aim of characterizing $Q(\text{OIP}(\mathcal{C}))$. The algorithms and lower bounds studied apply to arbitrary sets $\mathcal{C}$, not just to the class of sets of a certain size, as in the rest of the paper. We show that Algorithm 3 also performs well for any set $\mathcal{C}$, outperforming the best known algorithm. The known upper and lower bounds for this problem are in terms of a combinatorial parameter $\hat{\gamma}^{\mathcal{C}}$, defined by Servedio and Gortler. They showed that for any $\mathcal{C}$, $Q(\text{OIP}(\mathcal{C})) = \Omega(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} + \frac{\log M}{\log N})$ [18]. Later, Atıcı and Servedio showed that $Q(\text{OIP}(\mathcal{C})) = O(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} \log M \log\log M)$ [5].

While we do not define $\hat{\gamma}^{\mathcal{C}}$, we can informally describe it as follows: $\hat{\gamma}^{\mathcal{C}}$ is the largest $\alpha < 1$, such that for any set $S \subseteq \mathcal{C}$, if we know that x belongs to S, there is a bit of x that can be queried such that size of the set of strings consistent with the answer to this query is at most $(1 - \alpha)|S|$, no matter what the oracle responds. This ensures that if we query the oracle with the permutation of Lemma 3, which was chosen to maximize the number of strings eliminated with a query, each query reduces the size of S by a factor of $(1 - \hat{\gamma}^{\mathcal{C}})$.

This adds an extra constraint to Lemma 4 of the form $M \prod_i^r (1 - \hat{\gamma}^{\mathcal{C}})^{p_i} \geq 1$, since learning p_i bits will reduce the size of the remaining set by a factor of $(1 - \hat{\gamma}^{\mathcal{C}})^{p_i}$. From this constraint we get $(\sum_i p_i) \log(1 - \hat{\gamma}^{\mathcal{C}}) \geq -\log M$. Using $\log(1 - \hat{\gamma}^{\mathcal{C}}) \leq -\hat{\gamma}^{\mathcal{C}}$ gives $\sum_i p_i \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$.

We may now replace the constraint $\sum_i p_i \leq N$ with $\sum_i p_i \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$ in the optimization problem of Lemma 4. This inequality also implies $p_i \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$ and $r \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$. Thus we may simply replace all occurrences of N by $\frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$ in Lemma 4. This yields the following theorem, which resolves a conjecture of Hunziker et al. [13, Conjecture 2].

▶ **Theorem 12.** *Algorithm 3 solves $\text{OIP}(\mathcal{C})$ with $O\left(\sqrt{\frac{1/\hat{\gamma}^{\mathcal{C}}}{\log 1/\hat{\gamma}^{\mathcal{C}}}} \log M\right)$ queries.*

Since $Q(\text{OIP}(\mathcal{C})) = \Omega(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} + \frac{\log M}{\log N})$, we see that Algorithm 3 makes $O(\frac{Q(\text{OIP}(\mathcal{C}))^2}{\sqrt{\log Q(\text{OIP}(\mathcal{C}))}} \log N)$ queries, which means it can be at most about quadratically worse than the optimal algorithm for $\text{OIP}(\mathcal{C})$.

4.2 Boolean matrix multiplication

In this section we show how to improve the upper bound on Boolean matrix multiplication (BMM) from $O(n\sqrt{l}\,\mathrm{poly}(\log n))$ [14] to $O(n\sqrt{l})$, where n is the size of the matrices and l is the output sparsity. Like in the analysis in Section 3, we will break up the BMM algorithm of [14] into a sequence of algorithms A_i such that the output of A_i is the input of A_{i+1}, and convert each algorithm into a feasible solution for the corresponding SDP.

The BMM algorithm is almost of this form: It uses two subroutines for graph collision, one for the decision problem and another to find all collisions. The first subroutine solves the problem on a bipartite graph with $2n$ vertices and m nonedges in $O(\sqrt{n}+\sqrt{m})$ queries. Since this query complexity is not input dependent, there is a feasible SDP solution for this problem with $c(x) = O(\sqrt{n}+\sqrt{m})$ using the known characterization of Lee et al. [16].

The second subroutine finds all graph collisions in an instance with λ collisions using $O(\sqrt{n\lambda}+\sqrt{m})$ queries. This upper bound is input dependent, since λ is a function of the input. In this subroutine, the only input-dependent algorithm is the variant of Grover's algorithm that uses $O(\sqrt{nk})$ queries to find all k ones in an n-bit string with k ones. It is easy to show that there is a feasible cost function for this with $c(x) = O(\sqrt{nk})$. For example, we may compose the SDP solution for the find-first-one function (Theorem 11) with itself repeatedly to find all ones. The cost function of the resultant SDP will satisfy $c(x) = O(\sum_i \sqrt{p_i})$, where p_is are the locations of the ones. By the Cauchy-Schwarz inequality this is $O(\sqrt{nk})$. Thus the second subroutine has a feasible cost function $c(x) = O(\sqrt{n\lambda}+\sqrt{m})$.

The BMM algorithm breaks up the problem into n instances of graph collision. The algorithm repeatedly searches for indices i such that the ith graph collision instance has a collision. Then it finds all graph collisions of this instance and repeats. Instead of searching for any i, we can search for the first i. The problem of searching for the first i that has a graph collision is the composition of the find-first-one function (Theorem 11) and the graph collision function. It is a composition in the sense that each input bit of the first problem is the output bit of another problem. It is known that the optimal value of the γ SDP for $f \circ g^n$ is at most $\gamma(J-F)\gamma(J-G)$. Similarly, it can be shown that there is a feasible cost function for $f \circ g$ that is at most the product of the cost functions. This is similar to [16, Lemma 5.1] or Lemma 9, but we take the tensor product instead of taking the direct sum.

Finally, let $p_1, \ldots, p_t$ be the positions of indices found in the algorithm. The search problem requires $O(\sqrt{p_i}(\sqrt{n}+\sqrt{m}))$ queries for each i, since it is the composition of the two above-mentioned algorithms. The algorithm that finds all graph collisions has a feasible cost function $O(\sqrt{n\lambda_i}+\sqrt{m})$, where λ_i is the number of graph collisions in the ith graph collision instance. This gives a feasible cost function for BMM with cost $O(\sum_i(\sqrt{p_i}(\sqrt{n}+\sqrt{m})+\sqrt{n\lambda_i}+\sqrt{m}))$, which is the same optimization problem solved in [14], without log factors. This is $O(n\sqrt{l})$.

5 Open questions

Our composition theorem only works for solutions of the filtered γ_2-norm SDP, not for quantum query complexity itself. While this is sufficient for our application, it would be interesting to know if bounded-error quantum algorithms with input-dependent query complexities can be composed in general without incurring log factors.

While the query complexity of oracle identification in terms of M and N has been fully characterized, finding an optimal quantum algorithm for $\mathrm{OIP}(\mathcal{C})$ remains open, even classically. It would also be interesting to study time-efficient oracle identification algorithms for specific sets $\mathcal{C}$, since none of the known algorithms is known to be time efficient.

Acknowledgments. I thank Andrew Childs and Ben Reichardt for helpful discussions, Seiichiro Tani for pointing me to Ref. [3], and Andrew Childs and Ansis Rosmanis for comments on a preliminary draft. This work was supported in part by NSERC, the Ontario Ministry of Research and Innovation, and the US ARO.

References

1 Andris Ambainis, Kazuo Iwama, Akinori Kawachi, Hiroyuki Masuda, Raymond H. Putra, and Shigeru Yamashita. Quantum Identification of Boolean Oracles. In *STACS 2004*, volume 2996 of *LNCS*, pages 105–116. Springer, 2004.

2 Andris Ambainis, Kazuo Iwama, Akinori Kawachi, Rudy Raymond, and Shigeru Yamashita. Improved algorithms for quantum identification of Boolean oracles. *Theor. Comput. Sci.*, 378(1):41 – 53, 2007.

3 Andris Ambainis, Kazuo Iwama, Masaki Nakanishi, Harumichi Nishimura, Rudy Raymond, Seiichiro Tani, and Shigeru Yamashita. Average/worst-case gap of quantum query complexities by on-set size. *arXiv preprint arXiv:0908.2468*, 2009.

4 Dana Angluin. Queries and Concept Learning. *Machine Learning*, 2:319–342, 1988.

5 Alp Atıcı and Rocco Servedio. Improved Bounds on Quantum Learning Algorithms. *Quantum Information Processing*, 4:355–386, 2005.

6 Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and Weaknesses of Quantum Computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997.

7 Ethan Bernstein and Umesh Vazirani. Quantum Complexity Theory. *SIAM J. Comput.*, 26(5):1411–1473, 1997.

8 Dan Boneh and Mark Zhandry. Quantum-Secure Message Authentication Codes. In *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 592–608. Springer, 2013.

9 Andrew M. Childs, Robin Kothari, Maris Ozols, and Martin Roetteler. Easy and Hard Functions for the Boolean Hidden Shift Problem. In *TQC 2013*, volume 22 of *LIPIcs*, pages 50–79, 2013.

10 Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM J. Comput.*, 35(6):1310–1328, 2006.

11 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC 1996*, pages 212–219, 1996.

12 Tibor Hegedűs. Generalized teaching dimensions and the query complexity of learning. In *COLT 1995*, pages 108–117, 1995.

13 Markus Hunziker, David A. Meyer, Jihun Park, James Pommersheim, and Mitch Rothstein. The geometry of quantum learning. *Quantum Information Processing*, 9(3):321–341, 2010.

14 Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Improving Quantum Query Complexity of Boolean Matrix Multiplication Using Graph Collision. In *ICALP 2012*, volume 7391 of *LNCS*, pages 522–532. Springer, 2012.

15 Robin Kothari. An optimal quantum algorithm for the oracle identification problem. *arXiv preprint arXiv:1311.7685*, 2013.

16 Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Špalek, and Mario Szegedy. Quantum Query Complexity of State Conversion. In *FOCS 2011*, pages 344–353, 2011.

17 Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

18 Rocco A. Servedio and Steven J. Gortler. Equivalences and Separations Between Quantum and Classical Learnability. *SIAM J. Comput.*, 33(5):1067–1092, 2004.

19 Wim van Dam. Quantum Oracle Interrogation: Getting All Information for Almost Half the Price. In *FOCS 1998*, page 362, 1998.

20 Wim van Dam, Sean Hallgren, and Lawrence Ip. Quantum Algorithms for Some Hidden Shift Problems. *SIAM J. Comput.*, 36(3):763–778, 2006.

A Solution to Wiehagen's Thesis*

Timo Kötzing

Friedrich-Schiller-Universität Jena, Jena, Germany
timo.koetzing@uni-jena.de

Abstract

Wiehagen's *Thesis in Inductive Inference* (1991) essentially states that, for each learning criterion, learning can be done in a normalized, enumerative way. The thesis was not a formal statement and thus did not allow for a formal proof, but support was given by examples of a number of different learning criteria that can be learned enumeratively.

Building on recent formalizations of learning criteria, we are now able to formalize Wiehagen's Thesis. We prove the thesis for a wide range of learning criteria, including many popular criteria from the literature. We also show the limitations of the thesis by giving four learning criteria for which the thesis does not hold (and, in two cases, was probably not meant to hold). Beyond the original formulation of the thesis, we also prove stronger versions which allow for many corollaries relating to *strongly decisive* and *conservative* learning.

1998 ACM Subject Classification I.2.6 Learning

Keywords and phrases Algorithmic Learning Theory, Wiehagen's Thesis, Enumeration Learning

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.494

1 Introduction

In Gold-style learning [10] (also known as *inductive inference*) a *learner* tries to learn an infinite sequence, given more and more finite information about this sequence. For example, a learner h might be presented longer and longer initial segments of the sequence $g = 1, 4, 9, 16, \ldots$. After each new datum of g, h may output a description of a function (for example a Turing machine program computing that function) as its conjecture. h might output a program for the constantly-1 function after seeing the first element of this sequence g, and then, as soon as more data is available, a program for the squaring function. Many criteria for saying whether h is successful on g have been proposed in the literature. Gold, in his seminal paper [10], gave a first, simple learning criterion, later called *Ex-learning*[1], where a learner is successful iff it eventually stops changing its conjectures, and its final conjecture is a correct program (computing the input sequence).

Trivially, each single, describable sequence g has a suitable constant function as an Ex-learner (this learner constantly outputs a description for g). Thus, we are interested in sets of total computable functions $\mathcal{S}$ for which there is a single learner h learning each member of $\mathcal{S}$ (those sets $\mathcal{S}$ are then called *Ex-learnable*).

Gold [10] showed an important class of sets of functions to be Ex-learnable:[2] each

* We would like to thank Sandra Zilles for bringing Wiehagen's Thesis in connection with the approach of abstractly defining learning criteria, as well as the anonymous reviewers for their friendly and helpful suggestions.

[1] "Ex" stands for *explanatory*.

[2] We let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of natural numbers and we fix a coding for programs based on Turing machines letting, for any program (code) $p \in \mathbb{N}$, φ_p be the function computed by the Turing machine coded to p.



© Timo Kötzing;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 494–505



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



uniformly computable set of total functions is Ex-learnable; a set of functions $\mathcal{S}$ is uniformly computable iff there is a computable function e such that $\mathcal{S} = \{\varphi_{e(n)} \mid n \in \mathbb{N}\}$. The corresponding learner learns by enumeration: in every iteration, it finds the first index n such that $\varphi_{e(n)}$ is consistent with all known data, and outputs $e(n)$ as the conjecture.

However, it is well-known that there are sets which are not uniformly computable, yet Ex-learnable. Blum and Blum [6] gave the following example. Let e be a total computable listing of programs such that the predicate $\varphi_{e(n)}(x) = y$ is decidable in n, x and y. Crucially, some of the $\varphi_{e(n)}$ may be undefined on some arguments; these functions are not required to be learned, but the set of all the *total* functions enumerated is Ex-learnable. This uses the same strategy as for uniformly computable sets of functions, but this learning already goes *beyond enumeration* of all and only the learned functions, as there are sets which are so learnable, but not uniformly computable. The price is that the learner may give intermediate conjectures $e(n)$ which are programs for partial functions; this is necessarily so, as noted in [9].

As already shown by Wiehagen [16], there are Ex-learnable sets of functions that cannot be learned while always having a hypothesis that is consistent with the known data. Thus, the above strategy for learning employed by Blum and Blum [6] is not applicable for all learning tasks. In [17, 18] Wiehagen was looking for whether there is a more general strategy which also enumerates a list of candidate conjectures and is applicable to *all* Ex-learnable sets. He showed that this is indeed possible, giving an insightful characterization of Ex-learning.

A main focus of the research in inductive inference defines learning criteria that are different from (but usually similar in flavor to) Ex-learning. For example, *consistent* learning requires that each conjecture is consistent with the known data; *monotone* learning requires the sequence of conjectures to be monotone with respect to inclusion of the graphs of the computed functions. Wiehagen also gives characterizations for these learning criteria and more. Other researchers give similar characterizations; recent work in this area includes, for example, [1]. For any learning criterion I we are again interested in sets of total computable functions $\mathcal{S}$ for which there is a single learner h which learns every function in $\mathcal{S}$ in the sense specified by I; we call such $\mathcal{S}$ *I-learnable*.

Wiehagen was inspired by his work to conjecture a general structure of learning, as stated in his *Thesis in Inductive Inference* [18], which we rephrase in the language of this paper:

> Let $\mathcal{I}$ be any learning criterion. Then for any $\mathcal{I}$-learnable class $\mathcal{S}$, an enumeration of programs e can be constructed such that $\mathcal{S}$ is $\mathcal{I}$-learnable with respect to e by an enumerative learner.

Note that [18] called a learning criterion an "inference type" and a learner an "inference strategy". About his thesis, Wiehagen [18] wrote that "We do not exclude that one nice day a formal proof of this thesis will be presented. This would require 'only' to formalize the notions of 'inference type' and 'enumerative inference strategy' which does not seem to be hopeless. But up to this moment we prefer 'verifying' our thesis analogously as it has been done with 'verifying' Church's thesis, namely by formally proving it for 'real', reasonable, distinct inference types."

Recently, the notion of a learning criterion was formalized in [13] (see Section 2.1 for the formal notions relevant to this paper). Our first contribution in this paper is a formalization of "enumeration learner" in Definition 2. It is in the nature of the very general thesis that any formalization may be too broad in some respects and too narrow in other. For example, our formalizations exclude some learning criteria, such as finite learning, learning by non-total

learners, and criteria featuring global restrictions on the learner. However, for the scope of our definitions, we already get very strong and insightful results in this paper.

In Theorem 3 we discuss four different learning criteria in which the thesis does *not* hold. The first one is *prediction*, which attaches a totally different meaning to the "conjectures" than Ex-learning (the thesis was probably never meant to hold for such learning criteria). The second criterion involves mandatory oscillation between (correct) conjectures, which is in immediate contradiction to enumerative learning. The third learning criterion is *transductive learning*, where the learner has very little information in each iteration. The fourth is learning in a non-standard hypothesis space. The last two learning criteria do not contradict enumerative learning directly, but still demand too much for learning by enumeration.

In Section 4 we show that there is a broad core of learning criteria for which Wiehagen's Thesis holds. For this we introduce the notion of a *pseudo-semantic restriction*, where only the semantics of conjectures and possibly the occurrence of mind changes matter, but not other parts of their syntax. Theorem 10 shows that Wiehagen's Thesis holds in the case of *full information learning* (like in Ex-learning given above, where the learner only gets more information in each iteration) when all restrictions are pseudo-semantic, and in Theorem 16 we see that the same holds in the case of *iterative learning* (a learning model in which a learner has a restricted memory). Note that these two theorems already cover a very wide range of learning criteria from the literature, including all given by Wiehagen [18].

Finally, going beyond the scope of Wiehagen's Thesis, we show that we can assume the enumeration e of programs to be *semantically 1-1* (each $e(n)$ codes for a different function) if we assume a little bit more about the learning criteria, namely that their restrictions allow for *patching* and *erasing* (see Definition 11). This is formally shown in Theorem 13 (for the case of full information learning) and in Theorem 17 (for the case of iterative learning). Example criteria to which these theorems apply include Ex-learning, as well as consistent and monotone learning. Wiehagen [18] already pointed out in special cases that one can get such semantically 1-1 enumerations. From these results on learning with a semantically 1-1 enumeration we can derive corollaries to conclude that the learning criteria, to which the theorems apply, allow for *strongly decisive* and *conservative* learning (see Definition 1); for example, for plain Ex-learning, this proves (a stronger version of) a result from [15] (which showed that Ex-learning can be done decisively). Note that all positive results are sufficient conditions for enumerative learnability; except for the (weak) condition given in Remark 9, we could not find interesting *necessary* conditions.

The benefits of this work are threefold. First, we address a long-open problem in its essential parts. Second, we derive results about (strongly) decisive and conservative learning in many different settings. Finally, we further develop general techniques to derive powerful theorems applicable to many different learning criteria, thanks to general notions such as "pseudo-semantic restriction".

Note that we omit a number of nontrivial proofs due to space constraints.

2 Mathematical Preliminaries

We fix any computable 1-1 and onto pairing function $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$; Whenever we consider tuples of natural numbers as input to a function, it is understood that the general coding function $\langle \cdot, \cdot \rangle$ is used to code the tuples into a single natural number. We similarly fix a coding for finite sets and sequences, so that we can use those as input as well. We use $\emptyset$ to denote the empty sequence; for every non-empty sequence σ we let σ^- denote the sequence derived from σ by dropping the last listed element.

If a function f is not defined for some argument x, then we denote this fact by $f(x)\uparrow$, and we say that f on x *diverges*; the opposite is denoted by $f(x)\downarrow$, and we say that f on x *converges*. If f on x converges to p, then we denote this fact by $f(x)\downarrow = p$. For any total computable predicate P, we use $\mu x.P(x)$ to denote the minimal x such that $P(x)$ (undefined, if no such x exists). The special symbol ? is used as a possible hypothesis (meaning "no change of hypothesis").

Unintroduced notation for computability theory follows [14]. $\mathcal{P}$ and $\mathcal{R}$ denote, respectively, the set of all partial computable and the set of all computable functions (mapping $\mathbb{N} \to \mathbb{N}$). For any function $f : \mathbb{N} \to \mathbb{N}$ and all i, we use $f[i]$ to denote the sequence $f(0), \ldots, f(i-1)$ (undefined, if any one of these values is undefined).

We will use a number of basic computability-theoretic results in this paper. First, we fix a *padding function*, a 1-1 function $\text{pad} \in \mathcal{R}$ such that $\forall p, n, x : \varphi_{\text{pad}(p,n)}(x) = \varphi_p(x)$. Intuitively, pad generates infinitely many syntactically different copies of the semantically same program. We require that pad is monotone increasing in both arguments. The *S-m-n Theorem* states that there is a 1-1 function $s \in \mathcal{R}$ such that $\forall p, n, x : \varphi_{s(p,n)}(x) = \varphi_p(n, x)$. Intuitively, s-m-n allows for "hard-coding" arguments to a program.

2.1 Learning Criteria

In this section we formally introduce our setting of learning in the limit and associated learning criteria. We follow [13] in its "building-blocks" approach for defining learning criteria. A *learner* is a partial computable function from $\mathbb{N}$ to $\mathbb{N} \cup \{?\}$. A *sequence generating operator* is a function β taking as arguments a function h (the learner) and a function g (the learnee) and that outputs a function p. We call p the *conjecture sequence* of h given g. Intuitively, β defines how a learner can interact with a given learnee to produce a sequence of conjectures.

The most important sequence generating operator is **G** (which stands for "Gold", who first studied it [10]), which gives the learner full information about the learning process so far; this corresponds to the examples of learning criteria given in the introduction. Formally, **G** is defined such that

$$\forall h, g, i : \mathbf{G}(h, g)(i) = h(g[i]).$$

We define two additional sequence generating operators **It** (*iterative learning*, [16]) and **Td** (*transductive learning*, [8]) as follows. For all learners h, learnees g and all i,

$$\mathbf{It}(h,g)(i) = \begin{cases} h(\emptyset), & \text{if } i = 0;\ ^{3} \\ h(\mathbf{It}(h,g)(i-1), i-1, g(i-1)), & \text{otherwise;} \end{cases}$$

$$\mathbf{Td}(h,g)(i) = \begin{cases} h(\emptyset), & \text{if } i = 0; \\ \mathbf{Td}(h,g)(i-1), & \text{else, if } h(i-1, g(i-1)) = ?; \\ h(i-1, g(i-1)), & \text{otherwise.} \end{cases}$$

For both of iterative and transductive learning, the learner is presented with a new datum each turn (argument/value pair from the learnee in complete and argument-increasing order). Furthermore, in iterative learning, the learner has access to the previous conjecture, but not so in transductive learning; however, in transductive learning, the learner can *implicitly* take over the previous conjecture by outputting "?".

Successful learning requires the learner to observe certain restrictions, for example convergence to a correct index. These restrictions are formalized in our next definition. A

[3] $h(\emptyset)$ denotes the *initial conjecture* (based on no data) made by h.

sequence acceptance criterion is a predicate δ on a learning sequence and a learnee. The most important sequence acceptance criterion is denoted **Ex** (which stands for "Explanatory"), already studied by Gold [10]. The requirement is that the conjecture sequence converges (in the limit) to a correct hypothesis for the learnee (we met this requirement already in the introduction). Formally, for any programming system[4] ψ, we define $\mathbf{Ex}_\psi$ as a predicate such that

$$\mathbf{Ex}_\psi = \{(p,g) \in \mathcal{R}^2 \mid \exists n_0, q : \forall n \geq n_0 : p(n) = q \wedge \psi_q = g\}.$$

Standardly we use $\mathbf{Ex} = \mathbf{Ex}_\varphi$. We will meet many more sequence acceptance criteria below. We combine any two sequence acceptance criteria δ and δ' by intersecting them; we denote this by juxtaposition (for example, the sequence acceptance criteria given below are meant to be always used together with **Ex**).

For any set $\mathcal{C} \subseteq \mathcal{P}$ of possible learners, any sequence generating operator β and any sequence acceptance criterion δ, $(\mathcal{C}, \beta, \delta)$ (or, for short, $\mathcal{C}\beta\delta$) is a *learning criterion.* A learner $h \in \mathcal{C}$ *$\mathcal{C}\beta\delta$-learns* the set $\mathcal{C}\beta\delta(h) = \{g \in \mathcal{R} \mid \delta(\beta(h,g),g)\}$. A set $\mathcal{S} \subseteq \mathcal{R}$ of possible learnees is called *$\mathcal{C}\beta\delta$-learnable* iff there is a function $h \in \mathcal{C}$ which $\mathcal{C}\beta\delta$-learns all elements of $\mathcal{S}$ (possibly more). Abusing notation, we also use $\mathcal{C}\beta\delta$ to denote the set of all $\mathcal{C}\beta\delta$-learnable sets (learnable by some learner).

Next we define a number of further sequence acceptance criteria which are of interest for this paper.

▶ **Definition 1.** With **Cons** we denote the restriction of *consistent learning* [4, 6] (being correct on all known data); with **Conf** the restriction of *conformal learning* [17] (being correct or divergent on known data); with **Conv** we denote the restriction of *conservative learning* [2] (never abandoning a conjecture which is correct on all known data); with **Mon** we denote the restriction of *monotone learning* [12] (conjectures make all the outputs that previous conjectures made – monotonicity in the graphs); finally, with **PMon** we denote the restriction of *pseudo-monotone learning* [18] (conjectures make all the *correct* outputs that previous conjectures made). The following definitions formalize these restrictions.

$$\begin{aligned}
\mathbf{Conf} &= \{(p,g) \in \mathcal{R}^2 \mid \forall n \forall x < n : \varphi_{p(n)}(x)\!\downarrow \Rightarrow \varphi_{p(n)}(x) = g(x)\};\\
\mathbf{Cons} &= \{(p,g) \in \mathcal{R}^2 \mid \forall n \forall x < n : \varphi_{p(n)}(x) = g(x)\};\\
\mathbf{Conv} &= \{(p,g) \in \mathcal{R}^2 \mid \forall n : p(n) \neq p(n+1) \Rightarrow \exists x < n+1 : \varphi_{p(n)}(x) \neq g(x)\};\\
\mathbf{Mon} &= \{(p,g) \in \mathcal{R}^2 \mid \forall i \leq j\ \forall x : \varphi_{p(i)}(x)\!\downarrow \Rightarrow \varphi_{p(j)}(x)\!\downarrow = \varphi_{p(i)}(x)\};\\
\mathbf{PMon} &= \{(p,g) \in \mathcal{R}^2 \mid \forall i \leq j\ \forall x : \varphi_{p(i)}(x)\!\downarrow = g(x) \Rightarrow \varphi_{p(j)}(x)\!\downarrow = \varphi_{p(i)}(x)\}.
\end{aligned}$$

An example of a well-studied learning criterion is $\mathcal{R}$**GConsEx**, requiring convergence of the learner to a correct conjecture, as well as consistent conjectures along the way.

Furthermore, we are interested in a number of restrictions which disallow certain kinds of returning to abandoned conjectures. We say that a learner exhibits a *U-shape* when it first outputs a correct conjecture, abandons this, and then returns to a correct conjecture. We distinguish between *syntactic* U-shapes (returning to the syntactically same conjecture), *semantic* U-shapes (returning to the semantically same conjecture, after semantically abandoning it; note that we drop the qualifier "semantic" in this case) and *strong* U-shapes (outputting a semantically same conjecture after syntactically abandoning it; this is called strong, because it leads to the stronger restriction). Forbidding these kinds of U-shapes leads

[4] We call ψ a *programming system* iff, for all p, ψ_p is a computable function, and the function mapping any p and x to $\psi_p(x)$ is also (partial) computable.

to the respective non-U-shapedness restrictions **SynNU**, **NU** and **SNU**. If we consider forbidding returning to abandoned conjectures more generally, we get three corresponding restrictions of *decisiveness*. We give the formal definitions here.

$$\begin{aligned}
\mathbf{SynNU} &= \{(p,g) \in \mathcal{R}^2 \mid \forall i \leq j \leq k : (\varphi_{p(i)} = g \wedge p(i) = p(k)) \Rightarrow p(j) = p(i)\};\\
\mathbf{NU} &= \{(p,g) \in \mathcal{R}^2 \mid \forall i \leq j \leq k : \varphi_{p(i)} = g = \varphi_{p(k)} \Rightarrow \varphi_{p(j)} = \varphi_{p(i)}\};\\
\mathbf{SNU} &= \{(p,g) \in \mathcal{R}^2 \mid \forall i \leq j \leq k : \varphi_{p(i)} = g = \varphi_{p(k)} \Rightarrow p(j) = p(i)\};\\
\mathbf{SynDec} &= \{(p,g) \in \mathcal{R}^2 \mid \forall i \leq j \leq k : p(i) = p(k) \Rightarrow p(j) = p(i)\};\\
\mathbf{Dec} &= \{(p,g) \in \mathcal{R}^2 \mid \forall i \leq j \leq k : \varphi_{p(i)} = \varphi_{p(k)} \Rightarrow \varphi_{p(j)} = \varphi_{p(i)}\};\\
\mathbf{SDec} &= \{(p,g) \in \mathcal{R}^2 \mid \forall i \leq j \leq k : \varphi_{p(i)} = \varphi_{p(k)} \Rightarrow p(j) = p(i)\}.
\end{aligned}$$

Of these variations of disallowing returning to abandoned conjectures, mostly **NU** [3] and **Dec** [15] are well-studied, but also **SNU** [5, 18] drew some attention; however, almost all of this work was done for the case of learning of languages (with the exception of [15]).

Note that the literature knows many more learning criteria than those constructible from the parts given in this section (see the text book [11] or the survey [19] for an overview).

3 Learning by Enumeration

In this section we formally introduce our notions of *learning by enumeration* and derive some easy statements from these definitions. We start with the general definition of learning by enumeration.

▶ **Definition 2.** Let $\mathcal{I}$ be a learning criterion and let $\mathcal{S} \subseteq \mathcal{R}$ be $\mathcal{I}$-learnable by some learner $h \in \mathcal{R}$. We say that h *learns by enumeration* iff there is a 1-1 enumeration $e \in \mathcal{R}$ of possible conjectures such that, for each $g \in \mathcal{R}$, there is a monotonically non-decreasing function r such that $e \circ r$ is the conjecture sequence of h on g. We say that a learning criterion $\mathcal{I}$ *allows for learning by enumeration* iff each $\mathcal{I}$-learnable set is $\mathcal{I}$-learnable by a learner learning by enumeration. We call e the enumeration of conjectures.

Note that, since e is required to be 1-1 and r non-decreasing, in any learning sequence of an enumeration learner h, no once abandoned hypothesis will be returned to; such abandoned hypotheses we call *refuted*. This immediately gives the following remark.

▶ Remark. Let h learn by enumeration. Then h learns syntactically decisively. In particular, for any learning criterion $\mathcal{I}$ allowing for learning by enumeration, every $\mathcal{I}$-learnable set is $\mathcal{I}$-learnable by a syntactically decisive learner.

From the wealth of (theoretically possible) learning criteria we quickly see that there are learning criteria which do not allow for learning by enumeration. For example, the task of *prediction* is typically modeled by using the sequence acceptance criterion **M** (for *matching*) defined as $\{(p,g) \mid \exists n_0 \forall n \geq n_0 : p(n) = g(n)\}$; in this case, the output of the learner is interpreted as the prediction for the next element in the sequence, instead of as a program (we consider the learning criterion $\mathcal{R}$**GM**). A relaxation of the strict convergence required by **Ex** is given by $\mathbf{Fex}_{\leq k}$ ($k > 0$), where a learner may oscillate between at most k different (but correct!) hypotheses in the limit; as a somewhat unnatural variant, we let $\mathbf{Fex}_{=k}$ require oscillation between *exactly* k different (and correct) hypotheses. With these definitions, we get the follwing theorem.

▶ **Theorem 3.** *The following learning criteria do* not *allow for learning by enumeration.*
1. $\mathcal{R}\mathbf{GM}$.
2. $\mathcal{R}\mathbf{GFex}_{=2}$.
3. $\mathcal{R}\mathbf{TdEx}$.
4. *For some programming systems* ψ, $\mathcal{R}\mathbf{GEx}_{\psi}$.

In fact, all these learning criteria do not even allow for syntactically decisive learning; in the case of all items except (3) not even for syntactically non-U-shaped learning.

At the side we remark that Theorem 3, (3) cannot be strengthened in the same way as the other items: $\mathcal{R}\mathbf{TdEx}$-learning *does* allow for strongly non-U-shaped learning, as the next theorem shows.

▶ **Theorem 4.** *We have that every* $\mathcal{R}\mathbf{TdEx}$*-learnable set is so learnable by a strongly non-U-shaped learner, i.e.*

$$\mathcal{R}\mathbf{TdSNUEx} = \mathcal{R}\mathbf{TdEx}.$$

We will see in Theorem 10 that many learning criteria allow for learning by enumeration because of a simple padding trick, by semantically (but not syntactically) repeating any relevant conjecture infinitely in the enumeration (see below for details). In the following definition we strengthen the definition of learning by enumeration by requiring the enumeration of hypothesis to never *semantically* repeat a hypothesis.

▶ **Definition 5.** A function $e \in \mathcal{R}$ is called *semantically 1-1* iff, for all i, j, $\varphi_{e(i)} = \varphi_{e(j)}$ implies $i = j$. That is (by taking the contrapositive), different pre-images under e not only give different images, but even *semantically* different images.

A learner h which learns by enumeration using some $e \in \mathcal{R}$ as the enumeration of conjectures is said to *learn by semantically 1-1 enumeration* iff e is semantically 1-1. For $\mathcal{I}$ a learning criterion, a set $\mathcal{S} \subseteq \mathcal{R}$ is said to be $\mathcal{I}$*-learnable by semantically 1-1 enumeration* iff there is an $\mathcal{I}$-learner h for $\mathcal{S}$ learning by semantically 1-1 enumeration. We say that a learning criterion $\mathcal{I}$ *allows for learning by semantically 1-1 enumeration* iff each $\mathcal{I}$-learnable set $\mathcal{S}$ is $\mathcal{I}$-learnable by semantically 1-1 enumeration.

Some of the power of learning by semantically 1-1 enumeration is shown in the following remark, strengthening the conclusion of Remark 3.

▶ Remark. Let h learn by semantically 1-1 enumeration. Then h learns strongly decisively. In particular, for any learning criterion $\mathcal{I}$ allowing for learning by semantically 1-1 enumeration, every $\mathcal{I}$-learnable set is $\mathcal{I}$-learnable by a strongly decisive learner.

4 The Power of Enumeration Learning

In this section we give our theorems confirming Wiehagen's Thesis for a wide range of learning criteria. First we look at the very important family of learning criteria which use **G** as their sequence generating operator (full information learning). Note that all examples given in [18] were from this family (but did not require total learners).

We start by giving a definition for enumerative learning in the **G**-setting (Definition 6) and that of pseudo-semantic restrictions (Definition 8). After this we are ready for the first main theorem of the paper, Theorem 10, which shows that Wiehagen's Thesis holds for many learning criteria using **G** as their sequence generating operator. With Definition 11 we introduce *patching* and *erasing*, which will allow for us to give the second main theorem of the paper, Theorem 13, which shows that many learning criteria with **G** as their sequence generating operator even allow for semantically 1-1 enumeration.

At the end of this section, with Theorems 16 and 17 and Corollary 18 we show that all results carry over to **It** as sequence generating operator.

▶ **Definition 6.** A pair (R, e) where R is a total computable predicate over pairs of numbers and (finite) data sequences and $e \in \mathcal{R}$ is a 1-1 computable function, is called a **G**-*style enumeration by refutation pair* iff, for all i, σ, y, $R(i, \sigma)$ implies $R(i, \sigma y)$ (i.e., R is monotone in the second argument, any conjecture, once refuted, stays refuted) and, for all σ, there is i such that $R(i, \sigma)$. The associated *enumeration learner* $h_{(R,e)}$ is defined such that

$$\forall \sigma : h_{(R,e)}(\sigma) = e(\mu i . \neg R(i, \sigma)).$$

The following theorem is straightforward to verify.

▶ **Theorem 7.** *For every δ, if h is a learner $\mathcal{R}\mathbf{G}\delta$-learning by enumeration according to Definition 2 with some enumeration of hypotheses e, then there is some R such that $h = h_{(R,e)}$.*

In order to exclude the examples given in Theorem 3 we now make some definitions for learning criteria which allow for learning by enumeration. Intuitively, we focus our attention on learning criteria which consider all conjectures as φ-conjectures, and are only interested in syntactic properties as far as mind changes are concerned.

▶ **Definition 8.** For all $p \in \mathcal{R}$, we let

$$\begin{aligned} \mathrm{Sem}(p) &= \{p' \in \mathcal{R} \mid \forall i : \varphi_{p(i)} = \varphi_{p'(i)}\}; \\ \mathrm{Mc}(p) &= \{p' \in \mathcal{R} \mid \forall i : (p(i) = p(i+1) \Rightarrow p'(i) = p'(i+1))\}. \end{aligned}$$

A sequence acceptance criterion δ is said to be a *semantic restriction* iff, for all $(p, g) \in \delta$ and $p' \in \mathrm{Sem}(p)$, $(p', g) \in \delta$. A sequence acceptance criterion δ is said to be a *pseudo-semantic restriction* iff, for all $(p, g) \in \delta$ and $p' \in \mathrm{Sem}(p) \cap \mathrm{Mc}(p)$, $(p', g) \in \delta$.

Intuitively, semantic restrictions allow for arbitrarily changing the syntax of the conjectures, as long as the semantics stay the same. Pseudo-semantic restrictions further require that no additional mind changes are introduced this way.

▶ **Example 9.** Any intersection of two (pseudo-) semantic restrictions is a (pseudo-) semantic restriction. Example semantic restrictions include **Conf**, **Cons**, **Mon**, **PMon**, **NU**, **Dec**; pseudo-semantic restrictions include **Ex**, **Conv**, **SNU** and **SDec**. Many more learning criteria from the literature could be added to these lists.

Example sequence acceptance criteria which are *not* pseudo-semantic restrictions include **M** (prediction), **SynNU**, **SynDec** and several more from the literature.

With these definitions we can now formulate the first main theorem of the paper, confirming Wiehagen's Thesis for a large family of **G**-style learning criteria.

▶ **Theorem 10.** *Let δ be a pseudo-semantic restriction. Then $\mathcal{R}\mathbf{G}\delta$ allows for learning by enumeration.*

Proof. The proof is based on a "padding trick": we can safely refute any hypothesis as long as we make sure that a (syntactically different) copy of the refuted hypothesis is still available. Formally, let $\mathcal{S} \in \mathcal{R}\mathbf{G}\delta$, as witnessed by a learner $h \in \mathcal{R}$. We define a computable predicate R and $c, e \in \mathcal{R}$ such that

$$\begin{aligned} c(\emptyset) &= 0; \\ \forall \sigma \neq \emptyset : c(\sigma) &= \mu n . \langle h(\sigma^-), c(\sigma^-)\rangle \leq \langle h(\sigma), n\rangle; \\ \forall m, n : e(m, n) &= \mathrm{pad}(m, n); \text{[5]} \\ \forall i, \sigma : R(i, \sigma) &\Leftrightarrow \langle h(\sigma), c(\sigma)\rangle > i. \end{aligned}$$

Clearly, e is 1-1 and R is monotone in the second component (as c is monotone). Let $g \in \mathcal{S}$. Let $p = \mathbf{G}(h, g)$ be the conjecture sequence of h on g and $p' = \mathbf{G}(h_{(R,e)}, g)$ the conjecture sequence of $h_{(R,e)}$ on g. It remains to show that $p' \in \mathrm{Sem}(p) \cap \mathrm{Mc}(p)$. We start with $p' \in \mathrm{Sem}(p)$. For all j, x we have[6]

$$\begin{aligned}
\varphi(p'(j), x) &= \varphi(h_{(R,e)}(g[j]), x) \\
&= \varphi(e(\mu i \,.\, \neg R(i, g[j])), x) \\
&= \varphi(e(\mu i \,.\, \langle h(g[j]), c(g[j])\rangle \leq i), x) \\
&= \varphi(e(\langle h(g[j]), c(g[j])\rangle), x) \\
&= \varphi(\mathrm{pad}(h(g[j]), c(g[j])), x) \\
&= \varphi(h(g[j]), x) \\
&= \varphi(p(j), x).
\end{aligned}$$

Hence, $p' \in \mathrm{Sem}(p)$.

Suppose $j \in \mathbb{N}$ such that $h(g[j]) = h(g[j+1])$. Then, $c(g[j+1]) = c(g[j])$; hence, for all i, $R(i, g[j]) \Leftrightarrow R(i, g[j+1])$ (there are no new hypotheses rejected). Therefore, $h_{(R,e)}(g[j]) = h_{(R,e)}(g[j+1])$. This shows $p' \in \mathrm{Mc}(p)$. ◀

We are now interested in strengthening the conclusion of Theorem 10 by restricting the family of learning criteria under consideration. For this we introduce variations on the notion of a pseudo-semantic restriction.

▶ **Definition 11.** Let δ be a sequence acceptance criterion. We say that δ *allows for patching* iff, for all $(p, g) \in \delta$ and $p' \in \mathrm{Mc}(p)$ such that all conjectures of p' are just as the corresponding conjectures of p, only possibly corrected for some arguments (these corrections are called *patches*); we say δ *allows for monotone patching* if this holds for all $p' \in \mathrm{Mc}(p)$ which patch later conjectures in all the places that earlier conjectures are patched at. Formally, δ allows for monotone patching iff, for all $(p, g) \in \delta$ and all $p' \in \mathrm{Mc}(p)$ where there is $(A_n)_{n\in\mathbb{N}}$ such that

$$\begin{aligned}
&\forall n < m: && A_n \subseteq A_m; \\
&\forall n, x: && \varphi_{p'(n)}(x) = \begin{cases} \varphi_{p(n)}(x), & \text{if } x \notin A_n; \\ g(x), & \text{if } x \in A_n. \end{cases}
\end{aligned}$$

we have $(p', g) \in \delta$. If we drop the first requirement of monotonicity of $(A_n)_{n\in\mathbb{N}}$, we get the formal definition for δ allowing for patching.

We say that δ *allows for erasing* iff, for all $(p, g) \in \delta$ and $p' \in \mathrm{Mc}(p)$ such that all conjectures of p' are just as the corresponding conjectures of p, only possibly made divergent for some arguments for which no data is known (the arguments set to diverge are called *erased*); we say δ *allows for monotone erasing* if this holds for all $p' \in \mathrm{Mc}(p)$ which erase in later conjectures at at most the places that earlier conjectures were erased at (and only on unknown data). Finally, we say δ *allows for almost-monotone erasing* iff monotone erasing is violated only when the new conjecture corrects an earlier mistake; formally: δ allows for almost-monotone erasing iff for all $(p, g) \in \delta$ and $p' \in \mathrm{Mc}(p)$ we have $(p', g) \in \delta$ if there is a

[5] The function pad was defined in Section 2.
[6] For convenience we write, for all a, z, $\varphi(a, z)$ instead of $\varphi_a(z)$.

sequence $(A_n)_{n\in\mathbb{N}}$ of subsets of $\mathbb{N}$ such that the following hold.

$$\begin{aligned}
\forall n: &\quad A_n \cap \{0,\dots n-1\} = \emptyset;\\
\forall n,m: &\quad n \leq m \Rightarrow (A_m \subseteq A_n \vee \exists x : \varphi_{p(m)}(x) = g(x) \neq \varphi_{p(n)}(x)\downarrow);\\
\forall n,x: &\quad \varphi_{p'(n)}(x) = \begin{cases} \varphi_{p(n)}(x), & \text{if } x \notin A_n;\\ \uparrow, & \text{if } x \in A_n. \end{cases}
\end{aligned}$$

Intuitively, an almost-monotone erasing erases less and less except when the new conjecture corrects a *convergent* mistake, and only erases where no data is available.

For example, if some δ allows for (monotone) patching, then a $\mathcal{R}\mathbf{G}\delta$-learner can always patch all known data into the conjecture (up to the last mind change – otherwise $p' \in \mathrm{Mc}(p)$ will be violated). We use almost-monotone erasing in Theorem 13, where we need something more than just monotone erasing, and do not require full erasing power for the sake of generality. Note that any δ allowing for (monotone) patching or erasing is a pseudo-semantic restriction.

▶ **Example 12.** Any intersection of two sequence acceptance criteria allowing for (monotone) patching or erasing again allows for (monotone) patching or erasing, respectively; the same holds for almost-monotone erasing. Examples of sequence acceptance criteria allowing for patching and erasing are **Conf**, **Cons** and **Ex**; **Mon** and **PMon** allow for monotone patching and monotone erasing. **Mon**, but not **PMon**, allows for almost-monotone erasing.

With the definition of a patching and erasing we can now give another main theorem of the paper. The proof uses ideas from proofs in [18] (where, implicitly, special cases of this theorem have been proven), as well as from [7], which gives a general technique for avoiding U-shapes in language learning.

▶ **Theorem 13.** *Let δ allow for monotone patching and almost-monotone erasing. Then $\mathcal{R}\mathbf{G}\delta$ allows for learning by semantically 1-1 enumeration. Furthermore, there is an enumeration learner which learns conservatively.*

Proof. Let $\mathcal{S} \in \mathcal{R}\mathbf{G}\delta$ as witnessed by some learner $h \in \mathcal{R}$. As δ allows for monotone patching, we can assume, without loss of generality, that h patches each new conjecture with the known data at every mind change.

Let M be the set of all finite sequences σ such that either $\sigma = \emptyset$ or $h(\sigma^-) \neq h(\sigma)$. Note that M is a decidable set. We can assume, without loss of generality, that M is infinite; as otherwise we can introduce dummy members into M which will not invalidate the proof. Thus, there is a 1-1 total computable enumeration $(\tau_i)_{i\in\mathbb{N}}$ of all and only the elements in M respecting the order on finite sequences (i.e., for all i,j, if $\tau_i \subseteq \tau_j$, then $i \leq j$; in particular, $\tau_0 = \emptyset$). For all i, we let $z(i) = h(\tau_i)$ be the conjecture after the ith listed sequence and $n(i) = \mathrm{len}(\tau_i)$ the length of the ith sequence. We define e with s-m-n such that, for all i and x,

$$\varphi_{e(i)}(x) = \begin{cases} \varphi_{z(i)}(x), & \text{if } x < n(i) \text{ or, for all } y \text{ with } n(i) < y < x :\\ & \quad \varphi_{z(i)}(y)\downarrow \text{ and } h(\varphi_{z(i)}[y+1]) = z(i);\\ \uparrow, & \text{otherwise.} \end{cases}$$

Note that, for all i, $\varphi_{z(i)}[n(i)]\downarrow = \tau_i$, as we assumed that h patches all known data into the new conjecture at each mind change. In particular, this shows that e is semantically 1-1. We define R as follows.

$$R(i,\sigma) \Leftrightarrow \tau_i \text{ and } \sigma \text{ are } \subseteq\text{-incomparable or } \tau_i \subset \sigma \text{ and } \exists \sigma' : \tau_i \subseteq \sigma' \subseteq \sigma \wedge h(\sigma') \neq h(\tau_i).$$

Note that R is total computable and monotone in its second argument. Intuitively, we always use the conjecture that h would have used, modified appropriately to ensure that the enumeration is semantically 1-1. Clearly, (R, e) is an **G**-style enumeration by refutation pair. We show that $h_{(R,e)}$ $\mathcal{R}\mathbf{G}\delta$-learns $\mathcal{S}$.

Let $g \in \mathcal{S}$. Let $p = \mathbf{G}(h, g)$ be the conjecture sequence of h on g and $p' = \mathbf{G}(h_{(R,e)}, g)$ the conjecture sequence of $h_{(R,e)}$ on g. From the order of listing of the τ_i and the definition of R we get that, for all n, $p'(n) = e(i)$ for i such that τ_i is the $\subseteq$-maximal element of M with $\tau_i \subseteq g[n]$; this also gives that h made no mind change between τ_i and $g[n]$. Thus, we get $p' \in \mathrm{Mc}(p)$ and, for all n, $p(n)$ and $p'(n)$ are semantically equivalent apart from possibly erased arguments; let $(A_n)_{n\in\mathbb{N}}$ be the corresponding sequence of erased sets of arguments (which are thus exactly the arguments on which the corresponding conjecture $p'(n)$ is undefined). Let now $n < m$ be such that $A_m \not\subseteq A_n$. Without loss of generality, $n = 0$ or $p(n) \neq p(n-1)$ and $p(m) \neq p(m-1)$. Thus, $p'(n) = e(i)$ with $\tau_i = g[n]$ and $p'(m) = e(j)$ with $\tau_j = g[m]$. Then we get from patching that $\varphi_{p(m)}(m-1) = g(m-1)$. Furthermore, A_m contains only numbers $\geq m$, and since A_n is closed upwards and $A_m \not\subseteq A_n$, we get $m - 1 \notin A_n$. This shows that $\varphi_{p'(n)}(m-1) = \varphi_{e(i)}(m-1)$ converges; thus, it converges to the same value as $\varphi_{p(n)}$ on $m-1$. However, this value cannot equal $g(n)$, as this value leads to a mind change (this we get from $\tau_j \in M$), and any value leading to a mind change would be erased by the definition of e. ◀

We can see the deep power and versatility of Theorem 13 in connection with Remark 3 and the various examples of sequence acceptance criteria fulfilling the prerequisites of Theorem 13, which leads, for example, to the following corollary.

▶ **Corollary 14.** *The following learning criteria allow for learning strongly decisively and conservatively.* $\mathcal{R}$**GEx**; $\mathcal{R}$**GConfEx**; $\mathcal{R}$**GConsEx**; $\mathcal{R}$**GMonEx**; $\mathcal{R}$**GConsMonEx**.

At the side we remark that Theorem 13 cannot be improved to apply also to pseudo-monotone learning, as the following Theorem shows.

▶ **Theorem 15.** *There is a* $\mathcal{R}$**GPMonEx**-*learnable set of functions which cannot be so learned strongly non-U-shapedly.*

Finally, we show that analogous theorems can also be derived for iterative learning. Theorem 16 is analogous to Theorem 10, and Theorem 17 is analogous to Theorem 13; both proofs are also analogous, but different in some details.

▶ **Theorem 16.** *Let δ be a pseudo-semantic restriction. Then $\mathcal{R}\mathbf{It}\delta$ allows for learning by enumeration.*

▶ **Theorem 17.** *Let δ allow for monotone patching and almost-monotone erasing. Then $\mathcal{R}\mathbf{It}\delta$ allows for learning by semantically 1-1 enumeration. Furthermore, there is an enumeration learner which learns conservatively.*

Just as in the case of **G**-style learning, were we got a powerful corollary (Corollary 14), we get the analogous corollary also for **It**-style learning.

▶ **Corollary 18.** *The following learning criteria allow for learning strongly decisively and conservatively.* $\mathcal{R}$**ItEx**; $\mathcal{R}$**ItConfEx**; $\mathcal{R}$**ItConsEx**; $\mathcal{R}$**ItMonEx**; $\mathcal{R}$**ItConsMonEx**.

References

1. Y. Akama and T. Zeugmann. Consistent and coherent learning with δ-delay. *Information and Computation*, 206:1362–1374, 2008.
2. D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.
3. G. Baliga, J. Case, W. Merkle, F. Stephan, and W. Wiehagen. When unlearning helps. *Information and Computation*, 206:694–709, 2008.
4. J. Bārzdiņš. Inductive inference of automata, functions and programs. In *Proc. of the International Congress of Mathematicians*, pages 455–560, 1974. English translation in, *American Mathematical Society Translations*: Series 2 109 (1977), pp. 107-112.
5. H.R. Beick. *Induktive Inferenz mit Höchster Inferenzgeschwindigkeit*. PhD thesis, Humboldt University of Berlin, 1984.
6. L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.
7. J. Case and T. Kötzing. Strongly non-U-shaped learning results by general techniques. In *Proc. of COLT (Conference on Learning Theory)*, pages 181–193, 2010.
8. J. Case and T. Kötzing. Learning secrets interactively. Dynamic modeling in inductive inference. *Information and Computation*, 220:60–73, 2012.
9. J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.
10. E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
11. S. Jain, D. Osherson, J. Royer, and A. Sharma. *Systems that Learn: An Introduction to Learning Theory*. MIT Press, Cambridge, Massachusetts, second edition, 1999.
12. K. Jantke. Monotonic and non-monotonic inductive inference of functions and patterns. In J. Dix, K. Jantke, and P. Schmitt, editors, *Nonmonotonic and Inductive Logic*, volume 543 of *Lecture Notes in Computer Science*, pages 161–177. 1991.
13. T. Kötzing. *Abstraction and Complexity in Computational Learning in the Limit*. PhD thesis, University of Delaware, 2009. Available online at http://pqdtopen.proquest.com/#viewpdf?dispub=3373055.
14. H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw Hill, New York, 1967. Reprinted by MIT Press, Cambridge, Massachusetts, 1987.
15. G. Schäfer-Richter. *Über Eingabeabhängigkeit und Komplexität von Inferenzstrategien*. PhD thesis, RWTH Aachen, 1984.
16. R. Wiehagen. Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. *Elektronische Informationverarbeitung und Kybernetik*, 12:93–99, 1976.
17. R. Wiehagen. *Zur Theorie der Algorithmischen Erkennung*, 1978. Dissertation B, Humboldt University of Berlin.
18. R. Wiehagen. A thesis in inductive inference. In *Proc. of the Workshop on Nonmonotonic and Inductive Logic*, pages 184–207, 1991.
19. Thomas Zeugmann and Sandra Zilles. Learning recursive functions: A survey. *Theoretical Computer Science*, 397:4–56, 2008.

Space-Efficient String Indexing for Wildcard Pattern Matching

Moshe Lewenstein[1], Yakov Nekrich[*2], and Jeffrey Scott Vitter[2]

1 **Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel**
2 **Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence, US**

Abstract

In this paper we describe compressed indexes that support pattern matching queries for strings with wildcards. For a constant size alphabet our data structure uses $O(n \log^{\varepsilon} n)$ bits for any $\varepsilon > 0$ and reports all occ occurrences of a wildcard string in $O(m + \sigma^g \cdot \mu(n) + \text{occ})$ time, where $\mu(n) = o(\log \log \log n)$, σ is the alphabet size, m is the number of alphabet symbols and g is the number of wildcard symbols in the query string. We also present an $O(n)$-bit index with $O((m + \sigma^g + \text{occ}) \log^{\varepsilon} n)$ query time and an $O(n(\log \log n)^2)$-bit index with $O((m + \sigma^g + \text{occ}) \log \log n)$ query time. These are the first non-trivial data structures for this problem that need $o(n \log n)$ bits of space.

1998 ACM Subject Classification F2.2.Nonnumerical Algorithms and Problems

Keywords and phrases compressed data structures, compressed indexes, pattern matching

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.506

1 Introduction

In the string indexing problem, we pre-process a source string T, so that all occurrences of a query string P in T can be reported. This is one of the most fundamental data structure problems. While handbook data structures, suffix arrays and suffix trees, can answer string matching queries efficiently, they store the source string T in $\Theta(\log n)$ bits of space per symbol. In situations when massive amounts of data must be indexed, the space usage can become an issue. Compressed indexes that use $o(\log n)$ or even H_0 bits per symbol, where H_0 denotes the zero-order entropy, were studied extensively. We refer the reader to [12] for a survey of results on compressed indexing.

In many scenarios we are interested in reporting all occurrences of strings that resemble the query string $\tilde{P}$ but do not have to be identical to $\tilde{P}$. The problem of approximate pattern matching is important for biological applications and information retrieval and has received considerable attention [3, 9, 14, 19, 1, 2]. In this paper we consider a variant of the approximate pattern matching when the query string $\tilde{P}$ may contain wildcards (don't care symbols), and the wildcard symbol matches any alphabet symbol.

The standard indexing data structures can be used to answer wildcard pattern matching queries. A pattern $\tilde{P}$ with g wildcard symbols matches σ^g different patterns, where σ denotes the size of the alphabet. We can generate all patterns that match $\tilde{P}$ and report all occ occurrences of these patterns (and hence all occurrence of $\tilde{P}$) in $O(m \cdot \sigma^g + \text{occ})$ time, where m is the number of alphabet symbols. If the maximal number of wildcards in a query is bounded

* The work of this author is partially supported by NSERC of Canada.



© Moshe Lewenstein, Yakov Nekrich, and Jeffrey Scott Vitter;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 506–517



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



Table 1 Previous and new results on unbounded wildcard indexing; m and g denote the number of alphabet symbols and wildcards in the query pattern.

Ref.	Space Usage	Query Time
[3]	$O(n \log n)$ words	$O(m + \sigma^g \log\log n + \mathrm{occ})$
[1]	$O(n)$ words	$O(m + \sigma^g \log\log n + \mathrm{occ})$
New	$O(n \log^{\varepsilon} n)$ bits	$O(m + \sigma^g \sqrt{\log^{(3)} n} + \mathrm{occ})$
New	$O(n(\log\log n)^2)$ bits	$O((m + \sigma^g + \mathrm{occ}) \log\log n)$
New	$O(n)$ bits	$O((m + \sigma^g + \mathrm{occ}) \log^{\varepsilon} n)$

by k (k-bounded indexing), we can store a compressed trie with all possible combinations of k wildcard symbols for every suffix. Then a query can be answered in $O(|\tilde{P}| + \mathrm{occ})$ time, but the total space usage is $O(n^{k+1})$ words of $\Theta(\log n)$ bits.

Cole *et al.* [3] presented an elegant data structure for k-bounded indexing. Their solution needs $O(n \log^k n)$ words of space and answers wildcard queries in $O(m + 2^g \log\log n + \mathrm{occ})$ time. Very recently this has been improved in [10] to $O(n \log^{k+\varepsilon} n)$ bits of space with the same query time as Cole *et al.* [3]. Bille *et al.* [1] obtained another trade-off: for any predefined k and β, their k-bounded index uses $O(n \log n \log_{\beta}^{k-1} n)$ words and answers queries in $O(m + \beta^g \log\log n + \mathrm{occ})$ time. These indexes can provide fast answers to wildcard queries when the number of wildcards is small. However the space usage of the above data structures is high even when k is a constant. For super-constant values of k (for instance, when the maximal number of wildcards is bounded by $\log\log n$) the cost of storing the data structure may become prohibitive.

Another line of research is the design of data structures that use linear or almost-linear space and support queries with an arbitrarily large number of wildcards. Cole *et al.* [3] describe a data structure that uses $O(n \log n)$ words and answers queries in $O(m + \sigma^g \log\log n + \mathrm{occ})$ time. Iliopoulos and Rahman [14] and Lam *et al.* [9] describe linear-space indexes; however, their data structures need $\Theta(n)$ worst-case time to answer a query. Recently, Bille *et al.* [1] described an $O(n)$-words data structure that answers queries in $O(m + \sigma^g \log\log n + \mathrm{occ})$ time.

When the amount of stored data is very large, even linear space usage can be undesirable. While numerous compressed indexes for exact pattern matching are known, there are no previously described data structures for wildcard indexing that use $o(n \log n)$ bits. In this paper we present sublinear space indexes for wildcard pattern matching. Our results are especially conspicuous when the alphabet size is constant. Our first data structure uses $O(n \log^{\varepsilon} n)$ bits and reports occurrences of a wildcard pattern in $O(m + \sigma^g \sqrt{\log^{(3)} n} + \mathrm{occ})$ time[1]; henceforth ε denotes an arbitrarily small positive constant. Thus we improve both the space usage and the query time of the previous best data structure [1]. The space usage can be further decreased at cost of slightly increasing the query time. We describe two indexes that use $O(n)$ and $O(n(\log\log n)^2)$ bits of space; queries are supported in $O((m + \sigma^g + \mathrm{occ}) \log^{\varepsilon} n)$ and $O((m + \sigma^g + \mathrm{occ}) \log\log n)$ time respectively. Previous and new results with worst-case efficient query times are listed in Table 1.

In this paper we assume, unless specified otherwise, that the alphabet size is a constant. But our techniques are also relevant for the case when the alphabet size is arbitrarily large. We can obtain an $O(n \log \sigma)$-bit data structure that answers queries in $O((m + \sigma^g + \mathrm{occ}) \log_{\sigma}^{\varepsilon} n)$

[1] $\log^{(3)} n = \log\log\log n$.

time. We can also obtain an $O(n \log n)$-bit data structure that supports queries in $O(m + \sigma^g + \mathrm{occ})$ time if $\sigma \geq \log \log n$. Other interesting trade-offs are possible and will be described in the full version of this paper.

In Section 2, we recall some results related to compressed suffix trees and suffix arrays and compressed data structures for a set of integers. We also define the unrooted LCP queries, introduced in Cole *et al.* [3], that are the main tool in all currently known efficient structures for wildcard indexing. In Section 3 we describe data structures that answer unrooted LCP queries on a small subtree of the suffix tree. Our data structures need only a small number of additional bits if the (compressed) suffix tree and suffix array of the source text are available. In Section 4, we describe compact data structures that answer LCP queries and wildcard pattern matching queries on an arbitrarily large suffix tree. These data structures are based on a subdivision of suffix tree nodes into small subtrees. In Sections 5, 9, and 7 we show how we can speed-up the data structures from [3], [1] and retain $o(n \log n)$ space usage. The main component of our improvement is a method for processing batches of unrooted LCP queries. In previous works [3, 1] LCP queries were answered one-by-one.

2 Preliminaries

Unrooted LCP Queries. In this paper $s_1 \circ s_2$ denotes the concatenation of strings s_1 and s_2 and $\mathcal{T}$ denotes the suffix tree of the source text. A string $str(v, u)$ is obtained by concatenating labels of all edges on the path from v to u and $str(u) = str(v_r, u)$ for the root node v_r of $\mathcal{T}$. A *location* on a suffix tree $\mathcal{T}$ is an arbitrary position on an edge of $\mathcal{T}$; a location on an edge (v, u) can be uniquely identified by specifying the edge (u, v) and the offset from the upper node of (u, v). We can straightforwardly extend the definitions of $str(\tilde{v}, \tilde{u})$ and $str(\tilde{u})$ to arbitrary locations $\tilde{u}$ and $\tilde{v}$. The unrooted LCP query (v, P), defined in [3], asks for the lowest descendant location $\tilde{u}$ of a node v, such that $str(v, \tilde{u})$ is a prefix of a string P. Thus an unrooted LCP query provides the answer to the following question: if we were to search for a pattern P in a subtree with root v, where would the search end? While we can obviously answer this question in $O(|P|)$ time by traversing the trie starting at v, faster solutions are also possible.

As in the previous works [3, 1], we consider the following two-stage scenario for answering queries: during the first stage an arbitrary string P is pre-processed in $O(|P|)$ time; during the second stage, we answer queries (u, P_j) for any suffix P_j of P and any $u \in \mathcal{T}$. Cole *et al.* [3] described an $O(n \log^2 n)$-bit data structure that answers unrooted LCP queries in $O(\log \log n)$ time. Bille *et al.* [1] improved the space usage to linear ($O(n \log n)$ bits).

Compressed Suffix Arrays and Suffix Trees. The suffix array SA for a text T contains starting positions of T's suffixes sorted in lexicographic order: $SA[i] = k$ if the suffix $T[k..n]$ is the k-th smallest suffix of the text T. We will say that i is the rank of the suffix $T[k..n]$. An inverse suffix array stores information about lexicographic order of suffixes: $SA^{-1}[k] = i$ iff $SA[i] = k$. We will say that a data structure provides a suffix array functionality in time t_{SA} if it enables us to compute $SA[i]$ and $SA^{-1}[k]$ for any $1 \leq i, k \leq n$ in $O(t_{SA})$ time. A number of compressed data structures provide suffix array functionality in little time.

▶ **Lemma 1.** *If the alphabet size $\sigma = O(1)$, the following trade-offs for space usage $s(n)$ and t_{SA} are possible: (a) $s(n) = O((1/\varepsilon)n)$ and $t_{SA}(n) = O(\log^{\varepsilon} n)$, or (b) $s(n) = O(n \log \log n)$ and $t_{SA}(n) = O(\log \log n)$, or (c) $s(n) = O(n \log^{\varepsilon} n)$ and $t_{SA}(n) = O(1)$ for any constant $\varepsilon > 0$*

Proof. Result (a) is shown in [17] and results (b), (c) are from [15] ◀

If $SA[t] = f$ the function $\Psi^i(t)$ computes the position of the suffix $T[f + i..n]$ in the suffix array. This function can be computed in $O(t_{SA})$ time as $SA^{-1}[SA[t] + i]$. Let the string depth of a node $v \in \mathcal{T}$ be the length $str(v)$. If the suffix array functionality is available, we can store the suffix tree in $O(n)$ additional bits, so that the string depth of any node v can be computed in $O(t_{SA})$ time [18, 4, 16].

Using $O(n)$ additional bits, we can process a string P in $O(|P|t_{SA})$ time and find for any suffix $P^j = P[j..|P|]$ of P: (i) the rank r_j of P^j in T and (ii) the longest common prefix (LCP) of P^j and the suffixes $SA[r_j]$, $SA[r_j + 1]$ of T. We can obtain this information in $O(|P|t_{SA})$ time by following the suffix links in a compressed suffix tree. For completeness, we provide a description of this procedure in [11].

Heavy Path Decomposition. Let $\mathcal{T}$ be an arbitrary tree. We can decompose $\mathcal{T}$ into disjoint root-to-leaf paths, called *heavy paths*. If an internal node $u \in \mathcal{T}$ is on a heavy path p, then its heaviest child u_i (that is, the child with the greatest number of leaf descendants) is also on p. If the child u_j of u is not on p, then u has at least twice as many leaf descendants as u. Therefore the heavy-path decomposition of $\mathcal{T}$ guarantees that *any* root-to-leaf path in $\mathcal{T}$ intersects with at most $\log n$ heavy paths; we refer to [8] for details.

Searching in a Small Set. We can search in a set with a poly-logarithmic number of elements using the data structure called an atomic heap [5]. An atomic heap on a set of integers S, $|S| = \log^{O(1)} n$, uses linear space and enables us to find for any integer q the largest $e \in S$ such that $e \leq q$ (respectively, the smallest $e \in S$ such that $e \geq q$) in $O(1)$ time. Using the result of Grossi *et al.* [6], we can search in a small set using small additional space and only one access to elements of S.

▶ **Lemma 2** ([6], Lemma 3.3). *Suppose that $|S| = \log^{O(1)} n$ and $e \leq n$ for any $e \in S$. There exists a data structure D that uses $O(|S| \log\log n)$ additional bits and answers predecessor and successor queries on S in $O(1)$ time. When a query is answered, only one element $e' \in S$ needs to be accessed.*

3 Unrooted LCP Queries on Small Sets

In this section we describe compact data structures that answer LCP queries on a small set of suffixes. We consider a set S that contains a poly-logarithmic number of consecutive suffixes from the suffix array of S. Our data structure supports queries of the form (u_0, P) where $u_0 \in \mathcal{T}_0$ and $\mathcal{T}_0$ is a subtree of the suffix tree $\mathcal{T}$ induced by suffixes from S; the query answer is the lowest location $\tilde{v} \in \mathcal{T}_0$ below $\tilde{u}$, such that $str(u_0, \tilde{v}_0)$ is a prefix of P. These data structures are an important building block of data structures that will be constructed in the following sections and a key to space-saving solution: we will show in section 4 how a suffix tree can be divided into small subtrees. In this section we show how unrooted LCP queries can be supported on such small subtrees. The main idea is to keep the (ranks of) suffixes in succinct predecessor data structures that need $O(\log\log n)$ additional bits per element; we do not have to store the ranks in these data structures because they can be retrieved in $O(t_{SA})$ time using the (compressed) suffix tree and the (compressed) suffix array. Thus we can answer unrooted LCP queries on $\mathcal{T}_0$ using $O((\log\log n)^2)$ bits per suffix. We assume in the rest of this section that S contains $f = O(\log^3 n)$ consecutive suffixes and $\mathcal{T}_0$ is a subtree of the suffix tree induced by suffixes from S.

▶ **Lemma 3.** *There exists a data structure that uses $O(f(\log\log n)^2)$ additional bits of space and answers unrooted LCP queries on $\mathcal{T}_0$ in $O(1)$ time. We assume that our data structure*

can access the suffix tree of T, the suffix array of T, the inverse suffix array of T, and a universal look-up table of size $O(n^g)$ for an arbitrarily small positive constant g.

Proof. Let $\mathcal{T}_0$ denote the part of the suffix tree induced by suffixes in S. We apply the heavy path decomposition to nodes of $\mathcal{T}_0$. Let $S(u)$ denote the set that contains all strings $str(w, v_l)$ for the parent w of u and all leaf descendants v_l of u. We remark that all elements of $S(u)$ are suffixes of T. The global rank of a suffix Suf is its position in the suffix array of T. Let $R(u)$ denote the set of global ranks of all suffixes in $S(u)$. For every node $u \in \mathcal{T}_0$ and each of its children u_i that are not on the same heavy path as u, we store a data structure $D(u_i)$. $D(u_i)$ answers predecessor queries on $R(u_i)$. It is not necessary to store the set $R(u)$ itself: an arbitrary element of $R(u)$ can be accessed using the functionality provided by the suffix array. Suppose that the global rank of the suffix corresponding to $str(w, v_p)$, where v_p is the p-th leaf descendant of $S(u)$, should be computed. Since we can access the suffix tree, we can find the rank r_1 of the suffix that ends in the leaf v_p. Then the suffix corresponding to $str(w, v_p)$ has rank $SA[SA^{-1}[r_1] + depth(w)]$ where $depth(w)$ is the string depth of the node w in the global suffix tree. By Lemma 2, $D(u_i)$ can be stored in $O(|S(u_i)| \log\log n)$ bits and answer predecessor queries in $O(1)$ time. The total number of elements in all $D(u)$ is $O(f \log f) = O(f \log\log n)$. Thus all $D(u)$ need $O(f(\log\log n)^2)$ bits or $o(f)$ words of $\log n$ bits. For every heavy path h_j on $\mathcal{T}_0$ we keep a data structure H_j that contains the depths of all nodes. H_j is also implemented as described in Lemma 2 and uses $O(\log\log n)$ bits per node.

The search for an LCP in $\mathcal{T}_0$ is organized in the same way as in [3]. To answer a query (u, P_j), $u \in \mathcal{T}_0$, we start by finding $l_0 = lcp(P_j, SA[r])$, where r is the rank of the suffix that starts at u and ends in the leaf v_h, such that u and v_h are on the same heavy path. Let u' denote the lowest node of depth $d_1 \leq depth(u) + l_0$ that is on the same heavy path h_0 in $\mathcal{T}_0$ as u. If $d_1 \neq depth(u) + l_0$, then u' is the answer to our query. If $d_1 = depth(u) + l_0$ and u' is a leaf, then again u' is the answer to our query. If $d_1 = depth(u) + l_0$ and u' is not a leaf, we identify the child u_j of u' that is labelled with $P_j[d_1 + 1]$. If such a child does not exist, then again u' is the answer. Otherwise, we find the rank r' of $P'_j = P_j[d_1 + 1..|P_j|]$. Using $D(u_j)$, we find the predecessor and the successor of r' in $S(u_j)$.

Let S_l and S_r denote the corresponding suffixes of $D(u_j)$. We can compute $l_l = lcp(P'_j, S_l)$ and $l_r = lcp(P'_j, S_r)$. Suppose that $l_l \geq l_r$. Let u_l be the node of depth at most $depth(u_j) + l_j$ on the path from u_j to the leaf l_l containing S_l. The node u_l, that can be found by answering an appropriate level ancestor query for l_l, is the answer to the original LCP query. The case when $l_r > l_l$ is handled in the same way. ◀

In the following two Lemmas we extend the result of Lemma 3 to the situation when the data structure is stored in compressed form. We assume that we can compute $SA[i]$, $SA^{-1}[i]$ for any i, $1 \leq i \leq n$, in $O(t_{SA})$ time; we also assume that compressed suffix tree with functionality described in Section 2 is available. Only additional bits necessary to support queries on $\mathcal{T}_0$ are counted.

▶ **Lemma 4.** *There exists a data structure that uses $O(f(\log\log n)^3)$ additional bits of space and answers unrooted LCP queries on $\mathcal{T}_0$ in $O(t_{SA})$ time. Our data structure uses a universal look-up table of size $O(n^g)$ for an arbitrarily small positive constant g.*

Proof. We use the same data structure as in the proof of Lemma 4, but $SA[SA^{-1}[r_1] + depth(w)]$ and $depth(u)$ are computed in $O(t_{SA})$ time. It is not necessary to store $\mathcal{T}$. Information about the heavy path decomposition of $\mathcal{T}_0$ can be stored in $O(f)$ bits. We show how this can be done in [11]. Data structures H_i need $O(\log\log n)$ bits per node.

Since queries on H_j and $D(u)$ are answered in $O(t_{SA})$ time, an unrooted LCP query is also answered in $O(t_{SA})$ time. ◀

The following Lemma is proved in [11].

▶ **Lemma 5.** *There exists a data structure that uses $O(f)$ additional bits of space and answers unrooted LCP queries on $\mathcal{T}_0$ in $O((t_{SA}(\log\log\log n))$ time. Our data structure uses a universal look-up table of size $O(n^g)$ for an arbitrarily small positive constant g.*

4 Wildcard Pattern Queries in Less Space

Now we are ready to describe the compact data structure for wildcard indexing. Our approach is as follows. We divide the suffix tree $\mathcal{T}$ into subtrees, so that each subtree has a poly-logarithmic number of nodes and results of Section 3 can be applied to each subtree. We also keep a tree $\mathcal{T}_m$ that has one representative node for each subtree and stores information about positions of small subtrees in $\mathcal{T}$. Unrooted LCP queries are answered in two steps. First, we identify the small subtree that contains the answer using data structures on $\mathcal{T}_m$. Then we search in the small subtree using the data structure of Section 3. We select the size of subtrees so that $\mathcal{T}_m$ and data structures for $\mathcal{T}_m$ use $O(n)$ bits. A detailed description of our data structure is given below.

Data Structure. Let $\tau = \sigma \log^2 n$. We visit all leaves of the suffix tree $\mathcal{T}$ in left-to-right order and mark every τ-th leaf. We visit all internal nodes of $\mathcal{T}$ in bottom-to-top order and mark each node u such that at least two children of u have marked descendants. Finally the root node is also marked.

We divide the nodes of the suffix tree into groups as follows. Let u be a marked internal node, such that all its non-leaf descendants are unmarked. Each child u_i of u contains at most one marked leaf (because otherwise the subtree rooted at u_i would contain marked internal nodes). The subtrees rooted at children $u_i, \ldots, u_d$ of u are distributed among groups $G_j(u)$. We select indices $i_1 = 1$, i_2, …, $i_t = m$ such that exactly one node among u_{i_j}, …, $u_{i_{j+1}-1}$ has a marked leaf descendant. For each j, $1 \le j < t$, all nodes in the subtrees of $u_{i_j}, \ldots, u_{i_{j+1}-1}$ are assigned to group $G_j(u)$. Every $G_j(u)$ contains $O(\tau)$ nodes. Now suppose that a marked node u has marked descendants. We divide the children of u into groups $G(u,v)$ such that exactly one child u_i of u in each $G(u,v)$ has exactly one direct marked descendant. That is, in every $G(u,v)$ there is exactly one child u_i of u satisfying one of the following two conditions: (i) u_i is marked (in this case u_i is assigned to the group $G(u,u_i)$) or (ii) u_i has exactly one marked descendant v such that there are no other marked nodes between u_i and v. The group $G(u,v)$ also contains all nodes that are descendants of u_i but are not proper descendants of v. To make nodes of $G(u,v)$ a subtree, we also include u into $G(u,v)$. The number of nodes in $G(u,v)$ is also bounded by $O(\tau)$.

Each node $w \in \mathcal{T}$ belongs to some group $G_j(u)$ or $G(v,u)$. The total number of groups is $O(n/\tau)$ because each group can be associated with one marked node. Since every $G_j(u)$ is a subtree, we can answer unrooted LCP queries on the nodes (and locations) of $G_j(u)$ implemented according to Lemma 4. Furthermore we divide every $G(v,u)$ into two overlapping subgroups: $G_l(v,u)$ contains all nodes of $G(v,u)$ that are on the path from v to u or to the left of this path; $G_r(v,u)$ contains all nodes of $G(v,u)$ that are on the path from v to u or to the right of this path. We also add the leftmost and rightmost leaf descendants of the node u, where u is the marked node in $G(v,u)$, to $G_l(v,u)$ and $G_r(v,u)$ respectively. The leaves in each group $G_l(v,u)$ and $G_r(v,u)$ correspond to τ consecutive suffixes. Therefore we can

answer unrooted LCP queries on $G_l(u,v)$ and $G_r(u,v)$ using Lemmas 4 or 5. The answer to an unrooted LCP query on $G(u,v)$ can be obtained from answers to the same query on $G_l(u,v)$ and $G_r(u,v)$. The data structures for unrooted LCP queries on $G_j(u)$, $G_l(u,v)$ and $G_r(u,v)$ will be denoted $D_j(u)$, $D_l(u,v)$ and $D_r(u,v)$ respectively. Each node belongs to at most two groups; therefore all group data structures need $O(n)$ bits of space.

The nodes of the suffix tree are stored in compressed form described in Section 2. The depth and the string depth of any node can be computed in $O(t_{SA})$ time. We can also pre-process an arbitrary pattern in $O(|P|t_{SA})$ time, so that the LCP of any suffixes $P[j..|P|]$ and $T[i..n]$ can be found in $O(t_{SA})$ time.

Moreover, we keep all suffixes that are stored in marked leaves of the suffix tree in a compressed trie $\mathcal{T}_m$. Nodes of $\mathcal{T}_m$ correspond to marked nodes of $\mathcal{T}$. Unrooted LCP queries on $\mathcal{T}_m$ can be answered in $O(\log\log n)$ time using $O((n/\tau)\log^2 n) = O(n/\sigma)$ bits; see Lemma 11 in [11].

In every node of $\mathcal{T}_m$ we store a pointer to the corresponding marked node of $\mathcal{T}$. We also keep a bit vector B that keeps data about marked and unmarked nodes of $\mathcal{T}$; the order of nodes is determined by a pre-order traversal of $\mathcal{T}$. The i-th entry $B[i]$ is set to 1 if the i-th node (in pre-order traversal) is marked, otherwise $B[i]$ is set to 0. Using $o(n)$ additional bits, we can compute the number of preceding 1's for any position in B in $O(1)$ time [13]. Hence for any node $u \in \mathcal{T}$, we can find the number of marked nodes that precede u in the pre-order traversal of $\mathcal{T}$. We also store an array A_m; the i-th entry of A_m contains a pointer to the node of $\mathcal{T}_m$ that corresponds to the i-th marked node in $\mathcal{T}$. Using B and A_m, we can find the node of $\mathcal{T}_m$ that corresponds to a given marked node of $\mathcal{T}$ in $O(1)$ time. We will also need another data structure to facilitate the navigation between marked nodes and its children. For every marked node u with marked internal descendants and for all groups $G(u,v)$, we store the first character on the label of the edge from u to its leftmost child $u_i \in G(u,v)$ in a predecessor data structure.

Queries. Consider an unrooted LCP query (u,P). If u is marked, we find the lowest marked descendant u' of u, such that $str(u,u')$ is a prefix of P. We find the child u_i of u' such that the edge from u' to u_i is labelled with a string s_i and $str(u,u') \circ s_i$ is a prefix of P. Then we use the data structure $D_j(u)$ (respectively $D_l(u,w)$ and $D_r(u,w)$) for the subtree that contains u_i and answer an unrooted LCP query (u_i,P') for P' satisfying $str(u,u') \circ s_i \circ P' = P$. The answer to the latter query provides the answer to the original query (u,P). If u is unmarked, we start by answering the query (u,P) using the data structure for the group that contains u. If the answer is an unmarked node u_1 (or a location $\tilde{u}_1$ on an edge that starts in an unmarked node), then u_1 (respectively $\tilde{u}_1$) is the answer to our query. If u_1 is marked, we answer the query (u_1,P_1), where P_1 is the remaining suffix of P, as described above. Again we obtain the answer to the original query (u,P).

We can report all occurrences of $\tilde{P} = \phi P_1 \phi P_2 \ldots \phi P_d$ by answering at most σ^d unrooted LCP queries and σ^d accesses to the compressed suffix tree. For all alphabet symbols a we find the location of the pattern aP_1 by answering a wildcard LCP query. For each symbol a, such that the location $\tilde{u}_a$ of aP in $\mathcal{T}$ was found, we continue as follows. If $\tilde{u}_a$ is a position on an edge (u_a,u'_a), we check whether the remaining part of the edge label equals aP'_2 for some symbol a and a prefix P'_2 of P_2. If this is the case, we answer a query (u'_a,P''_2) where P''_2 satisfies $P_2 = P'_2 \circ P''_2$. If $\tilde{u}_a$ is a node, we find the loci of patterns $str(\tilde{u}_a) \circ xP_2$, where x denotes any alphabet symbol, as described above. We proceed in the same way until the loci of all $x_1P_1 \ldots x_mP_m$ for any alphabet symbol x_i are found. This approach can be straightforwardly extended to reporting occurrences of a general wildcard expression

$\tilde{P} = \phi^{k_1} P_1 \phi^{k_2} P_2 \dots \phi^{k_d} P_d$, where ϕ^{k_i} denotes an arbitrary sequence of k_i alphabet symbols and $k_i \geq 0$ for $1 \leq i \leq d$.

▶ **Theorem 6.** *There exists an $O(n + s_{small} n)$-bit data structure that reports all* occ *occurrences of a wildcard pattern $\phi^{k_1} P_1 \phi^{k_2} P_2 \dots \phi^{k_d} P_d$ in $O(\sum_{i=1}^{d} |P_i| t_{SA} + \sigma^g t_{small}(n) + \text{occ} \cdot t_{SA})$ time, where $g = \sum_{i=1}^{m} k_i$; s_{small} and t_{small} denote the average space usage and query time of the data structures described in Lemmas 3 or 4.*

Two interesting corollaries of this result are the following indexes. We use the same notation as in Theorem 6. If we combine Lemma 1, (a) with Lemma 5 we get $t_{small} = O(\log^{\varepsilon} n)$ and $s_{small} = O(1)$ (the query time $O(\log^{\varepsilon} n \log^{(3)} n)$ can be simplified to $O(\log^{\varepsilon} n)$ by replacing ε with some $\varepsilon' < \varepsilon$). If we plug in this result into Theorem 6, we obtain our first main data structure.

▶ **Corollary 7.** *There exists an $O(n)$-bit data structure that answers wildcard pattern matching queries in $O((\sum_{i=1}^{d} |P_i| + \sigma^g + \text{occ}) \log^{\varepsilon} n)$ time.*

We remark that the result of Corollary 7 can be also extended to the case of an arbitrarily large alphabet. In this case the index uses $O(n \log \sigma)$ bits and queries are answered in $(\sum_{i=1}^{d} |P_i| + \sigma^g + \text{occ}) \log_{\sigma}^{\varepsilon} n)$ time. This variant can be obtained by using the suffix array of Grossi *et al.* [7]; the compressed suffix tree uses $O(n \log \sigma)$ bits in this case.

If we combine Lemma 1, (b) with Lemma 5 and plug in the result into Theorem 6, we obtain our second main data structure.

▶ **Corollary 8.** *There exists an $O(n(\log \log n)^2)$-bit data structure that answers wildcard pattern matching queries in $O((\sum_{i=1}^{d} |P_i| + \sigma^g + \text{occ}) \log \log n)$ time.*

5 LCP Queries for Patterns with Wildcards, $\sigma = \log \log n$

In the remaining part of this paper we describe faster solutions that use linear or sublinear space. In sections 5 and 6 we describe an $O(n \log n)$-bit data structure for $\sigma \geq \log \log n$. In section 7 we use a more technically involved variant of the same approach to obtain fast solutions for $\sigma < \log \log n$.

In this section we will show how to answer a batch of LCP queries called wildcard LCP queries. A wildcard LCP query $(u, \phi P)$ returns the loci of $str(u) \circ aP$ in the suffix tree of a source text T for all $a \in \Sigma$ such that $str(u) \circ aP$ occurs in T. As before, we assume that we can preprocess some pattern $\overline{P}$ in $O(\overline{P})$ time; then, queries (u, P) where P is a suffix of $\overline{P}$ are answered. The pre-processing is the same as in Section 3.

A leaf descendant v_l of a node u is a light descendant of u if v_l and u are not on the same heavy path. A wildcard tree $\mathcal{T}_u$ for a node u is a compressed trie that contains all strings s satisfying $a \circ s = str(u, v_l)$ for some symbol a and some light leaf descendant v_l of u. The main idea of our approach is to augment the suffix tree $\mathcal{T}$ with wilcard trees in order to accelerate the search. To avoid logarithmic increase in space usage, only selected nodes of wilcard trees will be stored. We explain our method for the case $\sigma = \log \log n$.

Let $\tau = \sigma \log^2 n$. We mark the nodes of the suffix tree in the same way as described in Section 4. Every τ-th leaf of $\mathcal{T}$, each internal node with at least two children that have marked descendants, and the root of $\mathcal{T}$ are marked. The nodes of $\mathcal{T}$ will be called the *alphabet nodes*. We also store selected nodes from wildcard trees, further called *wildcard nodes*. A truncated wildcard tree $\mathcal{T}_u$ is a compressed trie containing all strings s, such that $a \circ s = str(u, v_l)$ for some marked light leaf descendant v_l of u. Each leaf-to-root path intersects $O(\log n)$ heavy paths. Therefore each marked leaf occurs in $O(\log n)$ truncated wildcard trees. Hence the

total number of wildcard nodes is $O((n/\tau)\log n)$. Every node in each truncated wildcard tree contains pointers to some alphabet nodes or locations on edges between alphabet nodes. Suppose that a node v is in a wildcard subtree $\mathcal{T}_w$, the parent of $\mathcal{T}_w$ is some node w, and the label of v in $\mathcal{T}_w$ is s. For every symbol a such that $s_a = str(w) \circ a \circ s$ occurs in the source text, we store a pointer from u to the location u_a of s_a. The total number of pointers is equal to $O(n \log n(\sigma/\tau))$. We distribute alphabet nodes into groups $G_j(u)$ and $G(v,u)$ as described in Section 4; data structures $D_j(u)$, $D_l(v,u)$, and $D_r(v,u)$ are also defined in the same way as in Section 4. Every pointer from a wildcard node to an alphabet node w (or edge (u,w)) contains a reference to the group that contains w. Moreover, both alphabet and wildcard nodes of our extended suffix tree are kept in the data structure, described in Lemma 11 in [11], that answers unrooted LCP queries in $O(\log\log n)$ time.

Queries. Suppose that a wildcard LCP query $(u, \phi P)$ must be answered. Let a_h be the first symbol in $str(u,u_h)$, where u_h is the child of u that is on the same heavy path. We answer a query $a_h \circ P$ in $O(\log\log n)$ time using the result of [1]. Next, we must find the locus nodes of all patterns $a_j \circ P$, $a_j \neq a_h$. We answer an LCP query P in the truncated wildcard tree $\mathcal{T}_u$ of the node u. Let w denote the node where the search for P in $\mathcal{T}_u$ ends and let w_r denote the root node of $\mathcal{T}_u$. The node w can also be found in $O(\log\log n)$ time.

1. Suppose that $str(w_r, w) = P$. We follow pointers from w to alphabet nodes w_1, …, w_σ marked with alphabet symbols a_1,…, a_σ. For each $1 \leq j \leq \sigma$ we find the group $G_r(u_j)$ (or $G(u_j, v_j)$) that contains w_j and answer an LCP query (w_j, P_j) on the tree induced by $G(u_j)$ (respectively $G(u_j, v_j)$). The string P_j is a suffix of P that satisfies $str(u, u_j) \circ P_j = a_j \circ P$. Using information in the pointer from w to u_j, we can find P_j in $O(1)$ time.
2. The pattern P can be also located between two nodes w' and w of $\mathcal{T}_u$ such that $str(w_r, w')$ is prefix of P and P is a prefix of $str(w_r, w)$. For every j, we follow the pointers marked with alphabet symbol a_j. Suppose that pointers from w' and w lead to locations $\tilde{w}'_j$ and $\tilde{w}_j$ respectively. Let w'_j be the lower node on the edge of $\tilde{w}'_j$ and let w_j be the upper node on the edge of $\tilde{w}_j$. There are no marked nodes between w'_j and w_j. Therefore we only need to search in the group that contains w_j to complete the LCP query.

The total search time is $O(\log\log n + \sigma \cdot t_{\text{small}})$ where t_{small} is the time needed to answer an LCP query on a subtree of τ nodes. We use Lemma 3; hence $t_{\text{small}} = O(1)$. Since $\sigma = \log\log n$, a wildcard LCP query is answered in $O(\log\log n) = O(\sigma)$ time.

6 Wildcard Pattern Matching Queries for $\sigma \geq \log\log n$

Wildcard LCP Queries. We can modify the data structure of Section 5 for the case when the alphabet size $\sigma \geq \log\log n$. We divide the alphabet Σ into groups such that every group, except the last one, contains $\log\log n$ elements. The last group contains at most $\log\log n$ elements. We will denote these groups Σ^1, …, Σ^g for $g = \lceil \sigma / \log\log n \rceil$. Instead of one wildcard tree $\mathcal{T}_u$, we will store g modified wildcard trees $\mathcal{T}^1_u, \ldots, \mathcal{T}^g_u$ in every node $u \in \mathcal{T}$. A wildcard tree $\mathcal{T}^i_u$ for a node u is a compressed trie that contains all strings s satisfying $a \circ s = str(u, v_l)$ for some symbol $a \in \Sigma^i$ and some marked light leaf descendant v_l of u. We keep the same data structure for every $\mathcal{T}^i_u$ as in Section 5. Thus we answer LCP queries for each group of $\log\log n$ alphabet symbols in $O(\log\log n)$ time. The total time needed to answer a wildcard LCP query is $O(\lceil \sigma / \log\log n \rceil \log\log n) = O(\sigma)$.

Indexing. Consider a query $\tilde{P} = \phi P_1 \phi P_2 \dots \phi P_d$. If $\sigma \geq \log\log n$, then our data structure for wildcard LCP queries enables us to find all occurrences of $\tilde{P}$ by answering wildcard LCP queries. We find the loci of all $a_i P_1$ for every $a_i P_1$ that occurs in the source text T. This is achieved by answering a wildcard LCP query $(u_r, \phi P_1)$. For every found location u_i^1 we proceed as follows. If u_i^1 is in a middle of an edge e, we move one symbol down and then check whether the remaining symbols of an e are labelled with a prefix of P_2. If this is the case and the remaining part of e is labelled with P_2', we answer a regular LCP query (w_i^1, P_2'') such that w_i^1 is the node at the lower end of e and $P_2 = P_2' \circ P_2''$. Using the data structure of Bille *et al.* [1], an LCP query can be answered in $O(\log\log n)$ time. If u_i^1 is a node in the suffix tree, then we answer a wildcard LCP query $(u_i^1, \phi P_2)$. We continue in the same manner until the loci of all $xP_1 \dots xP_m$, where x denotes an arbitrary symbol in Σ, are found. A general wildcard pattern $\phi^{k_1} P_1 \dots \phi^{k_d} P_d$ is processed in the same way.

Since the maximum number of wildcard LCP queries and standard LCP queries does not exceed σ^g, the total query time is $O(\sigma^g)$. Preprocessing stage for all wildcard LCP queries takes $O(\Sigma_{i=1}^{d} |P_i|)$ time.

▶ **Lemma 9.** *Suppose that the alphabet size $\sigma \geq \log\log n$. Using an $O(n \log n)$-bit data structure, we can report all occurrences of a pattern $\tilde{P} = \phi^{k_1} P_1 \phi^{k_2} P_2 \dots \phi^{k_d} P_d$ in $O(\sum_{i=1}^{d} |P_i| + \sigma^g + \mathrm{occ})$ time, where* occ *is the number of times $\tilde{P}$ occurs in the text and $g = \sum_{i=1}^{d} k_i$.*

7 Wildcard Pattern Matching Queries for Small Alphabets

In this section we consider the case when the alphabet size $\sigma < \log\log n$. We use the approach of Sections 5 and 6, but the notion of wildcard LCP queries is generalized. A t-wildcard LCP query $(u, \tilde{P})$ for a wildcard string $\tilde{P} = \phi^{k_1} P_1 \phi^{k_2} P_2 \dots \phi^{k_d} P_d$ such that $\sum k_i = t$, finds locations of all patterns $str(u) \circ P$, where $P = s_1 s_2 \dots s_{k_1} P_1 s_{k_1+1} \dots s_{k_2} P_2 \dots s_{t-1} s_t P_d$ and s_i, $1 \leq i \leq t$, are arbitrary alphabet symbols, in the suffix tree. A 1-wildcard LCP query, used in the previous sections, takes $O(\log\log n)$ time and can replace up to σ standard wildcard queries. Hence, when the alphabet size σ is small, we cannot achieve noteworthy speed-up in this way. A t-wildcard LCP query can replace up to σ^t regular LCP queries and lead to more significant speed-up even when σ is very small. We will use iterated wildcard subtrees in order to support s-wildcard LCP queries efficiently. Our construction consists of two parts. We mark selected nodes in the suffix tree $\mathcal{T}$ and divide it into subtrees $\mathcal{T}_i$ of size $O(\tau_1)$; we keep a data structure that supports t_1-wildcard LCP queries on the subtree $\mathcal{T}^m$ induced by marked nodes of $\mathcal{T}$. We also mark selected nodes, further called secondary marked nodes, in each subtree $\mathcal{T}_i$ and divide $\mathcal{T}_i$ into $\mathcal{T}_{i,j}$ of size $O(\tau_2)$. Let $\mathcal{T}_i^m$ be the subtree induced by secondary marked node of $\mathcal{T}_i$; we keep a data structure that answers standard wildcard LCP queries on $\mathcal{T}_i^m$. Details of our data structure and parameter values can be found below.

Trees $\mathcal{T}_i$ and $\mathcal{T}^m$. Let $t_1 = \log_{\sigma/2} \log\log n$ and $\tau_1 = \sigma^{t_1} \log^{t_1+1} n$. We use the same scheme as in Section 4 to mark every τ_1-th leaf and selected internal nodes, so that the suffix tree $\mathcal{T}$ is divided into subtrees $\mathcal{T}_i$ of size $O(\tau_1)$ and the number of marked nodes is $O(n/\tau_1)$. Trees $\mathcal{T}_i$ correspond to groups $G_j(u)$ and $G(u, v)$ defined in section 4.

Let $\mathcal{T}^m$ be the tree induced by marked nodes. We iteratively augment $\mathcal{T}^m$ with wildcard subtrees. For any marked internal node u, the (level-1) wildcard subtree $\mathcal{T}_u$ is a compressed trie containing all strings s, such that $a \circ s = str(u, v_l)$ for some marked light leaf descendant v_l of u. We also keep a level-$(i+1)$ wildcard subtree $\mathcal{T}_w$ for every node w in a level-i wildcard subtree $\mathcal{T}_u$. $\mathcal{T}_w$ contains all strings s such that $a \circ s = str(u, v_l)$ for some alphabet symbol a and a light leaf descendants v_l of w. We construct level-i wildcard subtrees for $1 \leq i \leq t_1$.

The parameter t_1 is chosen in such way that $\sigma^{t_1} = 2^{t_1} \log\log n$ and $t = \log_\sigma \log\log n$. Every node in all level-i wildcard trees has pointers to the corresponding locations in the alphabet tree $\mathcal{T}$. Each pointer also contains information about the subtree $\mathcal{T}_i$

The total number of nodes and pointers in wildcard subtrees is $(n/\tau_1)\sigma^{t_1} \log^{t_1} n$. Level-$t$ wildcard subtrees can be used to answer unrooted t-wildcard LCP queries on $\mathcal{T}_m$ in $O(2^t \log\log n)$ time; our method is quite similar to the procedure for answering wildcard queries in [3]. Consider a query $(\tilde{u}, \tilde{P})$, where $\tilde{u}$ is a location in the alphabet tree or in some i-wilcard subtree. We distinguish between the following four cases. (i) If $\tilde{u}$ is on a tree edge and the next symbol is a wildcard, we simply move down by one symbol along that edge. (ii) Suppose that $\tilde{u}$ is on a tree edge e and the next symbols are a string P_n of alphabet symbols. Let l denote the string label of the part of e below $\tilde{u}$, $l = str(\tilde{u}, u')$ where u' is the lower node on e. We compute $o = LCP(P_n, l)$. and move down by $\min(|l|, o)$ symbols along e. (iii) If $\tilde{u}$ is a node and the next unprocessed symbol in $\tilde{P}$ is a wildcard, our procedure branches and visits two locations: we move down by one symbol along the edge to the heavy child of $\tilde{u}$ and visit the root of the wildcard tree $\mathcal{T}_{\tilde{u}}$ (if $\tilde{u}$ is on a level-i wildcard tree, we visit the root of the $(i+1)$-subtree $\mathcal{T}_{\tilde{u}}$). (iv) If $\tilde{u}$ is a node and the next symbols are a string P_n of alphabet symbols, we answer a standard LCP query $(\tilde{u}, P_n)$. The procedure is finished when we cannot move down from any location that is currently visited. The number of branching points is 2^t and we answer 2^t standard LCP queries. We need $O(\sigma^t)$ time to return from locations in wildcard trees to the corresponding locations in the alphabet tree. Thus the total time is $O(2^t \log\log n + \sigma^t) = O(\sigma^t)$. When the search in $\mathcal{T}^m$ is completed we can continue searching in subtrees $\mathcal{T}_j$. Data structures for subtrees $\mathcal{T}_i$ are described in [11], where we show that an unrooted LCP query on $\mathcal{T}_i$ can be answered in $O((\log^{(3)} n)^{1/2})$ time.

Wildcard String Matching. It follows from the above description that we can answer t_1-wildcard LCP queries in $O(\sigma^{t_1}\sqrt{\log^{(3)} n})$ time. Consider now an arbitrary pattern $\tilde{P} = \phi^{k_1} P_1 \phi^{k_2} P_2 \ldots \phi^{k_d} P_d$. We divide it into chunks $\tilde{P}[1]$, $\tilde{P}[2]$, …, $\tilde{P}[r]$, such that each chunk $\tilde{P}[i]$, $i \geq 2$, contains exactly t_1 wildcard symbols. The chunk $P[1]$ contains $v \leq t_1$ wildcard symbols.

We start at the root and find locations of all $\tilde{P}[1] = \phi^{k_1} P_1 \ldots \phi^{k_f} P_f \phi^r$ where $r \leq k_{f+1}$. If $\sum_{i=1}^{f} |P_f| > (\log\log n) \cdot \sigma^t$, we answer at most σ^t standard LCP queries in $O(\sigma^t \log\log n) = O(\sum_{i=1}^{f} |P_i|)$ time. If $\sum_{i=1}^{f} |P_i| \leq (\log\log n) \cdot \sigma^t$, then the total length of $\tilde{P}[1]$ is at most $\ell = (\log\log n) \cdot \sigma^t + t$. Since $\sigma < \log\log n$, there are $O((\log\log n)^\ell)$ different patterns and each of this patterns fits into one machine word. Hence, all string patterns P_s that match $\tilde{P}[1]$ can be generated in $O(\sigma^v)$ time. We keep a look-up table with locations of all strings P, such that $|P| \leq \ell$ in $\mathcal{T}$. Using this table we find locations of all P_s that match $\tilde{P}[1]$ and occur in the source text. For every such location $\tilde{u}$, we answer queries $(\tilde{u}_1, \tilde{P}[2])$, $(\tilde{u}_2, \tilde{P}[3])$, …, where $\tilde{u}_1 = \tilde{u}$ and $\tilde{u}_i$ for $i > 1$ is an answer to some query $(\tilde{u}_{i-1}, \tilde{P}[i])$. It is easy to show that the total query time is $O(\sum_{i=1}^{d} |P_i| + \sigma^g \sqrt{\log^{(3)} n} + \mathrm{occ})$.

▶ **Theorem 10.** *If the alphabet size $\sigma = O(1)$ and $\sigma > 2$, then there exists an $O(n \log^\varepsilon n)$-bit data structure that reports all* occ *occurrences of a wildcard pattern $\phi^{k_1} P_1 \phi^{k_2} P_2 \ldots \phi^{k_d} P_d$ in $O(\sum_{i=1}^{d} |P_i| + \sigma^g \sqrt{\log^{(3)} n} + \mathrm{occ})$ time.*

We remark that the same query time as in Theorem 10 can be also achieved for a non-constant σ; the space usage would grow to $O(n \log n)$ bits, however. To obtain this result, we would need to use standard (uncompressed) suffix tree and suffix array for the source data.

Acknowledgement. The second author wishes to thank Gonzalo Navarro for pointing him to [15].

References

1 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind. String indexing for patterns with wildcards. In *Proc. 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2012)*, pages 283–294, 2012.

2 Ho-Leung Chan, Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, and Swee-Seong Wong. A linear size index for approximate pattern matching. *J. Discrete Algorithms*, 9(4):358–364, 2011.

3 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proc. 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 91–100, 2004.

4 Johannes Fischer, Veli Mäkinen, and Gonzalo Navarro. Faster entropy-bounded compressed suffix trees. *Theor. Comput. Sci.*, 410(51):5354–5364, 2009.

5 Michael L. Fredman and Dan E. Wilard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.

6 Roberto Grossi, Alessio Orlandi, Rajeev Raman, and S. Srinivasa Rao. More haste, less waste: Lowering the redundancy in fully indexable dictionaries. In *Proc. 26th Int'l Symp. on Theoretical Aspects of Computer Science (STACS 2009)*, pages 517–528, 2009.

7 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005.

8 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

9 Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, and Siu-Ming Yiu. Space efficient indexes for string matching with don't cares. In *Proc. 18th International Symposium on Algorithms and Computation (ISAAC 2007)*, pages 846–857, 2007.

10 Moshe Lewenstein, J. Ian Munro, Venkatesh Raman, and Sharma V. Thankachan. Less space: Indexing for queries with wildcards. In *Proc. 24th International Symposium on Algorithms and Computation (ISAAC 2013)*, pages 89–99, 2013.

11 Moshe Lewenstein, Yakov Nekrich, and Jeffrey Scott Vitter. Space-efficient string indexing for wildcard pattern matching. *CoRR*, abs/1401.0625, 2014.

12 Veli Mäkinen and Gonzalo Navarro. Compressed text indexing. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2008.

13 J. Ian Munro. Tables. In *Proc. 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1996)*, pages 37–42, 1996.

14 M. Sohel Rahman and Costas S. Iliopoulos. Pattern matching algorithms with don't cares. In *Proc. 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2007)*, pages 116–126, 2007.

15 S. Srinivasa Rao. Time-space trade-offs for compressed suffix arrays. *Inf. Process. Lett.*, 82(6):307–311, 2002.

16 Luís M. S. Russo, Gonzalo Navarro, and Arlindo L. Oliveira. Fully compressed suffix trees. *ACM Transactions on Algorithms*, 7(4):53, 2011.

17 Kunihiko Sadakane. Compressed text databases with efficient query algorithms based on the compressed suffix array. In *Proc. 11th International Conference on Algorithms and Computation (ISAAC 2000)*, pages 410–421, 2000.

18 Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory Comput. Syst.*, 41(4):589–607, 2007.

19 Alan Tam, Edward Wu, Tak Wah Lam, and Siu-Ming Yiu. Succinct text indexing with wildcards. In *Proc. 16th International Symposium on String Processing and Information Retrieval (SPIRE 2009)*, pages 39–50, 2009.

Synchronizing Relations on Words

Diego Figueira and Leonid Libkin

University of Edinburgh, UK

Abstract

While the theory of languages of words is very mature, our understanding of *relations* on words is still lagging behind. And yet such relations appear in many new applications such as verification of parameterized systems, querying graph-structured data, and information extraction, for instance. Classes of well-behaved relations typically used in such applications are obtained by adapting some of the equivalent definitions of regularity of words for relations, leading to non-equivalent notions of recognizable, regular, and rational relations.

The goal of this paper is to propose a systematic way of defining classes of relations on words, of which these three classes are just natural examples, and to demonstrate its advantages compared to some of the standard techniques for studying word relations. The key idea is that of a *synchronization* of a pair of words, which is a word over an extended alphabet. Using it, we define classes of relations via classes of regular languages over a fixed alphabet, just $\{1, 2\}$ for binary relations. We characterize some of the standard classes of relations on words via finiteness of parameters of synchronization languages, called shift, lag, and shiftlag. We describe these conditions in terms of the structure of cycles of graphs underlying automata, thereby showing their decidability. We show that for these classes there exist canonical synchronization languages, and every class of relations can be effectively re-synchronized using those canonical representatives. We also give sufficient conditions on synchronization languages, defined in terms of injectivity and surjectivity of their Parikh images, that guarantee closure under intersection and complement of the classes of relations they define.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Word Relations, Regular, Rational, Recognizable

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.518

1 Introduction

Foundations of formal language theory have been largely developed in the 1960s and 1970s, and used heavily in practically all areas of computer science. The field itself stayed somewhat dormant for a while, but that changed over the past 10–15 years due to new application areas requiring techniques that could not have been foreseen 30 or 40 years earlier. Among consumers of results in formal language theory are verification (for instance, automata-based approaches to model-checking are now part of standard industrial verification tools [7, 22]) and data management (standards for describing and querying XML documents, for instance, are rooted in both word and tree automata [24, 28], and emerging graph data models are borrowing many formal language concepts [3]).

Of interest to us in this paper are *relations on words*. That is, for a given finite alphabet $\mathbb{A}$, we deal with binary relations $R \subseteq \mathbb{A}^* \times \mathbb{A}^*$. Their study goes back to Elgot, Mezei, Nivat in the 1960s [15, 25] with much subsequent work done later (see, e.g., surveys [8, 13]). The standard notions of regularity that generate the same class of languages —recognizability by finite monoids, definability by automata, or by regular expressions— give rise to different classes of relations, called *recognizable, regular, and rational* relations. Their properties may differ significantly from properties of regular languages: for instance, rational relations are



© Diego Figueira and Leonid Libkin;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 518–529



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



not closed under intersection and it is even undecidable whether the intersection of two such languages is non-empty. Recognizable relations are just unions of products of regular languages; examples of regular relations are prefix, equality, or equal length of words; and examples of rational relations are suffix, subword (for instance, *bb* is a subword of *aabbaa*), and subsequence (*bb* is a subsequence of *abaaba*: letters need not be consecutive).

There has been renewed interest in relations on words as of late. One motivation comes from verification of safety and liveness properties of parameterized systems, where such relations describe transitions [1, 10, 20, 29]. Another comes from graph databases, which are actively studied as a suitable model for RDF data, social networks data, and others [3]. Paths in graph databases are described by their labels, and need to be compared, for instance, for their degree of similarity, e.g., their edit distance [4, 6, 23]. Yet another example is the study of formal models underlying IBM's tools for information extraction [16].

Many of the basic questions that arise in these new applications, however, are not the kind of questions that had been addressed previously. Just to give an example, it is well known that checking nonemptiness of the intersection of a rational relation and a regular relation is an undecidable problem. But what about *really used* rational relations such as subword, suffix, subsequence (as opposed to artificial codings of the halting problem) – can we test if their intersection with regular relations is nonempty? However natural these questions are, they were answered only recently [5].

An even more basic question relates to the very choice and structure of the main classes of relations: recognizable, regular, and rational. They appeared in a somewhat ad hoc way, just as analogs of different ways of defining regularity of languages, but is there another way to explain these, and perhaps other classes as well? This is the main point of our paper: we argue that there is a natural way to study relations on words, and we do it by explaining how positions in words are *synchronized*.

As an example of synchronization, consider words $w_1 = ababb$ and $w_2 = baaaba$. We can represent this pair as a single word over $\{a, b\}$, by shuffling w_1 and w_2, i.e., interspersing letters of w_1 among letters of w_2. For each position in the shuffle, we remember which word it came from – this is indicated by the symbols 1 or 2 above the letters in the figure.

w_1	a	b	a	b	b						
w_2	b	a	a	a	b	a					
	1	2	2	1	2	1	1	2	2	1	2
	a	b	a	b	a	a	b	a	b	b	a

When we read the letters marked i, for $i = 1, 2$ we get the word w_i. The word over $\{1, 2\}$ provides a *synchronization* of the pair (w_1, w_2) – in our example, 12212112212. We show that the commonly occurring classes of relations over words follow the same principle:

1. to decide whether (w_1, w_2) is in the relation, one runs an automaton over the shuffle;
2. classes of relations are then determined by the classes of allowed synchronizations.

For instance, recognizable relations are given by synchronizations from 1^*2^*, length-preserving regular relations by synchronizations from $(12)^*$, arbitrary regular relations by synchronizations from $(12)^*(1^*|2^*)$, and rational relations by synchronizations from $(1|2)^*$.

For relations, we have proper inclusions *recognizable* $\subsetneq$ *regular* $\subsetneq$ *rational* [8], making them very different from languages. This immediately raises the question: since every recognizable language is regular, and yet 1^*2^* is *not* contained in $(12)^*(1^*|2^*)$, there must be multiple ways of synchronizing relations to obtain even known classes. What are these ways, and how can they be characterized? And will those characterizations lead to new naturally appearing classes?

These are the questions we answer. We define three parameters of regular languages in $(1|2)^*$: the *shift* says how often we switch between 1s and 2s, the *lag* says how big the difference between the numbers of 1 and 2 is allowed to get, and *shiftlag* combines the two in a certain way. Then finite shift characterizes recognizability, while finite shiftlag characterizes regularity of relations. Finite lag, which appears to be a natural measure then, captures another known class of relations.

We provide automata characterizations of classes of synchronization languages in terms of the structure of cycles in the graph representations of automata. All these turn out to be decidable. This shows one advantage of dealing with relations in terms of their synchronizations. For instance, it is known that checking whether a given rational relation is regular, is an undecidable problem (assuming the input is a transducer, i.e., an automaton with output [8]). However, if the input to the problem is a synchronization language, then it *is* decidable whether the relations it describes are all regular.

Another advantage of describing relations by their synchronizations is the ability to find classes closed under intersection or complementation (rational relations, for instance, are not). We do it by imposing decidable conditions on Parikh images of synchronization languages to guarantee closure properties of classes of relations they give rise to.

We also look at re-synchronization of relations. For each class of relations, there may be many different regular synchronizing languages over $\{1, 2\}$. We show that in the standard cases, there exist canonical synchronizing languages, and relations can be effectively resynchronized using those canonical languages.

2 Recognizable, regular, and rational relations

We start with some basic notations. Throughout the paper, $\mathbb{A}$ stands for a finite alphabet, $\mathbb{N} = \{1, 2, \dots\}$ for the set of positive natural numbers, and $\mathbb{N}_0$ for $\mathbb{N} \cup \{0\}$. The set of all words over $\mathbb{A}$ is denoted by $\mathbb{A}^*$, and the length of w in $\mathbb{A}^*$ is denoted by $|w|$. If $w = a_1 \dots a_n$, then $w[i, j]$ stands for the subword $a_i \dots a_j$; in particular, $w[i]$ is the letter a_i.

Recall that there are three standard ways of defining regular languages:

- Recognizability by finite monoids: the set $\mathbb{A}^*$, equipped with the concatenation operation (denoted by '$\cdot$', whose unit is the empty word 'ε') is a monoid. A set $L \subseteq \mathbb{A}^*$ is recognizable if there is a finite monoid M and a homomorphism $\langle \mathbb{A}^*, \cdot, \varepsilon \rangle \to M$ so that $L = f^{-1}(M_0)$ for some $M_0 \subseteq M$.
- Definability by finite automata, say NFAs.
- Definability by regular (sometimes called rational) expressions, i.e., those built from the empty word and alphabet letters using union, concatenation, and the Kleene star.

Classical formal language theory tells us that these definitions generate the same class of languages, known as regular languages. We now adapt them to binary relations on words.

Recognizable relations Since $\langle \mathbb{A}^*, \cdot, \varepsilon \rangle$ is a monoid, $\mathbb{A}^* \times \mathbb{A}^*$ has the structure of a monoid too. We can thus define *recognizable relations* as sets $R \subseteq \mathbb{A}^* \times \mathbb{A}^*$ for which there is a finite monoid M and a morphism $f : \mathbb{A}^* \times \mathbb{A}^* \to M$ such that $R = f^{-1}(M_0)$ for some $M_0 \subseteq M$. This class will be denoted by REC.

Regular relations Let $\perp \notin \mathbb{A}$ be a new alphabet letter. A pair (w_1, w_2) of words from $\mathbb{A}^*$ can be encoded by a single word of length $\max(|w_1|, |w_2|)$ over the alphabet $(\mathbb{A} \cup \{\perp\}) \times (\mathbb{A} \cup \{\perp\})$: its ith letter is the pair containing the ith letter of w_1 and the ith letter of w_2, with $\perp$ used when i is greater than the length of w_1 or w_2. For example, the encoding for the words of the figure of page 519 is $(a, b)(b, a)(a, a)(b, a)(b, b)(\perp, a)$. A regular relation

R is given by an automaton over this alphabet: it contains pairs (w_1, w_2) whose encodings are accepted by the automaton. The class of regular relations is denoted by REG.

Rational relations There are two equivalent ways of defining them. One uses regular expressions, which are now built from pairs in $(\mathbb{A} \cup \{\varepsilon\}) \times (\mathbb{A} \cup \{\varepsilon\})$ using the same operations of union, concatenation, and Kleene star. Alternatively, rational relations can be defined by means of 2-tape automata, that have 2 heads for the tapes and one additional control; at every step, based on the state and the letters it is reading, the automaton can enter a new state and move some (not necessarily all) tape heads. The class of rational relations is denoted by RAT.

Relations in REC are exactly the finite unions of products of regular languages over $\mathbb{A}$ [8, 15]. Examples of relations in REG $\setminus$ REC are prefix, equality, or equal length. Examples of relations in RAT $\setminus$ REG are suffix, given by $\big(\bigcup_{a\in\mathbb{A}}(\varepsilon, a)\big)^* \cdot \big(\bigcup_{a\in\mathbb{A}}(a, a)\big)^*$; subword: $\big(\bigcup_{a\in\mathbb{A}}(\varepsilon, a)\big)^* \cdot \big(\bigcup_{a\in\mathbb{A}}(a, a)\big)^* \cdot \big(\bigcup_{a\in\mathbb{A}}(\varepsilon, a)\big)^*$, and subsequence: $\big(\bigcup_{a\in\mathbb{A}}(\varepsilon, a) \cup (a, a)\big)^*$.

Note that unlike in the case of languages, where the three notions coincide, we have REC $\subsetneq$ REG $\subsetneq$ RAT. The classes REC and REG are closed under intersection; however the class of rational relations is not. In fact, one can find $R \in$ REG and $S \in$ RAT so that $R \cap S \notin$ RAT. However, if $R \in$ REC and $S \in$ RAT, then $R \cap S \in$ RAT.

Relations in REC and REG inherit all the closure/decidability properties of regular languages. If $R \in$ RAT, then each of its projections is a regular language, and can be effectively constructed. Hence, the nonemptiness problem is decidable for RAT. However, testing nonemptiness of the intersection of two rational relations is undecidable. We refer to [8, 12, 27] for basic information on these relations and their decision problems.

3 Synchronizations of relations

We now formalize the idea of synchronizations informally described in the introduction. We write $\mathbf{k}$ for the set $\{1, \ldots, k\}$. A *synchronization* of a pair (w_1, w_2) of words in $\mathbb{A}^*$ is a word over $\mathbf{2} \times \mathbb{A}$ so that the projection on $\mathbb{A}$ of positions labeled i is exactly w_i, for $i = 1, 2$ (see the figure on page 519). Every word w in $(\mathbf{2} \times \mathbb{A})^*$ is a synchronization of a uniquely determined pair (w_1, w_2), where w_i is the sequence of $\mathbb{A}$-letters corresponding to the symbol i in the first position of $\mathbf{2} \times \mathbb{A}$. We denote such (w_1, w_2) by $[\![w]\!]$ and extend it to languages $S \subseteq (\mathbf{2} \times \mathbb{A})^*$ by $[\![S]\!] = \{[\![w]\!] \mid w \in S\}$.

For two words $u = a_1 \cdots a_n \in \mathbb{A}^*$ and $v = b_1 \cdots b_n \in \mathbb{B}^*$, we write $u \otimes v$ for the word $(a_1, b_1) \cdots (a_n, b_n) \in (\mathbb{A} \times \mathbb{B})^*$. The main idea of our approach to relations on words comes from two different ways of viewing words in $(\mathbf{2} \times \mathbb{A})^*$.

- Every word $w \in (\mathbf{2} \times \mathbb{A})^*$ is a synchronization of a pair $[\![w]\!] = (w_1, w_2)$.
- Every word $w \in (\mathbf{2} \times \mathbb{A})^*$ is of the form $u \otimes v$ with $u \in \mathbf{2}^*$ and $v \in \mathbb{A}^*$.

This makes it possible to define relations consisting of pairs $[\![w]\!]$ with restricted synchronizations, i.e., $w = u \otimes v$ and u belongs to a given language $L \subseteq \mathbf{2}^*$.

Formally, if $L \subseteq \mathbf{2}^*$, we say that $u \otimes v$ is *L-controlled* if $u \in L$; a language is L-controlled if all its words are. We now look at relations given by L-controlled synchronizations, i.e., for a regular language $L \subseteq \mathbf{2}^*$, let

$$\textsc{Rel}(L) \;=\; \{[\![S]\!] \;\mid\; S \text{ is a regular } L\text{-controlled language}\} \tag{1}$$

If $\mathcal{C}$ is a class of relations over $\mathbb{A}^*$, then $L \subseteq \mathbf{2}^*$ is a *synchronization* for $\mathcal{C}$ if $\textsc{Rel}(L) \subseteq \mathcal{C}$, that is, all relations given by L-controlled synchronizations belong to $\mathcal{C}$. We remark that a

similar approach to defining relations was used in [18], although the questions considered were completely different.

Procedurally, each relation in $\textsc{Rel}(L)$ is obtained as follows:

1. Choose an automaton over $\mathbf{2} \times \mathbb{A}$;
2. consider words $u \otimes v$ it accepts so that $u \in L$,
3. view v as a synchronization of (w_1, w_2) and add the pair to the relation.

This view suggests natural candidates for capturing classes $\mathsf{REC}, \mathsf{REG}$, and RAT. For REC, relations are unions of products of regular languages, so synchronizations are of the form 1^*2^*: one starts by going over the first word, and then over the second. For REG, they are from $(12)^*(1^*|2^*)$: we first go over two words letter-by-letter, and then write out the rest of the longer word. For RAT, there are no restrictions. Indeed, we can show the following.

▶ Proposition 1.

(I) $\textsc{Rel}(1^*2^*) = \mathsf{REC}$.
(II) $\textsc{Rel}((12)^* \cdot (1^*|2^*)) = \mathsf{REG}$.
(III) $\textsc{Rel}((1|2)^*) = \mathsf{RAT}$.

It is easy to see that $\textsc{Rel}(L)$ is closed under union, alphabetic morphisms, and inverse alphabetic morphisms, and that $L_1 \subseteq L_2$ implies $\textsc{Rel}(L_1) \subseteq \textsc{Rel}(L_2)$.

▶ Remark. One may ask why we need to take both S and L regular in the definition (1) of $\textsc{Rel}(L)$. The reason why S needs to be regular is that even with regular L (e.g., 1^*), $\textsc{Rel}(L)$ would otherwise contain non-rational relations (e.g., $\{(a^n b^n, \varepsilon) \mid n \in \mathbb{N}\}$). If, on the other hand, L is not regular, strange things may happen. For instance, it could be that all relations in $\textsc{Rel}(L)$ are finite, although L is infinite. Indeed, take L as the set of all words 1^p for prime p. Note that there is no infinite regular L-controlled language, since it would imply that an infinite number of distinct primes is semi-linear. Thus, all regular L-controlled languages are finite, and $\textsc{Rel}(L)$ is the set of all finite relations on $\mathbb{A}^* \times \{\varepsilon\}$ so that the first component is of prime length.

4 Synchronizations for recognizable, regular, and rational relations

We have seen examples of languages characterizing the classes of recognizable, regular, and rational relations, but those are not unique. There are trivial examples such as $\textsc{Rel}(1^*2^*) = \textsc{Rel}(2^*1^*) = \mathsf{REC}$, and $\textsc{Rel}((12)^*(1^*|2^*)) = \textsc{Rel}((21)^*(1^*|2^*)) = \mathsf{REG}$, but others as well, e.g., $\textsc{Rel}(1^*2^*1^*2^*)$ equals REC, and $\textsc{Rel}(((12)^*1(12)^*2)^*(1^*|2^*)) = \mathsf{REG}$.

What kind of parameters guarantee that $L \subseteq \mathbf{2}^*$ synchronizes relations in a class $\mathcal{C}$, for the classes we study here? That is, what parameters guarantee that with the synchronization language L, we are guaranteed that the resulting relations are in $\mathcal{C}$?

We now answer this question, but first we need some definitions. Given a word w over some finite alphabet, and a letter a in the alphabet, we define $\#_a(w)$ as the number of occurrences of a in w. Given a word $w \in \mathbf{2}^*$, a position $i \leq |w|$, and $\delta \in \mathbb{N}$, we say i is

- δ-*lagged* if $|\#_1(w[1,i]) - \#_2(w[1,i])| = \delta$;
- $\geq\delta$-*lagged* if $|\#_1(w[1,i]) - \#_2(w[1,i])| \geq \delta$;
- $\leq\delta$-*lagged* if $|\#_1(w[1,i]) - \#_2(w[1,i])| \leq \delta$.

That is, these parameters show by how much the numbers of 1s and 2s in $w \in \mathbf{2}^*$ differ.

A *shift* of w is a position $i \in \{1, \ldots, |w| - 1\}$ so that $w[i] \neq w[i+1]$. Two shifts $i < j$ are *consecutive* if there is no shift l so that $i < l < j$.

Let $\mathit{shift}(w)$ be the number of shifts of w, let $\mathit{lag}(w)$ be the maximum lag of a position in w, and let $\mathit{shiftlag}(w)$ be the maximum $n \in \mathbb{N}$ so that w contains n consecutive shifts

which are $>n$-lagged. We lift these notions to languages by taking maxima, e.g., $\mathit{shift}(L) = \max_{w \in L} \mathit{shift}(w)$, and likewise for $\mathit{lag}(L)$ and $\mathit{shiftlag}(L)$. If words of arbitrarily large lag (shift, or shiftlag) occur in L, we write $\mathit{shift}(L) = \infty$ (and likewise for the other parameters).

Observe that finite shift and finite lag imply that shiftlag is finite, but the converse is not true: for $L = (12)^*1^*$ we have $\mathit{shiftlag}(L) < \infty$ and yet $\mathit{lag}(L) = \mathit{shift}(L) = \infty$.

It turns out that finiteness of the shiftlag parameter corresponds to synchronizing regular languages, and finiteness of shift corresponds to synchronizing recognizable languages. An arbitrary regular $L \subseteq \mathbf{2}^*$ is guaranteed to synchronize rational languages.

As for the finite lag, it corresponds to a class of languages that is known as well. The class REG^{bld} of *bounded length discrepancy* relations [17, 27] is defined as follows. Recall the definition of rational relations using two-tape automata. For a rational relation to be in REG^{bld} it is required that there be $\delta \geq 0$ so that in accepting runs of such automata, the heads for the two tapes are never more than δ positions apart. It also follows from [17, 27] that REG^{bld} is the class $\bigcup_{k \in \mathbb{N}_0} \textsc{Rel}(L_k)$, for $L_k = (12)^*(1^k|2^k)$. Note that $\textsc{Rel}(L_0)$ is the class of length preserving relations. A closely related class $R_{\leq} = \{(w_1, w_2) \in \mathbb{A}^* \times \mathbb{A}^* \mid |w_1| \leq |w_2|\}$ [21] can be equally defined by $\textsc{Rel}((12|2)^*)$.

Now we can state the characterization result.

▶ **Theorem 1.** *Let $L \subseteq \mathbf{2}^*$ be a regular language. Then:*

(I) *L synchronizes regular relations iff $\mathit{shiftlag}(L) < \infty$,*
(II) *L synchronizes recognizable relations iff $\mathit{shift}(L) < \infty$,*
(III) *L synchronizes relations in REG^{bld} iff $\mathit{lag}(L) < \infty$,*
(IV) *L synchronizes rational relations.*

Proof idea. For the 'if' direction of (1), one can easily show that for any regular language L with $\mathit{shiftlag}(L) < n$ there is some δ so that $L \subseteq L'$ for $L' = L_{\leq \delta\text{-lag}} \cdot (1^*|2^*)^n$, where $L_{\leq \delta\text{-lag}}$ is the (regular) language of all words with $\leq\delta$-lagged positions. On the other hand, it is easy to show that $\textsc{Rel}(L') = \mathsf{REG}$. Since $L \subseteq L'$, by applying monotonicity, we then have $\textsc{Rel}(L) \subseteq \mathsf{REG}$.

For the 'only if' direction of (1), suppose that $\mathit{shiftlag}(L) = \infty$. Note that this means that for every $s, \delta \in \mathbb{N}$ there is some $w \in L$ that has s consecutive shifts $>\delta$-lagged. Let $S \subseteq (\mathbf{2} \times \{a, b\})^*$ consist of all words $u \otimes v \in (\mathbf{2} \times \{a, b\})^*$ so that $u \in L$, and for every $i \in \{1, \ldots, |v|\}$, we have $v[i] = a$ if i is a shift of u, and $v[i] = b$ otherwise. One can show that S is an L-controlled relation so that $[\![S]\!] \in \mathsf{RAT} \setminus \mathsf{REG}$. ◀

We conclude the section with a couple of examples of applications of the main result. First, we show that $\textsc{Rel}((112)^*) \not\subseteq \mathsf{REG}$. Indeed, note that for every s, δ, the word $w = (112)^{\delta+s}$ is in $(112)^*$ and the last s shifts of w are $\geq\delta$-lagged. Hence, there must be some L-controlled regular language $S \subseteq (\mathbf{2} \times \mathbb{A})^*$ so that $[\![S]\!]$ is *not* a regular relation.

As another example, we get more ways of synchronizing regular relations: given $L_1 = (1^k \cdot 2^k)^*$, $L_2 = (1^* \cdot 2^*)^k$ for some fixed k, we have $\textsc{Rel}(L_i) \subseteq \mathsf{REG}$ (in fact, $\textsc{Rel}(L_2) \subseteq \mathsf{REC}$).

Finally, we consider the (r/s)-synchronized relations [27, p.660] studied in [11]. This class can be defined as $\textsc{Rel}(L_{r/s})$, where

$$L_{r/s} = (1^r 2^s)^*\Big(\bigcup_{r' < r} (1^{r'} 2^*) \mid \bigcup_{s' < s} (1^* 2^{s'}) \Big). \tag{2}$$

It is easy to see that $\mathit{shiftlag}(L_{r/s}) = \infty$ whenever $r \neq s$, and hence that (r/s)-synchronized relations (with $r \neq s$) are not in REG.

4.1 Automata theoretic characterizations

We characterized classes of relations via conditions imposed on their synchronization languages: finite shift, lag, or shiftlag. Now we show that these conditions themselves can be characterized using automata, or more precisely, the underlying labeled graphs of automata. It turns out that the structure of the cycles provides the desired characterizations.

Since in this section we deal with synchronization languages, we consider automata over the alphabet $\{1, 2\}$. For a given NFA A, we consider the *transition graph* G_A of A as the usual representation of the transition relation, where G_A is a directed graph where states are vertices and edges are labeled by transitions. Given a cycle C of G_A, we define $\#_a(C)$ as the number of edges in C labeled with transitions reading letter a. In a *heterogeneous* cycle C we have $\#_1(C) > 0$ and $\#_2(C) > 0$; otherwise a cycle is *homogeneous*. A cycle C is *balanced* if $\#_1(C) = \#_2(C)$, otherwise it is *unbalanced* (these definitions are closely related to the notions of balanced/unbalanced oriented cycles in digraphs, cf. [19]). Note that all balanced cycles are also heterogeneous.

Recall that the trim automaton is the result of removing all states which are not reachable from the initial state, and all states from which no final state is reachable.

▶ **Theorem 2.** *For any trim NFA A over the alphabet* **2***, and its transition graph G_A,*

(I) *$\mathit{shiftlag}(L(A)) = \infty$ iff*
- *G_A contains a heterogeneous unbalanced cycle, or*
- *G_A contains a path from a homogeneous to a heterogeneous cycle,*

(II) *$\mathit{shift}(L(A)) = \infty$ iff G_A has a heterogeneous cycle,*

(III) *$\mathit{lag}(L(A)) = \infty$ iff G_A has an unbalanced cycle.*

Proof idea. The 'if' directions of all items are straightforward. For the 'only if' direction of item (1), it can be shown that for $n = 2|Q| + 1$ (where $|Q|$ is the number of states of A), any accepting run of A on $w \in L(A)$ so that $\mathit{shiftlag}(w) \geq n$ must induce a path on the transition graph G_A of A containing either a heterogeneous unbalanced cycle, or a homogeneous cycle followed by a heterogeneous cycle. Once this is verified, the statement follows. Note that since $\mathit{shiftlag}(w) \geq n$, w must contain n consecutive $>n$-lagged shifts $1 \leq a_1 < a_2 < \cdots < a_n \leq |w|$ in w. Since a_1 is $>n$-lagged, there must be an unbalanced cycle C_1 contained in the path induced by the run ρ restricted to $w[1, a_1]$. Since there is a sufficiently large number of shifts, there must be some heterogeneous cycle C_2 contained in the path induced by the run ρ restricted to $w[a_1, |w|]$. Of course, we have that there is a path from C_1 to C_2 in G_A, showing (1). ◀

▶ **Corollary 3.** *Checking whether* $\textsc{Rel}(L(A)) \subseteq \mathsf{REG}$, $\textsc{Rel}(L(A)) \subseteq \mathsf{REC}$ *or* $\textsc{Rel}(L(A)) \subseteq \mathsf{REG}_2^{bld}$ *can be done in polynomial time in the size of A.*

Note that Corollary 3 does *not* mean that it is decidable whether a relation $R \in \mathsf{RAT}$ is in REG (in fact, this problem is undecidable [8, Theorem 8.4-(vi)]). What one can check is whether it has a "safe" control, in the sense that it synchronizes regular relations. Hence, for any relation R controlled by $L(A)$, if $\textsc{Rel}(L(A)) \subseteq \mathsf{REG}$ then $R \in \mathsf{REG}$, but the opposite does not necessarily hold. For example, if we take $L' = (1|2)^*$, we have that $\textsc{Rel}(L') \not\subseteq \mathsf{REG}$ but the universal relation $\mathbb{A}^* \times \mathbb{A}^*$ is obviously in REG.

5 Resynchronizing relations

We saw that different languages in $\mathbf{2}^*$ can generate the same class relations, and yet for the commonly used classes, we have synchronization languages that somehow look canonical:

for instance, $(12)^*(1^*|2^*)$ for REG. Thus, we now address the question whether we can resynchronize relations using those canonical synchronization languages, and if so, can we do it effectively?

To pose this formally, suppose two different languages $S, S' \subseteq (\mathbf{2} \times \mathbb{A})^*$ controlled by $L, L' \subseteq \mathbf{2}^*$ respectively represent the same relation, i.e., $[\![S]\!] = [\![S']\!]$. Then we say that S is an *L-resynchronization* of S'. Given a class $\mathcal{C}$ of regular languages over $\mathbf{2}$, we say that $L_0 \in \mathcal{C}$ is a *canonical representative* of $\mathcal{C}$ if for every $L \in \mathcal{C}$ and every L-controlled language S there exists an L_0-resynchronization of S. In other words, for every $L \in \mathcal{C}$ and $R \in \textsc{Rel}(L)$, there is an L_0-controlled $S' \in (\mathbf{2} \times \mathbb{A})^*$ so that $[\![S']\!] = R$. If, in addition, there is a recursive procedure that constructs such an L_0-resynchronization of S, then we say that L_0 is an *effective canonical representative* of $\mathcal{C}$.

Let RL_{all} be the class of all regular languages over $\mathbf{2}$, and let $\mathsf{RL}^{fin}_{param}$ stand for class of regular languages $L \subseteq \mathbf{2}^*$ with finite parameter *param*, where *param* is lag, or shift, or shiftlag. We also let $\mathsf{RL}_{lag \leq \delta}$ denote the class of all regular languages $L \subseteq \mathbf{2}^*$ with $lag(L) \leq \delta$.

▶ **Example 4.** *Take, for example, $L_1 = (1122)^*1^*2^*$ and $L_2 = (12)^*(1^*|2^*)$, and a L_1-controlled relation S_1. Since $shiftlag(L_1) < \infty$, $[\![S_1]\!] \in \mathsf{REG}$ by Theorem 1. Further, since by Proposition 1-(2) $\textsc{Rel}(L_2) = \mathsf{REG}$, there must be some L_2-controlled relation S_2 so that $[\![S_2]\!] = [\![S_1]\!]$. In other words S_2 is the L_2-resynchronization of S_1. Since $\textsc{Rel}(L_2) = \mathsf{REG}$ in fact L_2 is a canonical representative of $\mathsf{RL}^{fin}_{shiftlag}$.*

▶ **Theorem 5** (Resynchronization theorem)**.**

(I) *$(12)^*(1^*|2^*)$ is an effective canonical representative of $\mathsf{RL}^{fin}_{shiftlag}$;*

(II) *1^*2^* is an effective canonical representative of $\mathsf{RL}^{fin}_{shift}$;*

(III) *there is no canonical representative of RL^{fin}_{lag};*

(IV) *$(12)^*(1^{\leq\delta}|2^{\leq\delta})$ is an effective canonical representative of $\mathsf{RL}_{lag\leq\delta}$;*

(V) *$\mathbf{2}^*$ is an effective canonical representative of RL_{all}.*

If the relations are given as NFA, the synchronization procedures are in exponential time.

Proof idea. We only give the proof sketch for (1), the other items being easier.

The *strongly connected components* (henceforth SCC) of G_A are its maximal strongly connected subgraphs. An SCC is *heterogeneous* if it contains a heterogeneous cycle; an SCC is *homogeneous* if it contains a cycle and all the cycles it contains are homogeneous; otherwise, an SCC without cycles (that is, a single vertex) is an *edgeless* SCC. The *condensation* of G_A (written $con(G_A)$) is the labeled directed acyclic graph (henceforth labeled DAG) induced by the SCC's of G_A. This is the labeled DAG whose nodes are the SCC's of G_A, and there is an edge labeled $(q, (i, a), q')$ from vertex v to vertex v' iff $v \neq v'$, q belongs to the SCC v in G_A, q' belongs to the SCC v' in G_A, and there is an edge labeled $(q, (i, a), q')$ from q to q' in G_A (in other words, $(q, (i, a), q')$ is a transition of A).

Let $S \subseteq (\mathbf{2} \times \mathbb{A})^*$ be an L-controlled regular language with $shiftlag(L) < \infty$. Let A be an NFA recognizing S with statespace Q, initial state q_0 and set of final states Q_F.

Note that since the projection of S onto $\mathbf{2}$ is inside L, we can apply Theorem 2-(1) to A, obtaining that there are no paths from homogeneous SCC's to heterogeneous SCC's in G_A (and there are no heterogeneous cycles C with $\#_1(C) \neq \#_2(C)$). Let Q_{hom} be the set of all vertices of G_A that are reachable from a vertex of a homogeneous SCC. Note that Q_{hom} includes all vertices in homogeneous SCC's, plus some vertices from edgeless SCC's. Also, note that the subgraph of G_A induced by Q_{hom} has no heterogeneous cycles. Let $Q_{het} = Q \setminus Q_{hom}$. Hence, Q_{het} includes all vertices in heterogeneous SCC's and some vertices in edgeless SCC's. Also, by the property before, the subgraph of G_A induced by Q_{het} is

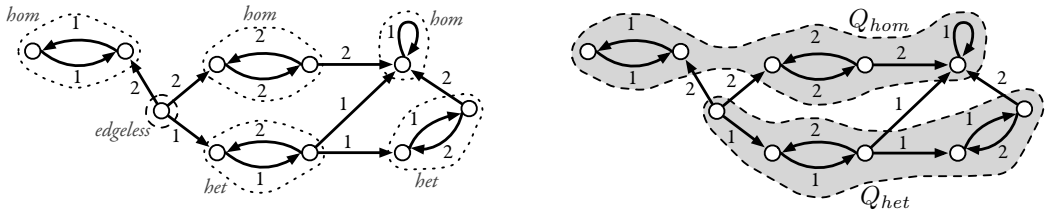


Figure 1 Example of G_A with the subgraphs induced by Q_{hom} and Q_{het}. For simplicity we assume that $\mathbb{A} = \{a\}$ and we hence omit the letter a when depicting edges labeled by (i, a).

connected. Figure 1 contains an example. Further, any path P in G_A is of the form (1) $P \cdot (q, \tau, q') \cdot P'$, (2) P, or (3) P', where

- P is a (possibly empty) path of the subgraph of G_A induced by Q_{het},
- P' is a (possibly empty) path of the subgraph of G_A induced by Q_{hom},
- $q \in Q_{het}$, $q' \in Q_{hom}$, and τ is a transition of A.

Let A^{het} be A restricted to Q_{het}, and let A^{hom} be A restricted to Q_{hom}. For every pair of states $q_{het} \in Q_{het}$ and $q_{hom} \in Q_{hom}$, let $L_{q_{het},q_{hom}}$ be the union of all

$$L(A^{het}[q_0, q_{het}]) \cdot \{(i, a)\} \cdot L(A^{hom}[q_{hom}, q_f])$$

for every $q_f \in Q_F$ and $(i, a) \in \mathbf{2} \times \mathbb{A}$ so that $(q_{het}, (i, a), q_{hom})$ is a transition of A. Let $L_{hom} = \bigcup_{q_f \in Q_F} L(A^{hom}[q_0, q_f])$ and $L_{het} = \bigcup_{q_f \in Q_F} L(A^{het}[q_0, q_f])$. It follows that

$$S = L_{hom} \cup L_{het} \cup \bigcup_{q_{het} \in Q_{het}, q_{hom} \in Q_{hom}} L_{q_{het},q_{hom}} .$$

We show that we can build, in exponential time, a $(12)^*(1^*|2^*)$-controlled automaton for each of these languages. Since the case of $L_{q_{het},q_{hom}}$ is more general than L_{hom} and L_{het}, we will only prove this case.

Note that by definition of A^{het} and A^{hom}, and since G_A has no unbalanced heterogeneous cycles, for every $q_{het} \in Q_{het}, q_{hom} \in Q_{hom}, q_f \in Q_F$ we have that $lag(L(A^{het}[q_0, q_{het}])) < \infty$ and $shift(L(A^{hom}[q_{hom}, q_f])) < \infty$. This implies that $lag(L(A^{het}[q_0, q_{het}])) \leq n$, and $shift(L(A^{hom}[q_{hom}, q_f])) \leq n$, for $n = |A|$.

By the already shown item (2), there exists a (1^*2^*)-controlled automaton $A^{hom}_{q_{hom},q_f}$ so that $[\![L(A^{hom}[q_0, q_{hom}])]\!] = [\![L(A^{hom}_{q_0,q_{hom}})]\!]$. By item (4), there exists a $(12)^*(1^{\leq n}|2^{\leq n})$-controlled automaton $A^{het}_{q_0,q_{het}}$ so that $[\![L(A^{het}[q_0, q_{het}])]\!] = [\![L(A^{het}_{q_0,q_{het}})]\!]$. These automata can be built in exponential time.

Indeed, a $(12)^*(1^*|2^*)$-controlled automaton for $L_{q_{het},q_{hom}}$ can be built from $A^{het}_{q_0,q_{het}}$ and all the $A^{hom}_{q_{hom},q_f}$'s for all $q_f \in Q_F$ in polynomial time, and thus the statement follows. This is shown by a variant of (2), showing that from any (1^*2^*)-controlled automaton one can build, in polynomial time, an equivalent automaton (in the sense of the relation it represents) that is $(12)^*(1^*|2^*)$-controlled. ◀

6 Closure via Parikh images

It is well known that the class REG is effectively closed under Boolean operations. Although RAT is a natural generalization of REG, it is not a Boolean algebra (let alone an effective one), not being closed under intersection or complement [8]. Even testing whether a rational relation is regular, or whether it has an empty intersection with a regular relation is undecidable [8].

Since regular relations are characterized via finite shiftlag, it is natural to ask whether infinite shiftlag somehow describes "dangerous" classes of relations. That is, does this mean for example that for any $L \subseteq \mathbf{2}^*$ with $\mathit{shiftlag}(L) = \infty$ the intersection problem is undecidable for $\textsc{Rel}(L)$? The answer to this question is negative: take for instance $L = (122)^*$ with $\mathit{shiftlag}(L) = \infty$. However, it is not hard to see that $\textsc{Rel}(L)$ is effectively closed under intersection.

This raises the question of whether there are classes $\mathcal{C} \subseteq \mathsf{RAT}$ that are natural, expressive, and well-behaved, that is, so that

- $\mathsf{REC} \subsetneq \mathcal{C}$,
- $\mathcal{C}$ is effectively closed under union, intersection and complementation (i.e., is an *effective Boolean algebra*); and
- $\mathcal{C}$ corresponds to a natural condition on the language.

Note that REG is one such example. Here we address the question from our perspective in terms of control languages. The idea is to show sufficient conditions of synchronization languages L so that $\textsc{Rel}(L)$ is effectively closed under intersection, or an effective boolean algebra. We state those in terms of Parikh images of languages.

Recall that the *Parikh image* of a word $w \in \mathbf{k}^*$, written $\Pi(w)$, is the vector of $\mathbb{N}_0^k$ whose ith component contains $\#_i(w)$, the number of occurrences of i in w. The Parikh image of a language L is $\Pi(L) = \{\Pi(w) \mid w \in L\}$. It is well known that for regular and context-free languages L, sets $\Pi(L)$ are exactly the semi-linear sets in $\mathbb{N}_0^k$, see [26].

A language $L \subseteq \mathbf{k}^*$ is

- *Parikh-injective* if the function $\Pi : L \to \mathbb{N}_0^k$ is injective, and
- *Parikh-surjective* if the function $\Pi : L \to \mathbb{N}_0^k$ is surjective.

▶ **Example 6.**

- $(12)^*(1^*|2^*)$ *and* 1^*2^* *are Parikh-injective, while* $(1|2)^*$ *is not.*
- *It can easily be shown that* $L = w_1^* \cdot w_2^* \cdots w_\ell^* \subseteq \mathbf{k}^*$ *is Parikh-injective if* $\ell \leq k$ *and* $\{\Pi(w_1), \ldots, \Pi(w_\ell)\}$ *generate a linear subspace of* $(\mathbb{N}_0)^k$ *of dimension* ℓ*. For example,* $(122)^*(112)^*$ *is Parikh-injective.*
- $(12)^*(1^*|2^*)$*,* 1^*2^**, and* $(1|2)^*$ *are Parikh-surjective, but* $(122)^*(112)^*$ *is not Parikh-surjective.*
- *It is easy to see that* $L_{r/s}$ *as defined in* (2) *is Parikh-injective and Parikh-surjective for any choice of* r, s*. For example, if* $r = 2$*,* $s = 1$*, we have* $L_{r/s} = (122)^*(22^*|1^*2|1^*)$*, which is Parikh-injective and Parikh-surjective, since every element of* $(\mathbb{N}_0)^2$ *is covered, and there is only one way to reach any element of* $(\mathbb{N}_0)^2$*.*

We now analyze the (effective) closure of classes $\textsc{Rel}(L)$ under Boolean operations. It turns out that closure under union is free, but for closure under intersection and complement, the newly introduced criteria serve as sufficient conditions.

▶ **Theorem 7.** *Let* $L \subseteq \mathbf{2}^*$ *be a regular language. Then*

(I) $\textsc{Rel}(L)$ *is effectively closed under union, alphabetic morphisms, and inverse alphabetic morphisms;*

(II) *If* L *is Parikh-injective, then* $\textsc{Rel}(L)$ *is effectively closed under intersection;*

(III) *if* L *is both Parikh-injective and Parikh-surjective, then* $\textsc{Rel}(L)$ *is effectively closed under complement.*

Proof idea. We prove only item (3). Let $S \subseteq (\mathbf{2} \times \mathbb{A})^*$ be an L-controlled relation. We show that $[\![S]\!]^c = [\![S^c \cap (L \otimes \mathbb{A}^*)]\!]$, where S^c, $[\![S]\!]^c$ denote the complement of S, $[\![S]\!]$ respectively, and $L \otimes \mathbb{A}^*$ denotes the set of all words $u \otimes v$ where $|u| = |v|$, $u \in L$ and $v \in \mathbb{A}^*$.

[$\subseteq$] Suppose $(u, v) \notin [\![S]\!]$. We show that there must be some $w \in S^c \cap (L \otimes \mathbb{A}^*)$ so that $(u, v) = [\![w]\!]$. By Parikh surjectivity and injectivity, there is exactly one word $w' \in L$ so that $\Pi(w') = (|u|, |v|)$. Let $w = u' \otimes v' \in (\mathbf{2} \times \mathbb{A})^*$ be the only word so that $u' = w'$ and $[\![w]\!] = (u, v)$. Note that $w \notin S$ and that its projection onto the first component (*i.e.*, w') is in L. Therefore, $w \in S^c \cap (L \otimes \mathbb{A}^*)$.

[$\supseteq$] Assume $w \in S^c \cap (L \otimes \mathbb{A}^*)$ and suppose that $[\![w]\!] \in [\![S]\!]$. Then, there is some $w' \in S$ so that $[\![w']\!] = [\![w]\!]$. It cannot be that $w' = w$, as it would be in contradiction with $w \in S^c \cap (L \otimes \mathbb{A}^*)$. Since L is Parikh-injective, and w, w' are L-controlled, $w = w'$, as otherwise $[\![w']\!] \neq [\![w]\!]$. This contradicts $w \in S^c \cap (L \otimes \mathbb{A}^*)$. Thus, $[\![w]\!] \notin [\![S]\!]$ and $[\![S]\!]^c \supseteq [\![S^c \cap (L \otimes \mathbb{A}^*)]\!]$. ◀

▶ **Corollary 8.** *If $L \subseteq \mathbf{2}^*$ is Parikh-injective and Parikh-surjective, then* $\textsc{Rel}(L)$ *is an effective boolean algebra, closed under alphabetic morphisms and inverse alphabetic morphisms.*

Observe that in this context, REG and REC are simply two examples of the (infinitely) many such well-behaved classes.

▶ **Example 9.**

- REC *and* REG *are effective boolean algebras because they correspond to* $\textsc{Rel}(1^*2^*)$ *and* $\textsc{Rel}((12)^*(1^*|2^*))$*, where* 1^*2^**,* $(12)^*(1^*|2^*)$ *are Parikh-injective and Parikh-surjective.*
- $\textsc{Rel}((122)^*(112)^*)$ *is effectively closed under intersection.*
- *It was shown in [11] that the class of (r/s)-synchronized relations is an effective Boolean algebra. Our results provide an alternative proof, since $L_{r/s}$ is Parikh-injective and Parikh-surjective.*

Observation. Note that Theorem 7 cannot be generalized to finite unions of Parikh-injective languages, since for example $\textsc{Rel}(L)$ for $L = ((12)^*1^*)|(1^*(12)^*)$ is not closed under intersection. In fact, its intersection problem is undecidable. This follows from the fact that $\textsc{Rel}(L)$ contains the suffix relation and all regular relations (where the first component is longer than the second). By [5, Theorem V.1], this problem is undecidable.

7 Future work

We presented a new way of looking at relations on words, and this new perspective opens up several directions. An obvious one is to extend results to k-ary relations, for $k > 2$. We know that exact analogs of Proposition 1, Theorem 1, and Theorem 2 continue to hold.

Another natural extension is to look for other classes of relations, say analogs of context-free languages. In particular, one can look at a generalization of rational relations, the *pushdown relations* of [14], which are those recognized by multi-tape automata with a stack or, equivalently, by a context-free grammar. We have some preliminary results in this direction but more work is needed.

We also would like to use the structural approach to look for better behaved classes of relational word transducers for verification purposes, and for classes of relations that can be effectively used in querying graph data. Finally, we would like to use it to identify classes of well behaved relations over *data words* [9] and study logics over them, extending the approach of [5, 6] with data.

Acknowledgment. Work partially supported by EPSRC grants G049165 and J015377.

References

1 P.A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *CONCUR'03*, pages 35–48.
2 R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC'04*, pages 202–211.
3 R. Angles and C. Gutiérrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), 2008.
4 K. Anyanwu and A.P. Sheth. ρ-queries: enabling querying for semantic associations on the semantic web. In *WWW'03*, pages 690–699.
5 P. Barceló, D. Figueira, and L. Libkin. Graph logics with rational relations. *LMCS*, 9(3:1), 2013.
6 P. Barceló, L. Libkin, A. W. Lin, and P. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst.*, 37(4):31, 2012.
7 M. Ben-Ari. *Principles of the Spin model checker*. Springer, 2008.
8 J. Berstel. *Transductions and Context-Free Languages*. B. G. Teubner, 1979.
9 M. Bojańczyk. Automata for data words and data trees. In *RTA'10*, pages 1–4.
10 A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV'00*, pages 403–418.
11 O. Carton. The growth ratio of synchronous rational relations is unique. *TCS*, 376(1-2):52–59, 2007.
12 O. Carton, C. Choffrut, and S. Grigorieff. Decision problems among the main subfamilies of rational relations. *RAIRO Theor. Inf. and Appl.*, 40(2):255–275, 2006.
13 C. Choffrut. Relations over words and logic: A chronology. *Bull. of the EATCS*, 89:159–163, 2006.
14 C. Choffrut and K. Culik II. Properties of finite and pushdown transducers. *SIAM J. Comput.*, 12(2):300–315, 1983.
15 C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9(1):47–68, January 1965.
16 R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. A formal framework for information extraction. In *PODS'13*, pages 37–48.
17 C. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. *TCS*, 108(1):45–82, 1993.
18 T. Harju, A. Mateescu, A. Salomaa. shuffle on trajectories: the Schützenberger product and related operations. *MFCS'98*, pages 503–511.
19 P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.
20 B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *TACAS'00*, pages 220–234.
21 J. Leguy. Transductions rationnelles décroissantes. *ITA*, 15(2):141–148, 1981.
22 K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
23 R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.
24 F. Neven. Automata, Logic, and XML. In *CSL'02*, pages 2–26.
25 M. Nivat. Transduction des langages de Chomsky. *Ann. Inst. Fourier*, 18:339–455, 1968.
26 R. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.
27 J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
28 T. Schwentick. Automata for XML – a survey. *JCSS*, 73(3):289–315, 2007.
29 A. W. To and L. Libkin. Algorithmic metatheorems for decidable LTL model checking over infinite systems. In *FOSSACS'10*, pages 221–236.

On Boolean closed full trios and rational Kripke frames

Markus Lohrey[1] **and Georg Zetzsche**[2]

1 **Department für Elektrotechnik und Informatik, Universität Siegen, Germany**
lohrey@eti.uni-siegen.de
2 **Fachbereich Informatik, Technische Universität Kaiserslautern, Germany**
zetzsche@cs.uni-kl.de

Abstract

A Boolean closed full trio is a class of languages that is closed under the Boolean operations (union, intersection, and complementation) and rational transductions. It is well-known that the regular languages constitute such a Boolean closed full trio. It is shown here that every such language class that contains any non-regular language already includes the whole arithmetical hierarchy (and even the one relative to this language).

A consequence of this result is that aside from the regular languages, no full trio generated by one language is closed under complementation.

Our construction also shows that there is a fixed rational Kripke frame such that assigning an arbitrary non-regular language to some variable allows the definition of any language from the arithmetical hierarchy in the corresponding Kripke structure using multimodal logic.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases rational transductions, full trios, arithmetical hierarchy, Boolean operations

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.530

1 Introduction

The study of closure properties of language classes has a long tradition, it can be traced back to the introduction of regular languages [10]. Among other applications, they provide insights about whether languages belong to certain classes and, as far as they are effective, allow the computation of representations of languages. They also often serve as a way to describe language classes without reference to concrete generating or accepting devices: In many cases, a language class can be described as the smallest class of languages that possesses a given collection of closure properties and contains certain generating languages.

Here, we are concerned with Boolean closed full trios, i.e., classes closed under the Boolean operations (union, intersection, and complementation) and rational transductions. It is well-known that the class of regular languages constitutes a Boolean closed full trio.

This combination of closure properties is interesting for several reasons. First, in the case of regular languages, this particular collection is exploited, for example, in the theory of automatic structures [9], since it implies that in such structures, every first-order definable relation can be represented by a regular language. Since emptiness is decidable for regular languages, one can therefore decide the first-order theory of these structures.

Second, the languages definable by multimodal logic in a rational Kripke frame, i.e., a Kripke frame in which the worlds are words and the visibility relations are given by rational transductions, are always confined to the Boolean closed full trio generated by the values



© Markus Lohrey and Georg Zetzsche;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 530–541



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



(that is, languages) assigned to the variables. This was observed by Bekker and Goranko [2] and then used to show that the model checking problem for multimodal logic and rational Kripke frames is decidable if all variables are assigned regular languages.

Third, a wide range of interesting language classes are principal full trios, i.e., full trios that are generated by one language. Since these are always union closed, their closure under complementation is equivalent to the class being a Boolean closed full trio. Examples of principal full trios are the context-free languages, languages accepted by multicounter automata (for a bounded number of counters and blind, partially blind, or with zero test [7]), and the languages accepted by valence automata over a finitely generated monoid [6].

Hence, the question arises whether there are language classes beyond the regular languages that enjoy these closure properties and still admit decision procedures for simple properties such as emptiness. Our first main result (Theorem 9) states that every Boolean closed full trio that contains *any non-regular language* already includes the whole arithmetical hierarchy (and even the arithmetical hierarchy relative to this language) and thus loses virtually all decidability properties. This is a remarkable fact, because it means that these closure properties are so extremely powerful that even the simplest non-regular languages allow the construction of a very large class of languages.

A large number of grammar and automata models is easily seen to exceed the regular languages but stay within the recursively enumerable languages. Hence, Theorem 9 also implies that the corresponding language classes are never Boolean closed full trios. We can also conclude that *other than the regular languages, no principal full trio* is closed under complementation.

It should be noted that Theorem 9 does not mean that there is no way of developing a theory of automatic structures beyond regular languages. It might well be that some smaller collection of closure properties suffices to obtain all first-order definable relations and still admits a decision procedure for the emptiness problem.

Actually, it turns out that three fixed rational transductions, together with the Boolean operations, suffice to construct all arithmetical languages from any non-regular language. Therefore, our second main result (Theorem 14) states that there is a fixed rational Kripke frame with three modalities such that assigning any non-regular language to a variable allows the definition of every arithmetical language using multimodal logic.

Other results of a similar spirit on closure properties of language classes have been known for a long time. For example, Hartmanis and Hopcroft [8] have proved that every intersection closed full AFL containing $\{a^n b^n \mid n \in \mathbb{N}\}$ includes the recursively enumerable languages. Here, a full AFL is a full trio that is closed under union and the Kleene star. Furthermore, Book [4] has shown that the arithmetical languages constitute the smallest Boolean closed full trio that is closed under homomorphic replication, the latter of which is a generalization of homomorphisms. Hence, our result means in Book's result one can replace the homomorphic replication by containment of any non-regular language. However, to the best of the authors' knowledge, to date there is no known combination of natural closure properties that are enjoyed by the regular languages but that yield all the recursively enumerable languages (let alone the arithmetical hierarchy) when applied to any non-regular language.

2 Preliminaries

Let Σ be a fixed countable set of abstract symbols, the finite subsets of which are called *alphabets*. Given an alphabet X, the set of words over X is denoted by X^* and the empty word by λ. Subsets of X^* for alphabets X are called *languages*. For a language L, the

smallest alphabet X with $L \subseteq X^*$ is denoted by $\alpha(L)$. The *complement* of L is defined as $\overline{L} = \alpha(L)^* \setminus L$. A *transduction* is a subset of $X^* \times Y^*$ for alphabets X, Y.

Let M be a monoid with neutral element 1. An *automaton over* M is a tuple $A = (Q, M, E, q_0, F)$, in which Q is a finite set of *states*, E is a finite subset of $Q \times M \times Q$ called the set of *edges*, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *final states*. The *step relation* $\Rightarrow_A$ of A is a binary relation on $Q \times M$, for which $(p, a) \Rightarrow_A (q, b)$ if and only if there is an edge (p, c, q) such that $b = ac$. The set generated by A is then

$$S(A) = \{a \in M \mid \exists q \in F : (q_0, 1) \Rightarrow_A^* (q, a)\}.$$

A set $R \subseteq M$ is called *rational* if it can be written as $R = S(A)$ for some automaton A over M. A rational language is also called *regular*. We use REG to denote the class of regular languages.

A *valence automaton over* M is an automaton A over the monoid $X^* \times M$, where X is an alphabet. The *language accepted by* A is defined as $L(A) = \{w \in X^* \mid (w, 1) \in S(A)\}$. The class of languages accepted by valence automata over M is denoted by $\mathsf{VA}(M)$.

Given alphabets X and Y, a *rational transduction* is a rational subset of the monoid $X^* \times Y^*$. For a language $L \subseteq Y^*$ and a rational transduction R, we write RL for $\{x \in X^* \mid \exists y \in L : (x, y) \in R\}$.

A *language class* is a set of languages that contains at least one non-empty language. A language class $\mathcal{C}$ is called a *full trio* (or *cone*) if it is closed under (arbitrary) homomorphisms, inverse homomorphisms, and intersection with regular languages. It is well-known [3] that a class $\mathcal{C}$ is a full trio if and only if it is closed under rational transductions, i.e., for every $L \in \mathcal{C}$ and every rational transduction R, we have $RL \in \mathcal{C}$. We call a language class *Boolean closed* if it is closed under all Boolean operations (union, intersection, and complementation). By the *full trio generated by* the language L we mean the smallest full trio that contains L. A full trio is called a *principal full trio* if it is generated by some language.

For any language class $\mathcal{C}$, we write $\mathsf{RE}(\mathcal{C})$ for the class of languages accepted by some Turing machine with an oracle $L \in \mathcal{C}$. Similarly, let $\mathsf{REC}(\mathcal{C})$ be the class of languages accepted by some Turing machine that halts on every input and has access to an oracle $L \in \mathcal{C}$. Furthermore, let REC denote the class of recursive languages. We also write $\mathsf{REC}(L)$ and $\mathsf{RE}(L)$ for $\mathsf{REC}(\{L\})$ and $\mathsf{RE}(\{L\})$, respectively. Then the *arithmetical hierarchy* (see, for example, [11]) is defined as

$$\Sigma_0 = \mathsf{REC}, \qquad \Sigma_{n+1} = \mathsf{RE}(\Sigma_n) \text{ for } n \geq 0, \qquad \mathsf{AH} = \bigcup_{n \geq 0} \Sigma_n.$$

Languages in AH are called *arithmetical*. The *arithmetical hierarchy relative to* L is defined as

$$\Sigma_0(L) = \mathsf{REC}(L), \qquad \Sigma_{n+1}(L) = \mathsf{RE}(\Sigma_n(L)) \text{ for } n \geq 0, \qquad \mathsf{AH}(L) = \bigcup_{n \geq 0} \Sigma_n(L).$$

We will often encode words from an alphabet X, $|X| \geq 2$, by words in $\{0,1\}^*$. If $X = \{a_1, \ldots, a_n\}$, then a homomorphism $g\colon X^* \to \{0,1\}^*$ with $g(a_i) = 10^i$ will be called a *standard encoding*. For each subset $Y \subseteq X$, the homomorphism $\pi_Y : X^* \to Y^*$ is defined by $\pi_Y(x) = x$ for $x \in Y$ and $\pi_Y(x) = \lambda$ for $x \in X \setminus Y$.

Let X be an alphabet. For languages $L \subseteq X^*$ and words $u, v \in X^*$, we write $u \equiv_L v$ if for each $w \in X^*$, we have $uw \in L$ if and only if $vw \in L$. The equivalence relation $\equiv_L$ is called the *Myhill-Nerode equivalence*. The well-known Myhill-Nerode Theorem states that L is regular if and only if $\equiv_L$ has a finite index.

▶ Remark. In the following, we will make statements about certain languages in $\{0,1\}^*$ being obtainable from other languages in $\{0,1\}^*$ either by using a finite set of transductions or by using a finite set of transductions and Boolean operations. It will then always be possible to use larger alphabets with auxiliary symbols for the following reason. Suppose there is a finite set S of transductions, each over the alphabet $X = \{a_1, \ldots, a_n\}$, where $\{0,1\} \subseteq X$. Let $h : X^* \to \{0,1\}^*$ be a standard encoding. Then we have

$$h(L \cap K) = h(L) \cap h(K), \qquad h(L \cup K) = h(L) \cup h(K),$$
$$h(\overline{L}) = \overline{h(L)} \cap h(\alpha(L))^*, \qquad h(RL) = hRh^{-1}(h(L)).$$

By induction, it follows that for every language $K \subseteq X^*$ that can be obtained from $L \subseteq X^*$ using transductions in S (and Boolean operations), we can obtain $h(K)$ from $h(L)$ by using transductions in

$$S' = \{\rho_Y \mid Y \subseteq X\} \cup \{hRh^{-1} \mid R \in S\}$$

(and Boolean operations), where ρ_Y is the rational transduction in $\{0,1\}^* \times \{0,1\}^*$ that maps M to $M \cap h(Y)^*$. In particular, if $K \subseteq \{0,1\}^*$ can be obtained from $L \subseteq \{0,1\}^*$ using transductions in S (and Boolean operations), we can obtain K from L by using transductions in $S'' = S' \cup \{h\,|_{\{0,1\}^*}, \rho_{\{0,1\}} \circ h^{-1}\}$ (and Boolean operations) by producing $h(L)$, then $h(K)$ using S' (and Boolean operations), and then $\rho_{\{0,1\}}(h^{-1}(h(K))) = K$. ◀

3 Boolean closed full trios

▶ **Lemma 1.** *Let $X = \{0,1\}$. There is a finite set F of rational transductions in $X^* \times X^*$ such that each regular language $K \subseteq X^*$ can be obtained from any non-empty $L \subseteq X^*$ using transductions in F.*

Proof. It suffices to prove the lemma for $K \subseteq X^*$ with $\lambda \notin K$: If $\lambda \in K$, $K \setminus \{\lambda\} \neq \emptyset$, we can use the rational transduction $\Lambda = \{(w, w) \mid w \in X^*\} \cup \{\lambda\} \times X^*$, which maps $w \in X^*$ to $\{w, \lambda\}$, to obtain K from $K \setminus \{\lambda\}$. If $K = \{\lambda\}$, we can use $\Lambda' = \{\lambda\} \times X^*$, which maps every $w \in \{0,1\}^*$ to λ, to obtain K directly from L. We may therefore assume that K is accepted by an automaton $A = (Q, X, E, q, Q_f)$, where $Q = \{0, \ldots, k\}$, $q = 0$, $Q_f = \{1\}$, and $E \subseteq Q \times X \times Q$.

Our goal is to produce the language T_A of all words $10^{i_0}1x_110^{i_1} \cdots x_n10^{i_n}$, such that $i_0 = 0$, $i_n = 1$, and $x_j \in \{0,1\}$ and $(i_j, x_{j+1}, i_{j+1}) \in E$ for $0 \leq j < n$. Then, clearly, the rational transduction P that outputs only the x_j will satisfy $PT_A = K$. By the above remark, it suffices to provide transductions over the extended alphabet $Y = \{0, 1, \#_1, \#_2\}$. The additional symbols $\#_1, \#_2$ are called *markers*.

First we use the initial transduction $I = 1(1\{0,1\}10^*)^*1\{0,1\}10 \times \{0,1\}^*$ to produce the set $1(1\{0,1\}10^*)^*1\{0,1\}10$ from L. In the following, a word $10^{i_0}1x_110^{i_1} \cdots x_n10^{i_n}$ is called an *encoding*. Its factors 0^{i_j} are called *state blocks* and its factors $0^{i_j}1x10^{i_{j+1}}$ are called *transition blocks*. The transduction I already guarantees that the leftmost and the rightmost state block correspond to the initial and the final state, respectively. We now wish to remove all words that contain a state block of length greater than k. In order to do this, we use the transduction S_1, which inserts the marker $\#_1$ in the beginning of every state block. Furthermore, we have the transduction M_1, which moves each occurrence of the marker one position to the right (i.e. outputs $0\#_1$ on input $\#_10$) if its right neighbor is a 0, and drops the occurrence otherwise. We also have the transduction R, which rejects all inputs that have a factor $\#_10$. All other words are accepted by R but stripped of their occurrences of

$\#_1$ in the output. Then applying $RM_1^kS_1$ yields the set of encodings with state blocks of length at most k.

In the next step, we wish to remove from the language all encodings that contain a transition block $10^\ell x10^m$ with $x \in \{0,1\}$, $0 \leq \ell, m \leq k$, and $(\ell, x, m) \notin E$. To this end, we have the transductions S_2 and M_2, which behave analogously to S_1 and M_1 by using $\#_2$ instead of $\#_1$. We assume that S_1 and S_2 are defined so as to add their marker and leave the other marker in place. We assume further that M_1 and M_2 move their marker so as to overtake the other marker if necessary. Finally, we have for each $x \in \{0,1\}$ the transition R_x, which rejects every word containing a transition block in which $\#_1$ is on the right end of the left state block, $\#_2$ is on the right end of the right state block, and the input letter is x. All other words are accepted by R_x but stripped of all occurrences of markers. Applying $R_xM_2^mS_2M_1^\ell S_1$ clearly yields the set of encodings that do not contain the transition block $10^\ell 1x10^m$. Therefore, we apply this sequence of transductions for each triple (ℓ, x, m) with $0 \leq \ell, m \leq k$, $x \in \{0,1\}$, and $(\ell, x, m) \notin E$. This clearly produces the language T_A and hence $K = PT_A$ is obtained. Since we only used transductions in $\{\Lambda, \Lambda', P, I, S_1, S_2, M_1, M_2, R, R_0, R_1\}$, the lemma is proven. ◀

▶ **Lemma 2.** *Let X be an alphabet with $|X| \geq 2$. For each finite set F of rational transductions in $X^* \times X^*$, there are rational transductions R, S, T in $X^* \times X^*$ such that every composition of transductions from F can be written in the form T^nS^mR with $m, n \in \mathbb{N}$.*

Proof. Let $0, 1 \in X$ be distinct letters and for $x \in \{0,1\}$, let A_x be the transduction that appends x to each input word, hence $A_x = \{(wx, w) \mid w \in X^*\}$. Furthermore, let $F = \{U_0, \ldots, U_{k-1}\}$, $b = k + 1$, and let U_i' be the rational transduction

$$U_i' = \{(u10^m, v10^{bm+i}) \mid (u, v) \in U_i, m \in \mathbb{N}\}, \quad U_k' = \{(w, w10^k) \mid w \in X^*\}$$

for each $0 \leq i < k$. We shall prove that $R = A_1$, $S = A_0$, and $T = \bigcup_{0 \leq i \leq k} U_i'$ have the desired property. Let $U_{i_n} \cdots U_{i_0}$ be a composition of elements of F and let $i_{n+1} = k$. We claim that

$$U_{i_n} \cdots U_{i_0} = T^{n+2}S^mR \quad \text{for } m = \sum_{j=0}^{n+1} i_j b^j.$$

Applying S^mR appends 10^m to each input word. Then, each application of T to a word $w10^\ell$ chooses some U_j', but this choice will only lead to a valid computation of the transducer if ℓ is congruent to j modulo b. Hence, applying T^{n+1} to $w10^m$ has the same effect as applying $U_{i_n}' \cdots U_{i_0}'$. Since the most significant digit in the b-ary representation of m is $i_{n+1} = k$, applying T once more means applying U_k' and hence removing the 10^k suffix of the input word. In the end, we applied $U_{i_n} \cdots U_{i_0}$. ◀

Lemmas 1 and 2 together immediately imply the following byproduct, which might be of independent interest.

▶ **Corollary 3.** *Let $X = \{0,1\}$. There are rational transductions R, S, T over X^* such that every regular language $K \subseteq X^*$ can be written as $T^nS^mRX^*$ for some $m, n \in \mathbb{N}$.*

We define the alphabet $\Delta = \{+, -, z\}$, whose elements will represent the operations *increment*, *decrement*, and *zero test*, respectively.

▶ **Definition 4.** Let $C \subseteq \Delta^*$ be the set of words $\delta_1 \cdots \delta_m$, $\delta_1, \ldots, \delta_m \in \Delta$ for which there are numbers $x_0, \ldots, x_m \in \mathbb{N}$ such that for $1 \leq i \leq m$:

1. if $\delta_i = +$, then $x_i = x_{i-1} + 1$,
2. if $\delta_i = -$, then $x_i = x_{i-1} - 1$, and
3. if $\delta_i = z$, then $x_i = x_{i-1} = 0$.

We shall prove that from L we can construct the following language $\hat{C}_L$ using a fixed finite set of rational transductions and Boolean operations.

▶ **Definition 5.** Suppose the alphabets X, Δ, and $\{\#\}$ are pairwise disjoint. Let $\hat{C}_L \subseteq (\Delta \cup X \cup \{\#\})^*$ be the set of all words

$$v_0\delta_1 v_1 \cdots \delta_m v_m \# u_0 \# \cdots u_n \#$$

with $\delta_i \in \Delta$, $v_i \in X^*$, $u_j \in X^*$, such that $u_k \not\equiv_L u_\ell$ for $k \neq \ell$ and for each $1 \leq i \leq m$ there is a $1 \leq j \leq n$ with

1. if $\delta_i = +$, then $v_{i-1} \equiv_L u_{j-1}$, $v_i \equiv_L u_j$,
2. if $\delta_i = -$, then $v_{i-1} \equiv_L u_j$, $v_i \equiv_L u_{j-1}$, and
3. if $\delta_i = z$, then $j = 1$ and $v_{i-1} \equiv_L v_i \equiv_L u_0 = u_{j-1}$.

▶ **Lemma 6.** *If L is not regular, then $\pi_\Delta(\hat{C}_L) = C$.*

Proof. In order to prove the inclusion "$\supseteq$", let $x_0, \ldots, x_m \in \mathbb{N}$ be numbers as in Definition 4 and suppose $\{x_0, \ldots, x_m\} \subseteq \{0, \ldots, n\}$. Since L is not regular, we can find words $u_0, \ldots, u_n \in X^*$ such that $u_k \not\equiv_L u_\ell$ for $k \neq \ell$. Now for each $0 \leq i \leq m$, let $v_i = u_{x_i}$. Then it can be checked straightforwardly that $v_0\delta_1 v_1 \cdots \delta_m v_m \# u_0 \# \cdots u_n \# \in \hat{C}_L$ and hence $\delta_1 \cdots \delta_m \in \pi_\Delta(\hat{C}_L)$.

For the inclusion "$\subseteq$", let $\delta_1 \cdots \delta_m \in \pi_\Delta(\hat{C}_L)$. Then there are words $v_0, \ldots, v_m \in X^*$, $u_0, \ldots, u_n \in X^*$ with $v_0\delta_1 v_1 \cdots \delta_m v_m \# u_0 \# \cdots u_n \# \in \hat{C}_L$. By the definition of $\hat{C}_L$, this means for each $1 \leq i \leq m$, there is a $1 \leq j \leq n$ such that 1–3 of Definition 5 hold. Hence, we can pick for each $1 \leq i \leq m$ an $x_i \in \{1, \ldots, n\}$ such that 1–3 of Definition 5 hold with $j = x_i$. Note that since this implies $v_{i-1} \equiv_L u_{j-1}$ for $\delta_i \in \{+, z\}$ and $v_{i-1} \equiv_L u_j$ for $\delta_i = -$ and the u_k are pairwise incongruent w.r.t. $\equiv_L$, this choice of x_i is unique. It can now be verified by induction on i that the conditions 1–3 of Definition 4 are satisfied. ◀

The following lemma is the central ingredient in our proof. The idea is to construct $\hat{C}_L$, which by Lemma 6 allows us to obtain C.

▶ **Lemma 7.** *Let $X = \{0, 1\}$. There is a finite set F of rational transductions such that for any non-regular $L \subseteq X^*$, the language C can be obtained from L using transductions in F and Boolean operations.*

Proof. We will use the alphabet $Y = X \cup \{\#\} \cup \Delta$. We prove the lemma by constructing C from L using a sequence of Boolean operations and transductions $T_1, \ldots, T_{19}$ over Y^* for which it will be clear that they do not depend on L.

There are clearly rational transductions T_1 and T_2 with

$$W_1 = \{u\#v\#w \mid u, v, w \in X^*,\ uw \in L\} = T_1 L,$$
$$W_2 = \{u\#v\#w \mid u, v, w \in X^*,\ vw \in L\} = T_2 L,$$

which means we can construct W_1 and W_2. Hence,

$$\begin{aligned} W' &= \{u\#v\#w \mid u, v, w \in X^*, (uw \in L, vw \notin L) \text{ or } (uw \notin L, vw \in L)\} \\ &= (W_1 \cap \overline{W_2}) \cup (\overline{W_1} \cap W_2) \end{aligned}$$

can also be constructed. We can clearly find a rational transduction T_3 with

$$W = \{u\#v \mid u, v \in X^*, u \not\equiv_L v\} = \{u\#v \mid u\#v\#w \in W' \text{ for some } w \in X^*\} = T_3W'.$$

This means $P = \{u\#v \mid u \equiv_L v\} = X^*\#X^* \setminus W = T_4\overline{W}$, for some T_4, can be constructed. With suitable rational transductions T_5, T_6, we have

$$\begin{aligned} S &= \{u_0\#u_1\#\cdots u_n\# \mid u_i \not\equiv_L u_j \text{ for all } i \neq j\} \\ &= (X^*\#)^* \setminus \{ru\#sv\#t \mid r, s, t \in (X^*\#)^*, u\#v \in P\} = T_6\overline{T_5P}, \end{aligned}$$

meaning that S can be constructed as well. Let M (*matching*) be the set of all words $v_1\delta v_2\#u_1\#u_2$ where $v_1, v_2, u_1, u_2 \in X^*$ with

- if $\delta = +$, then $v_1 \equiv_L u_1$ and $v_2 \equiv_L u_2$,
- if $\delta = -$, then $v_1 \equiv_L u_2$ and $v_2 \equiv_L u_1$, and
- if $\delta = z$, then $v_1 \equiv_L v_2 \equiv_L u_1$.

Since

$$\begin{aligned} M = {} & \{v_1{+}v_2\#u_1\#u_2 \mid v_1\#u_1 \in P,\ v_2\#u_2 \in P\} \\ & \cup \{v_1{-}v_2\#u_1\#u_2 \mid v_1\#u_2 \in P,\ v_2\#u_1 \in P\} \\ & \cup \{v_1zv_2\#u_1\#u_2 \mid v_1\#v_2 \in P,\ v_1\#u_1 \in P,\ u_2 \in X^*\} \\ = {} & (T_7P \cap T_8P) \cup (T_9P \cap T_{10}P) \cup (T_{11}P \cap T_{12}P) \end{aligned}$$

for suitable rational transductions $T_7, \ldots, T_{12}$, we can also construct M.

Let E (*error*) be the set of words $v_1\delta v_2\#u_0\#\cdots u_n\#$ such that for every $1 \leq j \leq n$, we have $v_1\delta v_2\#u_{j-1}\#u_j \notin M$ or we have $\delta = z$ and $v_1 \not\equiv_L u_0$. Since

$$E' = \{v_1\delta v_2\#ru_1\#u_2\#s \mid v_1\delta v_2\#u_1\#u_2 \in M, r, s \in (X^*\#)^*\} = T_{13}M$$

for some rational transduction T_{13}, we can construct E'. Furthermore, since

$$\begin{aligned} E &= \{v_1zv_2\#u_0r \mid v_1 \not\equiv_L u_0,\ r \in (X^*\#)^*, v_2 \in X^*\} \cup [(X^*\Delta X^*\#(X^*\#)^* \setminus E'] \\ &= T_{14}\overline{P} \cup T_{15}\overline{E'}, \end{aligned}$$

for some rational transductions T_{14}, T_{15}, we can construct E.

Let N (*no error*) be the set of words $v_0\delta_1v_1\cdots\delta_mv_m\#u_0\#\cdots u_n\#$ such that for every $1 \leq i \leq m$, there is a $1 \leq j \leq n$ with $v_{i-1}\delta_iv_i\#u_{j-1}\#u_j \in M$ and if $\delta_i = z$, then $v_{i-1} \equiv_L u_0$. Since

$$\begin{aligned} N' &= \{w \in (X^*\Delta)^*v_1\delta v_2(\Delta X^*)^*\#u_0\#\cdots u_n\# \mid v_1\delta v_2\#u_0\#\cdots u_n\# \in E\} = T_{16}E, \\ N &= (X^*\Delta)^+X^*\#(X^*\#)^* \setminus N' = T_{17}\overline{N'} \end{aligned}$$

for some rational transductions T_{16}, T_{17}, we can construct N.

Now we have $\hat{C}_L = N \cap (X^*\Delta)^*X^*\#S = N \cap T_{18}S$ for some rational transduction T_{18}, meaning we can construct $\hat{C}_L$. By Lemma 6, we have $C = T_{19}\hat{C}_L$ for some rational transduction T_{19}. This proves our claim and hence the lemma. ◀

▶ **Lemma 8.** *Let $X = \{0, 1\}$. There is a finite set F of rational transductions in $X^* \times X^*$ such that for any non-regular $L \subseteq X^*$, each $K \in \mathsf{RE}$, $K \subseteq X^*$, can be obtained from L using transductions in F and Boolean operations.*

Proof. Let F' contain the set of rational transductions provided by Lemma 1 and the one provided by Lemma 7. We will use the alphabet $Y = X \cup \Delta \cup \{\#\}$ and a standard encoding $g : Y^* \to X^*$.

Suppose $K \subseteq X^*$ is recursively enumerable and let $A = (Q, X, E, q_0, Q_f)$ be a 2-counter machine, $E \subseteq Q \times X^* \times \Delta \times \Delta \times Q$, accepting K and with $Q = \{0, \ldots, k\}$ and $Q_f = \{k\}$. Here, we assume that the machine operates on both counters in each step. Let R be the regular language of all words $0^{m_0} \prod_{i=1}^{n} \#w_i\#\delta_i^{(0)}\delta_i^{(1)}0^{m_i}$ with $(m_{i-1}, w_i, \delta_i^{(0)}, \delta_i^{(1)}, m_i) \in E$ for every $1 \leq i \leq n$, $m_0 = 0$, and $m_n = k$. We can obtain $g(R)$ from L using only transductions in F'. Thus, we can obtain $R = g^{-1}(g(R))$. Clearly, there are rational transductions T_1 and T_2 such that

$$U = \left\{ 10^0 \prod_{i=1}^{n} \#w_i\#\delta_i^{(0)}\delta_i^{(1)}10^{m_i} \in R \;\middle|\; \delta_1^{(k)} \cdots \delta_n^{(k)} \in C \text{ for } k = 0, 1 \right\} = R \cap T_1 C \cap T_2 C,$$

meaning that we can also obtain U. Finally, applying to U the transduction T_3 that outputs all occurrences of X after odd occurrences of $\#$ up to the next occurrence of $\#$ clearly yields K. If we let F consist of F' and g^{-1}, T_1, T_2, T_3, the lemma is proven. ◀

▶ **Theorem 9.** *Let $X = \{0, 1\}$. There are rational transductions R, S, T over X^* such that for any non-regular $L \subseteq X^*$, each $K \in \mathsf{AH}(L)$, $K \subseteq X^*$, can be obtained from L using R, S, T and Boolean operations.*

Proof. We shall prove that there is a finite set F of rational transductions in $X^* \times X^*$ such that for any $K \subseteq X^*$, we can obtain each $M \in \mathsf{RE}(K)$, $M \subseteq X^*$, from K and L using transductions in F and Boolean operations. This clearly implies that we can obtain all of $\Sigma_1(L) = \mathsf{RE}(L)$ from L and hence, by induction on i, all of $\Sigma_i(L)$ from L. According to Lemma 2 we can then find transductions R, S, T that have the desired property.

Let F' be the set of transductions provided by Lemma 8 and let $K \subseteq X^*$ be arbitrary and $M \in \mathsf{RE}(K)$, $M \subseteq X^*$. This means there is an oracle Turing machine A such that M is accepted by A^K. We will use the extended alphabet $Y = \{0, 1, \#_1, \#_2\}$ and a standard encoding $g : Y^* \to \{0, 1\}^*$. Let $M' \subseteq Y^*$ be the set of words

$$u_1\#_1 \cdots u_n\#_1 v_1\#_2 \cdots v_m\#_2 w$$

such that there is an accepting computation in A with input w and in which oracle queries about $u_1, \ldots, u_n$ are made with a positive result and oracle queries about $v_1, \ldots, v_m$ are made with a negative result. Note that this does not mean that $u_i \in K$ or $v_i \notin K$, we collect all computations that A could make and what inputs would be accepted provided that an oracle answered as specified. Then M' is clearly recursively enumerable. Therefore, $g(M')$ can be obtained from L by transductions in F' and Boolean operations.

Hence, we can obtain $M' = g^{-1}(g(M'))$ from L. Furthermore, since

$$(K\#_1)^* = \overline{(X^*\#_1)^*\overline{K}\#_1(X^*\#_1)} = \overline{T_1\overline{K}}, \qquad (\overline{K}\#_2)^* = \overline{(X^*\#_2)^*K\#_2(X^*\#_2)} = \overline{T_2 K}$$

for some rational transductions T_1, T_2, we can construct $(K\#_1)^*$ and $(\overline{K}\#_2)^*$ from K. Moreover, since

$$\begin{aligned} M'' &= \{u_1\#_1 \cdots u_n\#_1 v_1\#_2 \cdots v_m\#_2 w \in M' \mid u_1, \ldots, u_n \in K, \;\; v_1, \ldots, v_m \in \overline{K}\} \\ &= M' \;\cap\; (K\#_1)^*(X^*\#_2)^*X^* \;\cap\; (X^*\#_1)^*(\overline{K}\#_2)^*X^* \\ &= M' \;\cap\; T_3(K\#_1)^* \;\cap\; T_4(\overline{K}\#_2)^* \end{aligned}$$

for suitable rational transductions T_3, T_4, we can construct M'' from K and L. If we now apply a transduction T_5 that for an input from Y^* outputs the longest suffix in X^*, we obtain M from K and L. Since, apart from the transductions in F', we only used g^{-1} and $T_1, \ldots, T_5$, the lemma follows. ◀

▶ **Corollary 10.** *Let $L \subseteq X^*$ be a non-regular language. Then $\mathsf{AH}(L)$ is the smallest Boolean closed full trio containing L.*

Proof. Let $\mathcal{T}$ be the smallest Boolean closed full trio containing L. If $|X| \leq 2$, Theorem 9 implies that $\mathcal{T}$ includes $\mathsf{AH}(L)$. If $|X| > 2$, let $g : X^* \to \{0,1\}^*$ be a standard encoding. Then $g(L)$ is non-regular as well and we have $\mathsf{AH}(L) = \mathsf{AH}(g(L))$. Hence, according to Theorem 9, $\mathcal{T}$ includes $\mathsf{AH}(L) = \mathsf{AH}(g(L))$. The fact that $\mathsf{AH}(L)$ is a Boolean closed full trio concludes the proof. ◀

The following corollary applies to a wide range of language classes. A *full semi-AFL* is a union closed full trio. Although the authors are not aware of any particular full semi-AFL for which it is not known whether complementation closure is available, the following fact is interesting because of its generality.

▶ **Corollary 11.** *Other than the regular languages, no full semi-AFL $\mathcal{C} \subseteq \mathsf{RE}$ is closed under complementation.*

Proof. Suppose $\mathcal{C}$ were a complementation closed full semi-AFL that contains a non-regular language. According to Theorem 9, it would already include AH and thus not be included in RE. ◀

Note that the following corollary is not a special case of Corollary 11 as it is not restricted to language classes below RE.

▶ **Corollary 12.** *A principal full trio is closed under complementation if and only if it coincides with the regular languages.*

Proof. Let $\mathcal{T}$ be a principal full trio generated by the language L. If L is regular, $\mathcal{T}$ coincides with the regular languages and is therefore closed under complementation.

Suppose L is not regular. $\mathcal{T}$ consists of all languages of the form RL, where R is a rational transduction. Hence, $\mathcal{T}$ is contained in $\mathsf{RE}(L)$ and closed under union. If $\mathcal{T}$ were closed under complementation, it would be closed under all Boolean operations and thus, by Theorem 9, contain $\mathsf{AH}(L)$. Since $\mathsf{RE}(L) \subsetneq \mathsf{AH}(L)$, this is a contradiction. ◀

▶ **Corollary 13.** *For finitely generated monoids M, the following are equivalent:*

1. $\mathsf{VA}(M)$ *is closed under complementation.*
2. $\mathsf{VA}(M) = \mathsf{REG}$.
3. *M has only finitely many right-invertible elements.*

Proof. Let L be the identity language corresponding to some finite generating set of M. Since $\mathsf{VA}(M)$ is the principal full trio generated by L, Corollary 12 yields the equivalence between 3a and 3b. The equivalence between 3b and 3c has been shown in [14] (and independently in [16]). ◀

4 Rational Kripke frames

Theorem 9 can be also restated in terms of multimodal logic. A *Kripke structure* (or edge- and node-labeled graph) is a tuple

$$\mathcal{K} = (V, (E_a)_{a\in A}, (U_p)_{p\in P}),$$

where V is a set of nodes (also called worlds), A and P are finite sets of actions and propositions, respectively, for every $a \in A$, $E_a \subseteq V \times V$, and for every $p \in P$, $U_p \subseteq V$. The tuple $\mathcal{F} = (V, (E_a)_{a\in A})$ is then also called a *Kripke frame*. We say that $\mathcal{K}$ (and $\mathcal{F}$) is *word-based* if $V = X^*$ for some finite alphabet X. Formulas of *multimodal logic* are defined by the following grammar, where $p \in P$ and $a \in A$:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box_a\varphi \mid \Diamond_a\varphi.$$

The semantics $[\![\varphi]\!]_{\mathcal{K}} \subseteq V$ of formulas φ in $\mathcal{K}$ is defined inductively as follows:

$$\begin{aligned}
[\![p]\!]_{\mathcal{K}} &= U_p,\\
[\![\neg\varphi]\!]_{\mathcal{K}} &= V \setminus [\![\varphi]\!]_{\mathcal{K}},\\
[\![\varphi \wedge \psi]\!]_{\mathcal{K}} &= [\![\varphi]\!]_{\mathcal{K}} \cap [\![\psi]\!]_{\mathcal{K}},\\
[\![\varphi \vee \psi]\!]_{\mathcal{K}} &= [\![\varphi]\!]_{\mathcal{K}} \cup [\![\psi]\!]_{\mathcal{K}},\\
[\![\Box_a\varphi]\!]_{\mathcal{K}} &= \{v \in V \mid \forall u \in V : (v,u) \in E_a \to u \in [\![\varphi]\!]_{\mathcal{K}}\},\\
[\![\Diamond_a\varphi]\!]_{\mathcal{K}} &= \{v \in V \mid \exists u \in V : (v,u) \in E_a \wedge u \in [\![\varphi]\!]_{\mathcal{K}}\}.
\end{aligned}$$

A word-based Kripke frame $\mathcal{F} = (X^*, (E_a)_{a\in A})$ is called *rational* if every E_a is a rational transduction. Rational Kripke frames with a single relation are also known as *rational graphs* and have been studied intensively [5, 12, 13]. A word-based Kripke structure $\mathcal{K} = (X^*, (E_a)_{a\in A}, (U_p)_{p\in P})$ is called *rational* if every relation E_a is a rational transduction and every U_p is a regular language. The closure properties of regular languages imply that for every rational Kripke structure $\mathcal{K}$ and every multimodal formula φ, the set $[\![\varphi]\!]_{\mathcal{K}}$ is a regular language that can be effectively constructed from φ and (automata describing the structure) $\mathcal{K}$. Using this fact, Bekker and Goranko [2] proved that the *model-checking problem* for rational Kripke structures and multimodal logic is decidable. This problem has as input a rational Kripke structure $\mathcal{K}$ (given by a tuple of automata and transducers), a word $w \in X^*$ (where X^* is the node set of $\mathcal{K}$), and a multimodal formula φ, and it is asked whether $w \in [\![\varphi]\!]_{\mathcal{K}}$ holds. In contrast, there exist rational graphs (even acyclic ones) with an undecidable first-order theory [5, 15], but every rational tree has a decidable first-order theory [5]. Rational Kripke structures and frames were also considered in the context of querying graph databases [1].

Our reformulation of Theorem 9 in terms of multimodal logic is:

▶ **Theorem 14.** *Let $X = \{0,1\}$. There are rational transductions E_r, E_s, E_t in X^* such that the rational Kripke frame $\mathcal{F} = (X^*, E_r, E_s, E_t)$ has the following property: For every non-regular language $U_p \subseteq X^*$ and every language $K \in \mathsf{AH}(U_p)$, $K \subseteq X^*$, there exists a multimodal formula φ such that $K = [\![\varphi]\!]_{\mathcal{K}}$, where $\mathcal{K} = (X^*, E_r, E_s, E_t, U_p)$.*

Proof. Take the rational transductions R, S, T provided by Theorem 9. Let $U_p \subseteq X^*$ be a non-regular language and take the Kripke structure $\mathcal{K} = (X^*, E_r, E_s, E_t, U_p)$, where $E_r = R$, $E_s = S$, and $E_t = T$. By induction, we can construct for every language K obtainable from U_p by the transductions R, S, T and Boolean operations a multimodal formula φ with $K = [\![\varphi]\!]_{\mathcal{K}}$. For instance, if $K = [\![\psi]\!]_{\mathcal{K}}$, then $RK = [\![\Diamond_r\psi]\!]_{\mathcal{K}}$. The theorem follows immediately. ◀

The question arises whether an analogous statement holds when we allow choosing an arbitrary non-rational transduction instead of an arbitrary non-regular language. In other words: Are there rational transductions $R_1, \ldots, R_n$ and regular languages $L_1, \ldots, L_m$ over an alphabet X such that for any non-rational transduction T, the Kripke structure $(X^*, R_1, \ldots, R_n, T, L_1, \ldots, L_m)$ allows to define every arithmetical language in multimodal logic? The answer is no, since there are non-rational transductions T that preserve regularity, i.e., for which TL is regular whenever L is regular. Take, for example, the transduction $T = \{(w, ww) \mid w \in X^*\}$. It is clearly not rational, since $T^{-1}X^* = \{ww \mid w \in X^*\}$ is not regular. However, it is not hard to see that TL is effectively regular for regular languages L [17]. In particular, for every choice of $R_1, \ldots, R_n$ and $L_1, \ldots, L_m$ as above, every language definable in $(X^*, R_1, \ldots, R_n, T, L_1, \ldots, L_m)$ is regular and effectively constructible, implying that the model-checking problem is decidable.

5 Open problems

An interesting open problem is whether in Theorem 9 one can replace the rational transductions by suitable synchronized rational relations. A relation $R \subseteq X^* \times X^*$ is synchronized rational if the set of all convolutions $u \otimes v$ with $(u, v) \in R$ is a rational language. The convolution of two words $u = a_1 a_2 \cdots a_n$ and $v = b_1 b_2 \cdots b_m$ is the word $(a_1, b_1)(a_2, b_2) \cdots (a_k, b_k)$ where $k = \max\{n, m\}$, $a_i = \#$ for $i > n$, and $b_i = \#$ for $i > m$. Here, $\#$ is a fresh symbol not appearing in any pair from R. In other words, R can be recognized by an automaton on two tapes where both heads move synchronously. Synchronized rational relations underlie the definition of automatic structures [9]. Note that the rational transductions used in the proof of Theorem 9 are not synchronized rational.

Another open question is whether the number of rational transductions in Theorem 9 can be reduced to 1 or 2.

References

1 Pablo Barceló, Diego Figueira, and Leonid Libkin. Graph logics with rational relations and the generalized intersection problem. In *LICS*, pages 115–124. IEEE, 2012.

2 Wilmari Bekker and Valentin Goranko. Symbolic model checking of tense logics on rational Kripke models. In Margaret Archibald, Vasco Brattka, Valentin Goranko, and Benedikt Löwe, editors, *ILC*, volume 5489 of *Lecture Notes in Computer Science*, pages 2–20. Springer, 2007.

3 Jean Berstel. *Transductions and context-free languages.* Teubner, Stuttgart, 1979.

4 Ronald V. Book. Simple representations of certain classes of languages. *Journal of the ACM*, 25(1):23–31, 1978.

5 Arnaud Carayol and Christophe Morvan. On rational trees. In Zoltán Ésik, editor, *CSL*, volume 4207 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2006.

6 Henning Fernau and Ralf Stiebe. Sequential grammars and automata with valences. *Theoretical Computer Science*, 276:377–405, 2002.

7 Sheila A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7(3):311–324, 1978.

8 J. Hartmanis and J.E. Hopcroft. What makes some language theory problems undecidable. *Journal of Computer and System Sciences*, 4(4):368–376, 1970.

9 Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In *LCC: International Workshop on Logic and Computational Complexity*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer, 1995.

10 Stephen Cole Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ, 1956.

11 Dexter C. Kozen. *Automata and computability*. Springer-Verlag, New York, 1997.

12 Christophe Morvan. On rational graphs. In Jerzy Tiuryn, editor, *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2000), Berlin (Germany)*, number 2303 in Lecture Notes in Computer Science, pages 252–266. Springer, 2000.

13 Christophe Morvan and Colin Stirling. Rational graphs trace context-sensitive languages. In Jiri Sgall, Ales Pultr, and Petr Kolman, editors, *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001), Marianske Lazne (Czech Republic)*, number 2136 in Lecture Notes in Computer Science, pages 548–559. Springer, 2001.

14 Elaine Render. *Rational Monoid and Semigroup Automata*. PhD thesis, University of Manchester, 2010.

15 Wolfgang Thomas. A short introduction to infinite automata. In Werner Kuich, Grzegorz Rozenberg, and Arto Salomaa, editors, *Proceedings of the 5th International Conference on Developments in Language Theory (DLT 2001), Vienna (Austria)*, number 2295 in Lecture Notes in Computer Science, pages 130–144. Springer, 2001.

16 Georg Zetzsche. On the capabilities of grammars, automata, and transducers controlled by monoids. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *Automata, Languages and Programming 38th International Colloquium, ICALP 2011, Zürich, Switzerland, July 4-8, 2011, Proceedings, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2011.

17 Guo-Qiang Zhang. Automata, boolean matrices, and ultimate periodicity. *Information and Computation*, 152(1):138–154, 1999.

Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but were afraid to ask)

Dániel Marx[*1] **and Michał Pilipczuk**[†2]

1 **Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SZTAKI), Hungary**
dmarx@cs.bme.hu

2 **Department of Informatics, University of Bergen, Norway**
michal.pilipczuk@ii.uib.no

Abstract

Given two graphs H and G, the Subgraph Isomorphism problem asks if H is isomorphic to a subgraph of G. While NP-hard in general, algorithms exist for various parameterized versions of the problem. However, the literature contains very little guidance on which combinations of parameters can or cannot be exploited algorithmically. Our goal is to systematically investigate the possible parameterized algorithms that can exist for Subgraph Isomorphism.

We develop a framework involving 10 relevant parameters for each of H and G (such as treewidth, pathwidth, genus, maximum degree, number of vertices, number of components, etc.), and ask if an algorithm with running time $f_1(p_1, p_2, \ldots, p_\ell) \cdot n^{f_2(p_{\ell+1},\ldots,p_k)}$ exists, where each of $p_1, \ldots, p_k$ is one of the 10 parameters depending only on H or G. We show that *all* the questions arising in this framework are answered by a set of 11 maximal positive results (algorithms) and a set of 17 maximal negative results (hardness proofs); some of these results already appear in the literature, while others are new in this paper.

On the algorithmic side, our study reveals for example that an unexpected combination of bounded degree, genus, and feedback vertex set number of G gives rise to a highly nontrivial algorithm for Subgraph Isomorphism. On the hardness side, we present W[1]-hardness proofs under extremely restricted conditions, such as when H is a bounded-degree tree of constant pathwidth and G is a planar graph of bounded pathwidth.

1998 ACM Subject Classification G.2.2 Graph algorithms

Keywords and phrases parameterized complexity, subgraph isomorphism

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.542

1 Introduction

Subgraph Isomorphism is one of the most fundamental graph-theoretic problems: given two graphs H and G, the question is whether H is isomorphic to a subgraph of G. It can be easily seen that finding a k-clique, a k-path, a Hamiltonian cycle, a perfect matching, or a partition of the vertices into triangles are all special cases of Subgraph Isomorphism. Therefore, the

* Research supported by the European Research Council (ERC) grant "PARAMTIGHT: Parameterized complexity and the search for tight complexity results," reference 280152 and OTKA grant NK105645.
† Research supported by the European Research Council (ERC) grant "Rigorous Theory of Preprocessing", reference 267959.



© Dániel Marx and Michał Pilipczuk;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 542–553



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



problem is clearly NP-complete in general. There are well-known polynomial-time solvable special cases of the problem, for example, the special case of trees:

▶ **Theorem 1** ([27]). Subgraph Isomorphism *is P-time solvable if G and H are trees.*

Theorem 1 suggests that one should look at cases of Subgraph Isomorphism involving "tree like" graphs. The notion of treewidth measures, in some sense, how close a graph is to being a tree [3]. Treewidth has very important combinatorial and algorithmic applications; in particular, many algorithmic problems become easier on bounded-treewidth graphs. However, Subgraph Isomorphism is NP-hard even if both H and G have treewidth at most 2 [26].

Parameterized algorithms try to cope with NP-hardness by allowing exponential dependence of the running time on certain well-defined parameters of the input, but otherwise the running time depends only polynomially on the input size. We say that a problem is *fixed-parameter tractable* with a parameter k if it can be solved in time $f(k) \cdot n^{O(1)}$ for some computable function f depending only on k [13]. The definition can be easily extended to multiple parameters $k_1, \ldots, k_\ell$. The NP-hardness of Subgraph Isomorphism on graphs of treewidth at most 2 shows that the problem is not fixed-parameter tractable parameterized by treewidth (under standard complexity assumptions). However, there are tractability results that involve other parameters besides treewidth. For example, the following theorem, which follows easily from e.g. Courcelle's Theorem [6], shows the fixed-parameter tractability of Subgraph Isomorphism, jointly parameterized by the size of H and the treewidth of G:

▶ **Theorem 2** (cf. [13]). Subgraph Isomorphism *can be solved in time $f(|V(H)|, \mathbf{tw}(G)) \cdot n$ for some computable function f.*

Some of the results in the literature can be stated as algorithms where certain parameters do appear in the exponent of the running time, but others influence only the multiplicative factor. The classical color-coding algorithm of Alon, Yuster, and Zwick [1] is one such result:

▶ **Theorem 3** ([1]). Subgraph Isomorphism *can be solved in time $2^{O(|V(H)|)} \cdot n^{O(\mathbf{tw}(H))}$.*

One can interpret Theorem 3 as saying that if the treewidth of H is bounded by any fixed constant, then the problem becomes fixed-parameter tractable when parameterized by $|V(H)|$. Notice that treewidth appears in very different ways in Theorems 2 and 3: in the first result, the treewidth of G appears in the multiplicative factor, while in the second result, it is the treewidth of H that is relevant and it appears in the exponent. Yet another algorithm for Subgraph Isomorphism on bounded-treewidth graphs is due to Matoušek and Thomas [26]:

▶ **Theorem 4** ([26]). *For connected H,* Subgraph Isomorphism *can be solved in time $f(\Delta(H)) \cdot n^{O(\mathbf{tw}(G))}$ for some computable function f.*

Again, the dependence on treewidth takes a different form here: now it is the treewidth of G that appears in the exponent. Note that the connectivity condition cannot be omitted: there is an easy reduction from the NP-hard problem Bin Packing with unary sizes to the case of Subgraph Isomorphism where H and G both consist of a set of disjoint paths, i.e., have maximum degree 2 and treewidth 1. Therefore, as Theorem 4 shows, the complexity of the problem depends nontrivially on the number of connected components of the graphs as well.

As the examples above show, even the apparently simple question of how treewidth influences the complexity of Subgraph Isomorphism does not have a clear-cut answer: the treewidth of H and G influences the complexity in different ways, they can appear in the running time either as an exponent or as a multiplier, and the influence of treewidth can be interpreted only in combination with other parameters (such as the number of vertices or

maximum degree of H). The situation becomes even more complex if we consider further parameters of the graphs as well. Cliquewidth, introduced by Courcelle and Olariu [8], is a graph measure that can be always bounded by a function of treewidth, but treewidth can be arbitrary large even for graphs of bounded cliquewidth (e.g., for cliques). Therefore, algorithms for graphs of bounded cliquewidth are strictly more general than those for graphs of bounded treewidth. By the results of Courcelle et al. [7], Theorem 2 can be generalized by replacing treewidth with cliquewidth. However, no such generalization is possible for Theorem 3: cliques have cliquewidth 2, thus replacing treewidth with cliquewidth in Theorem 3 would imply that Clique (parameterized by the size of the clique to be found) is fixed-parameter tractable, contrary to widely accepted complexity assumptions. In the case of Theorem 4, it is not at all clear if treewidth can be replaced by cliquewidth: we are not aware of any result in the literature on whether Subgraph Isomorphism is fixed-parameter tractable parameterized by the maximum degree of H if G is a connected graph whose cliquewidth is bounded by a fixed constant.

Theorem 2 can be generalized into a different direction using the concept of *bounded local treewidth.* Model checking with a fixed first-order formula is known to be linear-time solvable on graphs of bounded local treewidth [15], which implies that Subgraph Isomorphism can be solved in time $f(|V(H)|)\cdot n$ if G is planar, or more generally, in time $f(|V(H)|, \mathbf{genus}(G))\cdot n$ for arbitrary G. Having an algorithm for bounded-genus graphs, one can try to further generalize the results to graphs excluding a fixed minor or to graphs not containing the subdivision of a fixed graph (that is, to graphs not containing a fixed graph as a topological minor). Such a generalization is possible: a result of Dvořak et al. [10] states that model checking with a fixed first-order formula is linear-time solvable on graphs of bounded expansion, and it follows that Subgraph Isomorphism can be solved in time $f(|V(H)|, \mathbf{hadw}(G)) \cdot n$ or $f(|V(H)|, \mathbf{hadw}_{\mathrm{T}}(G)) \cdot n$, where $\mathbf{hadw}(G)$ (resp., $\mathbf{hadw}_{\mathrm{T}}(G)$) is the maximum size of a clique that is a minor (resp., topological minor) of G. These generalizations of Theorem 2 show that planarity, and more generally, topological restrictions on G can be helpful in solving Subgraph Isomorphism, and therefore the study of parameterizations of Subgraph Isomorphism should include these parameters as well.

Our goal is to perform a systematic study of the influence of the parameters: for all possible combination of parameters in the exponent and in the multiplicative factor, we would like to determine if there is an algorithm whose running time is of this form. The main thesis of the paper is the following: (1) as the influence of the parameters on the complexity is highly nontrivial and subtle, even small changes in the choice of parameters can have substantial and counterintuitive consequences, and (2) the current literature gives very little guidance on whether an algorithm with a particular combination of parameters exist.

2 Our framework

We present a framework in which the questions raised above can be systematically treated and completely answer every question arising in the framework. Our setting is the following. First, we define the following 10 graph parameters (we give a brief justification for each parameter why it is relevant for the study of Subgraph Isomorphism):

- *Number of vertices* $|V(\cdot)|$. As Theorems 2 and 3 show, $|V(H)|$ is a highly relevant parameter for the problem. Note, however, that the problem becomes trivial if $|V(G)|$ can appear in the multiplier or in the exponent, or if $|V(H)|$ can appear in the exponent.
- *Number of connected components* $\mathbf{cc}(\cdot)$. As Theorem 4 and the reduction from Bin Packing show, it makes a difference if we restrict the problem to connected graphs (or, more generally, if we allow the running time to depend on the number of components).

- *Maximum degree* $\Delta(\cdot)$. The maximum degree of H plays an important role in Theorem 4, thus exploring the effect of this parameter is clearly motivated. In general, many parameterized problems become easier on bounded-degree graphs, mainly because then the distance-d neighborhood of each vertex has bounded size for bounded d.
- *Treewidth* $\mathbf{tw}(\cdot)$. Theorems 2–4 give classical algorithms where treewidth appears in different ways; understanding how exactly treewidth can influence complexity is one of the most important concrete goals of the paper.
- *Pathwidth* $\mathbf{pw}(\cdot)$. As pathwidth is always at least treewidth, but can be strictly larger, algorithms parameterized by pathwidth can exist even if no algorithms parameterized by treewidth are possible. Given the importance of treewidth, it is natural to explore the possibility of algorithms in the more restricted setting of bounded-pathwidth graphs.
- *Feedback vertex set number* $\mathbf{fvs}(\cdot)$. A feedback vertex set is a set of vertices whose deletion makes the graph a forest; the feedback vertex set number is the size of the smallest such set. Similarly to graphs of bounded pathwidth, graphs of bounded feedback vertex set number form a subclass of bounded-treewidth graphs, hence it is natural to explore what algorithms we can obtain with this parameterization. Note that Graph Isomorphism (not subgraph!) is fixed-parameter tractable parameterized by feedback vertex set number [19], while only $n^{O(\mathbf{tw}(G))}$ time algorithms are known parameterized by treewidth [2, 29]. This shows that $\mathbf{fvs}(\cdot)$ can be a useful parameter for problems involving isomorphisms.
- *Cliquewidth* $\mathbf{cw}(\cdot)$. As cliquewidth is bounded by a function of treewidth, parameterization by cliquewidth leads to more general algorithms than parameterization by treewidth. However, treewidth can be replaced by cliquewidth in Theorem 2, but not in Theorem 3. Therefore, understanding the role of cliquewidth is a nontrivial and interesting challenge.
- *Genus* $\mathbf{genus}(\cdot)$. Understanding the complexity of Subgraph Isomorphism on planar graphs (and more generally, on bounded-genus graphs) is a natural goal, especially in light of the positive results that arise from the generalizations of Theorem 2.
- *Hadwiger number* $\mathbf{hadw}(\cdot)$. That is, the size of the largest clique that is the minor of the graph. A graph containing a K_k-minor needs to have genus $\Omega(k^2)$; therefore, algorithms for graphs excluding a fixed clique as a minor generalize algorithms for bounded-genus graphs. In many cases, such a generalization is possible, thanks to structure theorems and algorithmic advances for H-minor free graphs [9, 17, 30].
- *Topological Hadwiger number* $\mathbf{hadw}_{\mathrm{T}}(\cdot)$. That is, the size of the largest clique whose subdivision is a subgraph of the graph. A graph containing the subdivision of a K_k contains K_k as a minor. Therefore, algorithms for graphs excluding a fixed topological clique minor generalize algorithms for graphs excluding a fixed clique minor. Recent work show that some algorithmic results for graphs excluding a fixed minor can be generalized to excluded topological minors [14, 16, 18]. In particular, the structure theorem of Grohe and Marx [18] states, in a precise technical sense, that graphs excluding a fixed topological minor are composed from parts that are either "almost bounded-degree" or exclude a fixed minor. Therefore, it is interesting to investigate in our setting how this parameter interacts with the parameters smallest excluded clique minor and maximum degree.

Given this list of 10 parameters, we would like to understand if an algorithm with running time of the form $f_1(p_1, p_2, \ldots, p_\ell) \cdot n^{f_2(p_{\ell+1}, \ldots, p_k)}$ exists, where each p_i is one of these 10 parameters applied on either H and G, and f_1, f_2 are arbitrary computable functions of these parameters. We call such a sequence of parameters a *description,* and we say that an algorithm is *compatible* with the description if its running time is of this form. Observe that Theorems 2 and 3 can be stated as the existence of algorithms compatible with particular descriptions. However, Theorem 4 has the extra condition that H is connected (or in other

words, the number of connected components of H is 1) and therefore it does not seem to fit into this framework. In order to include such statements into our investigations, we extend the definition of descriptions with some number of *constraints* that restrict the value of certain parameters to particular constants. Specifically, we consider the following 5 constraints on H and G, each of which corresponds to a particularly motivated special case of the problem:

- *Genus is 0.* That is, the graph is planar. Any positive result on planar graphs is clearly of interest, even if it does not generalize to arbitrary fixed genus. Conversely, whenever possible, we would like to state hardness results for planar graphs, rather than for bounded-genus with an unspecified bound on the genus.
- *Number of components is 1.* Any positive result under this restriction is quite motivated, and as the examples above show, the problem can become simpler on connected graphs.
- *Treewidth is at most 1.* That is, the graph is a forest. Trees can behave very differently than bounded-treewidth graphs (compare Theorem 1 with the fact the the problem is NP-hard on graphs of treewidth 2), thus investigating the special case of forests might turn up additional algorithmic results.
- *Maximum degree is at most 2.* That is, the graph consists of disjoint paths and cycles. Clearly, this class is very restricted, but as the NP-hardness of Hamiltonian Cycle shows, this property of H does not guarantee tractability without further assumptions.
- *Maximum degree is at most 3.* To provide contrast with the case of maximum degree at most 2, we would like to state negative results for graphs of maximum degree at most 3.

We restrict our attention to these 5 specific constraints. For example, we do not specifically investigate possible algorithms that work on, say, graphs of feedback vertex set size 1 or of pathwidth 2: we can argue that such algorithms are interesting only if they can be generalized to every fixed bound on the feedback vertex set size or on pathwidth (whereas an algorithm for planar graphs is interesting even if it does not generalize to higher genera).

3 Results

Our formulation of the general framework includes an enormous number of concrete research questions. Even without considering the 5 specific constraints, we have 19 parameters (10 for H and 9 for G) and each parameter can be either in the exponent of the running time, in the multiplier of the running time, or does not appear at all in the running time. Therefore, there are at least $3^{19} \approx 10^9$ descriptions and corresponding complexity questions in this framework. The present paper answers all these questions (under standard complexity assumptions).

In order to reduce the number of questions we observe that there are some clear implications between them. Clearly, the $f_1(|V(H)|) \cdot n^{f_2(\mathbf{tw}(H))}$ time algorithm of Theorem 3 implies the existence of, say, an $f_1(|V(H)|, \mathbf{genus}(G)) \cdot n^{f_2(\mathbf{pw}(H),\Delta(G))}$ time algorithm: $\mathbf{pw}(H)$ is always at least $\mathbf{tw}(H)$ and the fact that the latter running time can depend on $\mathbf{genus}(G)$ and $\Delta(G)$ can be ignored. The main claim of the paper is that every question arising in the framework can be answered by a set of 11 positive and 17 negative results:

> The positive and negative results presented in Table 1 imply a positive or negative answer to every question arising in this framework. (*)

That is, either there is a positive result for a more restrictive description, or a negative result for a less restrictive restriction. The following two examples show how one can deduce the answer to specific questions from Table 1.

Short Description	Theorem	H										G								
		$\lvert V(\cdot)\rvert$	cc	Δ	fvs	pw	tw	cw	genus	hadw	hadw_T	cc	Δ	fvs	pw	tw	cw	genus	hadw	hadw_T
FO model checking	Theorem P.1	M															M			
	Theorem P.2	M																		M
Color coding	Theorem P.3	M					E													
Matoušek-Thomas	Theorem P.4		M	M												E				
Paths&Cycles → Paths&Cycles	Theorem P.5											E	2							
Dynamic Programming	Theorem P.6		E	2												M				
	Theorem P.7		E	2													E			
	Theorem P.8⋆		M													1				
FVS and CSPs	Theorem P.9⋆		M	2									M	M						
	Theorem P.10⋆		E										M	M				E		
	Theorem P.11⋆		E	E									M	M					E	
Bin Packing	Theorem N.1											M	2			1				
	Theorem N.2		1				1							E	E			0		
	Theorem N.3			2								1	3		E	1				
Planar cubic HamPath	Theorem N.4		1	2			1						3					0		
Clique	Theorem N.5	M	1					E												
HamPath in bounded **cw**	Theorem N.6		1	2			1										M			
Grid Tiling, 1-in-n gadgets	Theorem N.7⋆		M			E	1					1	3	M	M			0		
	Theorem N.8⋆		1			E	1						M	M	M			M	E	
	Theorem N.9⋆		1			E	1						3	M	M			M		
	Theorem N.10⋆		1	3		E	1						M	M	M		E	M		
Grid Tiling, moustache gadgets	Theorem N.11⋆		1	3		E	1							M	M			0		
	Theorem N.12⋆		1			E	1						3		M			0		
Small planar graph	Theorem N.13⋆	M	1	3					0											
Exact Planar Arc Supply	Theorem N.14⋆		M	2			1					1		M	M			0		
	Theorem N.15⋆		M	2			1					1	3		M			0		
	Theorem N.16⋆		M	2			1					1		M	M		E	M		
	Theorem N.17⋆		M	2			1					1	M		M		E	M		

Figure 1 Positive (blue) and negative (rosa) results in the paper; the numbering refers to the full version. Asterisks denote new findings not known before. Entry **M** denotes that the parameter appears in the multiplier, entry **E** denotes that the parameter appears in the exponent, while explicit integer constants denote constraining the parameter to be bounded by the respective constant value.

▶ **Example 5.** Is there an algorithm for Subgraph Isomorphism with running time $n^{f(\mathbf{fvs}(G))}$ when G is a planar graph of maximum degree 3 and H is connected? Looking at Table 1, the line of Theorem P.10 shows the existence of an algorithm with running time $f_1(\mathbf{fvs}(G), \Delta(G)) \cdot n^{f_2(\mathbf{genus}(G),\mathbf{cc}(H))}$. When restricted to the case when G is a planar graph (i.e., $\mathbf{genus}(G) = 0$) with $\Delta(G) \leq 3$ and H is connected (i.e., $\mathbf{cc}(H) = 1$), then running time of this algorithm can be expressed as $f(\mathbf{fvs}(G)) \cdot n^{O(1)}$. This is in fact better than the running time $n^{f(\mathbf{fvs}(G))}$ we asked for, hence the answer is positive.

▶ **Example 6.** Is there an algorithm for Subgraph Isomorphism with running time $f(\mathbf{tw}(G)) \cdot n^{g(\Delta(G))}$ when G is a connected planar graph? Looking at Table 1, the line of Theorem N.7 gives a negative result for algorithms with running time $f_1(\mathbf{cc}(H), \mathbf{pw}(G), \mathbf{fvs}(G)) \cdot n^{f_2(\mathbf{pw}(H))}$ when restricted to instances where H is a forest and G is a connected planar graph of maximum degree 3. Note that $\mathbf{tw}(G) \leq \mathbf{pw}(G)$, so an $f(\mathbf{tw}(G)) \cdot n^{g(\Delta(G))}$ time algorithm for connected planar graphs would give an $f(\mathbf{pw}(G)) \cdot n^{O(1)}$ time algorithm for connected planar graphs of maximum degree 3, which is a better running time then the one ruled out by Theorem N.7. Therefore, the answer is negative.

To make claim (*) formal and verifiable, we define an ordering relation between descriptions in a way that guarantees that if description D_1 is *stronger* than D_2, then an algorithm compatible with D_1 implies the existence of an algorithm compatible with D_2. Roughly speaking, the definition of this ordering takes into account three immediate implications:

- Removing a parameter makes the description stronger.
- Moving a parameter from the exponent to the multiplier makes the description stronger.
- We consider a list of combinatorial relations between the parameters and their implications on the descriptions: for example, $\mathbf{tw}(H) \leq \mathbf{pw}(H)$ implies that replacing $\mathbf{pw}(H)$ with $\mathbf{tw}(H)$ makes the description stronger. Our list of relations include some more complicated and less obvious connections, such as $\mathbf{tw}(H)$ can be bounded by a function of $\mathbf{cw}(H)$ and $\Delta(H)$, thus replacing $\mathbf{cw}(H)$ and $\Delta(H)$ with $\mathbf{tw}(H)$ makes the description stronger.

The precise definition of the ordering of the descriptions appears in the full version of the paper. Given the ordering, we need to show the positive results only for the maximally strong descriptions and the negative results for the minimally strong descriptions. Our main result is that every question arising in the framework can be explained by a set of 11 maximally strong positive results and a set of 17 minimally strong negative results listed in Table 1.

▶ **Theorem 7.** *For every description D, either (a) Table 1 contains a positive result for a description D' such that D' is stronger than D, or (b) Table 1 contains a negative result for a description D' such that D is stronger than D'.*

At this point, the reader might wonder how it is possible to prove Theorem 7, that is to verify that the positive and negative results on Table 1 indeed cover every possible description. Interestingly, formulating the task of checking whether a set of positive and negative results on an unbounded set of parameters explains every possible description leads to an NP-hard problem (we omit the details). Therefore, we have implemented a simple backtracking algorithm that checks if every description is explained by the set of positive and negative results given in the input. We did not make a particular effort to optimize the program, as it was sufficiently fast for our purposes on contemporary desktop computers. The program indeed verifies that our set of positive and negative results is complete. We have used this program extensively during our research to find descriptions that are not yet explained by our current set of results. By focusing on one concrete unexplained description,

we could always either find a corresponding algorithm or prove a hardness result, which we could add to our set of results. By iterating this process, we have eventually arrived at a set of results that is complete. The program and the data files are available as electronic supplementary material of the arxiv version of the present paper [25].

As the systematic study of our framework involves proving dozens of results that require combination of many different tools, in this extended abstract we only survey our framework and state the results, giving a short glimpse into the most important findings and techniques used for proving them. For a full discussion of the results, including all the proofs, we refer to the full version of the paper that can be found on arxiv [25].

4 Algorithms

Let us highlight some of the new algorithmic results discovered by the exhaustive analysis of our framework. While the negative results suggest that the treewidth of G appearing in the multiplicative factor of the running time helps very little if the size of H can be large, we show that the more relaxed parameter feedback vertex set is useful on bounded-degree planar graphs. Specifically, we prove the following result:

▶ **Theorem 8.** Subgraph Isomorphism *can be solved in time* $f(\Delta(G), \mathbf{fvs}(G)) \cdot n^{O(1)}$ *if* H *is connected and* G *is planar.*

The proof of Theorem 8 turns the Subgraph Isomorphism problem into a Constraint Satisfaction Problem (CSP) whose primal graph is planar. We observe that this CSP instance has a special variable v that we call a *projection sink:* roughly speaking, it has the property that v can be reached from every other variable via a sequence of constraints that are projections. We prove the somewhat unexpected result that a planar CSP instance having a projection sink is polynomial-time solvable, which allows us to solve the Subgraph Isomorphism instance within the claimed time bound. This new property of having a projection sink and the corresponding polynomial-time algorithm for CSPs with this property can be interesting on its own and possibly useful in other contexts.

We generalize the result from planar graphs to bounded-genus graphs and to graphs excluding a fixed minor in the following way:

▶ **Theorem 9.** Subgraph Isomorphism *can be solved in time*
1. $f_1(\Delta(G), \mathbf{fvs}(G)) \cdot n^{f_2(\mathbf{genus}(G), \mathbf{cc}(H))}$, *and*
2. $f_1(\Delta(G), \mathbf{fvs}(G)) \cdot n^{f_2(\mathbf{hadw}(G), \Delta(H), \mathbf{cc}(H))}$.

For (1), we need only well-known diameter-treewidth relations for bounded-genus graphs [12], but (2) needs a nontrivial application of structure theorems for graphs excluding a fixed minor and handling vortices in almost-embeddable graphs. Note that these two results are incomparable: in (2), the exponent contains $\Delta(H)$ as well, thus it does not generalize (1). Intuitively, the reason for this is that when lifting the algorithm from the bounded-genus case to the minor-free case, high-degree apices turn out to be problematic. On the other hand, Theorem N.8 shows that incorporating other parameters is (probably) unavoidable when moving to the more general minor-free setting. We find it interesting that our study revealed that the bounded-genus case and the minor-free case are provably different when the parameterized complexity of Subgraph Isomorphism is concerned.

The reader might find it unmotivated to present algorithms that depend on so many parameters in strange ways, but let us emphasize that these results are maximally strong results in our framework. That is, no weakening of the description can lead to an algorithm

(under standard complexity assumptions): for example, $\mathbf{genus}(H)$ or $\mathbf{cc}(H)$ cannot be moved from the exponent to the multiplier, or $\Delta(H)$ cannot be omitted from the exponent in (2). Therefore, these result show, in a well-defined sense, the limits of what can be achieved. Finding such maximal results is precisely the goal of developing and analyzing our framework: it seems unlikely that one would come up with results of the form of Theorem 9 without an exhaustive investigation of all the possible combinations of parameters.

On the other hand, we generalize Theorem 1 from trees to forests, parameterized by the number of connected components of H. This seemingly easy task turns out to be surprisingly challenging. The dynamic programming algorithm of Theorem 1 relies on a step that involves computing maximum matching in a bipartite graph. The complications arising from the existence of multiple components of H makes this matching step more constrained and significantly harder. In fact, the only way we were able to solve these matching problems is by the randomized algebraic matching algorithm of Mulmuley et al. [28]. Therefore, our result is a randomized algorithm for this problem:

▶ **Theorem 10.** Subgraph Isomorphism *can be solved in randomized time* $f(\mathbf{cc}(H)) \cdot n^{O(1)}$ *with false negatives, if* H *and* G *are forests.*

Again, we find it a success of our framework that it directed attention to this particularly interesting special case of the problem. Obtaining a deterministic algorithm for this variant is an interesting open problem.

5 Hardness proofs

Two different technologies are needed for proving negative results about algorithms satisfying certain descriptions: NP-hardness and W[1]-hardness. Recall that a W[1]-hard problem is unlikely to be fixed-parameter tractable and one can show that a problem is W[1]-hard by presenting a parameterized reduction from a known W[1]-hard problem (such as Clique) to it. The most important property of a parameterized reduction is that the parameter value of the constructed instance can be bounded by a function of the parameter of the source instance; see [13] for more details.

- To give evidence that no $n^{f(p_1,\ldots,p_k)}$ time algorithm for Subgraph Isomorphism exists, one would like to show that Subgraph Isomorphism remains NP-hard on instances where the value of the parameters $p_1, \ldots, p_k$ are bounded by some universal constant.
- To give evidence that no $f_1(p_1, p_2, \ldots, p_\ell) \cdot n^{f_2(p_{\ell+1},\ldots,p_k)}$ time algorithm for Subgraph Isomorphism exists, one would like to show that Subgraph Isomorphism is W[1]-hard parameterized by $p_1, \ldots, p_\ell$ on instances where the values of $p_{\ell+1}, \ldots, p_k$ are bounded by some universal constant. That is, what is needed is a parameterized reduction from a known W[1]-hard problem to Subgraph Isomorphism in such a way that parameters $p_1, \ldots, p_\ell$ of the constructed instance are bounded by a function of the parameters of the source instance, while the values of $p_{\ell+1}, \ldots, p_k$ are bounded by some universal constant.

Additionally, the reductions need to take into account the extra constraints (planarity, treewidth 1, etc.) appearing in the description. The nontrivial results of this paper are of the second type: we prove the W[1]-hardness of Subgraph Isomorphism with certain parameters, under the assumption that certain other parameters are bounded by a universal constant. Intuitively, a substantial difference between NP-hardness proofs and W[1]-hardness proofs is that in a typical NP-hardness proof from, say, 3-SAT, one replaces each variable and clause with a small gadget having a *constant* number of states, whereas in a typical W[1]-hardness proof from, say, Clique, one creates a bounded number of large gadgets

having an *unbounded* number of states, e.g., the states correspond to the vertices of the original graph. Therefore, usually the first goal in W[1]-hardness proofs is to construct gadgets that are able to express a large number of states.

Most of our W[1]-hardness results are for planar graphs or for graphs close to planar. As many parameterized problems become fixed-parameter tractable on planar graphs, there is only a handful of planar W[1]-hardness proofs in the literature [4, 5, 11, 24]. These hardness proofs need to construct gadgets that are both planar and able to express a large number of states, which can be a challenging task. A canonical problem that can serve as a useful starting point for W[1]-hardness proofs on planar graphs is Grid Tiling [23, 24]. Most of our W[1]-hardness proofs indeed use Grid Tiling as the source problem. In some cases we use a new problem, Exact Planar Arc Supply, which we prove to be W[1]-hard and which is inspired by the problem Planar Arc Supply introduced by Bodlaender et al. [4].

Besides planarity (or near-planarity), our hardness proofs need to overcome other challenges as well: we bound combinations of maximum degree (of H or G), pathwidth, cliquewidth etc. The following theorem demonstrates the type of restricted results we are able to get. Note that the more parameters appear in the running time and the more restrictions H and G have, the stronger the hardness result is.

▶ **Theorem 11.** *Assuming $FPT \neq W[1]$, there is no algorithm for* Subgraph Isomorphism *with running time*

- $f_1(\mathbf{pw}(G)) \cdot n^{f_2(\mathbf{pw}(H))}$, *even if both H and G are connected planar graphs of maximum degree 3 and H is a tree, or*
- $f_1(\Delta(G), \mathbf{pw}(G), \mathbf{fvs}(G), \mathbf{genus}(G)) \cdot n^{f_2(\mathbf{pw}(H), \mathbf{cw}(G))}$, *even if both H and G are connected and H is a tree of maximum degree 3.*

6 Conclusions

In this paper we have developed a framework for studying different parameterizations of Subgraph Isomorphism and completely answered every question arising in this framework. Systematic studies of parameterizations have been performed before for various problems [20, 21, 22, 31], but never on such a massive scale as in the present paper. We have demonstrated that even if the number of questions is on the order of billions, finding the maximal set of positive results and the maximal set of negative results that explain every specific question of the framework is a doable project and might involve only a few dozen concrete results. At such a large scale, even verifying that a set of results explains every possible question is a daunting task. We have resorted to the help of a computer program that checks this efficiently; the program can be helpful for similar investigations in the future.

While developing the framework and showing that it can be completely explained by a small set of results is the conceptually most novel part of the paper, we would like to emphasize that some of the concrete positive and negative results are highly nontrivial and technically novel. On the algorithmic side, we have discovered a simple, but unexpectedly challenging case: packing a forest H into a forest G, parameterized by the number of connected components of H. We presented a nontrivial randomized dynamic programming algorithm for this problem using algebraic matching algorithms. Our investigations turned up an unexpected combination of parameters that results in tractable cases: maximum degree, feedback vertex set number, and genus of G. In a somewhat surprising manner, tractability relies on the fact that a certain property, the existence of a projection sink, allows us to dramatically reduce treewidth in bounded-genus CSP instances. This new result on CSPs can be of independent interest. We have generalized the result to graphs excluding a fixed minor (with a slightly different parameterization). The generalization is not just a

straightforward application of known structure theorems: we had to use a fairly complicated dynamic programming scheme on tree decompositions to exploit the existence of a projection sink and we had to handle almost embeddable graphs including all the gory details of vortices.

On the hardness side, many of our W[1]-hardness proofs involve planar (or bounded-genus) graphs. W[1]-hardness proofs are typically involved, as they require complicated gadget constructions. Reducing from the Grid Tiling problem helps streamlining the reductions, but the actual gadgets have to be constructed in a problem-specific way. In our case the construction of gadgets is particularly challenging since we have to satisfy extreme restrictions.

It might not be apparent from the paper, but the authors did exercise some restraint when defining the framework. Only those graph parameters were included in the framework that already had some interesting nontrivial connection to the Subgraph Isomorphism problem. One could extend the framework with further parameters, such as chromatic number, girth, or (edge) connectivity, but it is not clear whether these parameters would influence the complexity of the problem in an interesting way and whether these parameters would add anything to the message of the results besides further complications. Moreover, recall that for similar reasons we have constrained ourselves to 5 particularly interesting constraints corresponding to small fixed values of certain parameters.

The reader might wonder: do the authors advocate this kind of massive investigation for each and every problem? It seems that the Subgraph Isomorphism problem is particularly suited for such treatment. First, previous results suggest that a wide range of parameters influence the complexity of the problem in nontrivial ways. Second, the Subgraph Isomorphism problem involves two graphs H and G and the same parameter for H or G can play very different role. This effectively doubles the number of parameters that need to be considered. Therefore, the problem has a very complicated ecology of parameters that can be understood only with a large-scale formal investigation. For other problems, say, Vertex Coloring, the complexity landscape is expected to be much simpler, and probably fewer new results (if any) need to be invented to explain every combination of parameters. Therefore, we suggest exploring problems using a detailed framework similar to ours only if there is evidence for complex interaction of parameters. Variants of Subgraph Isomorphism might be natural candidates for such investigations: for example, (i) the homomorphism problem for graphs, (ii) colored versions of Subgraph Isomorphism, (iii) extension versions of Subgraph Isomorphism (where we have to extend a partial subgraph isomorphism given in the input), or (iv) the counting version of Subgraph Isomorphism (this problem was suggested by Petteri Kaski).

References

1 N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

2 H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms*, 11(4):631–643, 1990.

3 H. L. Bodlaender and A. M. C. A. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. *The Computer Journal*, 51(3):255–269, 2008.

4 H. L. Bodlaender, D. Lokshtanov, and E. Penninkx. Planar Capacitated Dominating Set is W[1]-hard. In *IWPEC*, pages 50–60, 2009.

5 L. Cai, M. R. Fellows, D. W. Juedes, and F. A. Rosamond. The complexity of polynomial-time approximation. *Theory Comput. Syst.*, 41(3):459–477, 2007.

6 B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

7 B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

8 B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

9 E. D. Demaine, M. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *FOCS*, pages 637–646, 2005.

10 Z. Dvořak, D. Král, and R. Thomas. Deciding first-order properties for sparse graphs. In *FOCS*, pages 133–142, 2010.

11 R. Enciso, M. R. Fellows, J. Guo, I. A. Kanj, F. A. Rosamond, and O. Suchý. What makes equitable connected partition easy. In *IWPEC*, pages 122–133, 2009.

12 D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27:275–291, 2000.

13 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1 edition, March 2006.

14 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Linear kernels for (connected) dominating set on graphs with excluded topological subgraphs. In *STACS*, pages 92–103, 2013.

15 M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

16 M. Grohe, K. Kawarabayashi, D. Marx, and P. Wollan. Finding topological subgraphs is fixed-parameter tractable. In *STOC*, pages 479–488, 2011.

17 M. Grohe, K. Kawarabayashi, and B. A. Reed. A simple algorithm for the graph minor decomposition - logic meets structural graph theory. In *SODA*, pages 414–431, 2013.

18 M. Grohe and D. Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. In *STOC*, pages 173–192, 2012.

19 S. Kratsch and P. Schweitzer. Isomorphism for graphs of bounded feedback vertex set number. In *SWAT*, pages 81–92, 2010.

20 M. Kronegger, A. Pfandler, and R. Pichler. Parameterized complexity of optimal planning: A detailed map. In *IJCAI*, 2013.

21 M. Lackner and A. Pfandler. Fixed-parameter algorithms for closed world reasoning. In *ECAI*, pages 492–497, 2012.

22 M. Lackner and A. Pfandler. Fixed-parameter algorithms for finding minimal models. In *KR*, 2012.

23 D. Marx. On the optimality of planar and geometric approximation schemes. In *FOCS*, pages 338–348, 2007.

24 D. Marx. A tight lower bound for planar multiway cut with fixed number of terminals. In *ICALP (1)*, pages 677–688, 2012.

25 D. Marx and M. Pilipczuk. Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but were afraid to ask). *CoRR*, abs/1307.2187, 2013.

26 J. Matoušek and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1-3):343–364, 1992.

27 D. W. Matula. Subtree isomorphism in $O(n^{5/2})$. In *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 91–106. Elsevier, 1978.

28 K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

29 I. N. Ponomarenko. The isomorphism problem for classes of graphs that are invariant with respect to contraction. *Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI)*, 174(Teor. Slozhn. Vychisl. 3):147–177, 182, 1988. In Russian.

30 N. Robertson and P.D. Seymour. Graph minors XVI. Excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 77:1–27, 1999.

31 M. Samer and S. Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2):103–114, 2010.

Data-Oblivious Data Structures

John C. Mitchell and Joe Zimmerman

Department of Computer Science, Stanford University, Stanford, US
{mitchell,jzim}@cs.stanford.edu

Abstract

An algorithm is called *data-oblivious* if its control flow and memory access pattern do not depend on its input data. Data-oblivious algorithms play a significant role in secure cloud computing, since programs that are run on secret data—as in fully homomorphic encryption or secure multi-party computation—must be data-oblivious. In this paper, we formalize three definitions of data-obliviousness that have appeared implicitly in the literature, explore their implications, and show separations. We observe that data-oblivious algorithms often compose well when viewed as data structures. Using this approach, we construct data-oblivious stacks, queues, and priority queues that are considerably simpler than existing constructions, as well as improving constant factors. We also establish a new upper bound for oblivious data compaction, and use this result to show that an "offline" variant of the Oblivious RAM problem can be solved with $O(\log n \log\log n)$ expected amortized time per operation— as compared with $O(\log^2 n/\log\log n)$, the best known upper bound for the standard online formulation.

1998 ACM Subject Classification D.4.6 Security and Protection, E.1 Data Structures, F.1.1 Models of Computation, F.1.2 Modes of Computation

Keywords and phrases Data-oblivious algorithms, Data-oblivious data structures, Oblivious RAM, Secure multi-party computation, Secure cloud computing

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.554

1 Introduction

An algorithm is called *data-oblivious*, or just *oblivious*, if its control flow and memory access pattern do not depend on its input data. Data-obliviousness has compelling applications in cryptography and security. When an untrusted cloud server computes on encrypted data, the computation must be oblivious, since by assumption the server cannot learn anything about the data. Obliviousness also subsumes standard notions of security against side-channel attacks. If an adversary cannot distinguish between different primitive operations, then by making a program oblivious, we ensure that the adversary gains no information from any other source—including standard side channels such as timing, cache performance, and termination behavior. In addition, deterministic oblivious algorithms correspond directly to uniform families of Boolean circuits, and so they are useful in designing hardware implementations whose structure must be fixed before the input is known.

There has been a large body of work on the Oblivious RAM problem, which concerns a general translation of RAM programs to versions that are data-oblivious in a certain sense [7, 15, 10, 17, 6], as well as on other general data-oblivious simulations [9, 16]. In addition, there has been some work on developing data-oblivious solutions to specific problems [4, 8], but this class of questions has received relatively little attention in the past, particularly with respect to the prospect of composable data-oblivious data structures. In this work:

- We formalize three definitions of data-obliviousness—including formulations that apply to modern cloud computing settings, in which algorithms need only be oblivious with respect to part of the data.


© John C. Mitchell and Joe Zimmerman;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 554–565

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


- We give new results on data-oblivious algorithms and simple composable data structures, including new bounds for data compaction and an "offline" variant of Oblivious RAM.
- We give a number of refinements of existing results, including simpler and tighter constructions for oblivious stacks, queues, and priority queues.

2 The formal computation model

We will consider algorithms executed on a word RAM, with a word size of $\Theta(\log n)$, where n is the size of the input (or the capacity of the data structure, as appropriate), and the entire memory consists of $\text{poly}(n)$ words.[1] The RAM has a constant number of public and secret registers, and can perform arbitrary operations (of circuit size $\text{poly}(\log n)$) on a constant number of registers in constant time. (Data-oblivious algorithms are sometimes also studied in the external memory model. In the language of that model, our results assume a block size of $B = \Theta(\log n)$, and a cache of a constant number of blocks, $M = O(B)$; this is the traditional setting of Oblivious RAM [7].)

We will further assume that the machine's memory is divided into *public* and *secret* regions, and that an operation is prohibited if it could cause values originating from secret regions to influence those in public regions. The formal mechanism for this restriction depends on the machine specification, but in general, we assume standard techniques from (branching-sensitive) static information flow control [14, 13]. For example, we may specify that (i) a register is tagged as secret when its value is read from a secret memory location or is updated based on another secret register; (ii) a register does not become public again until it is directly overwritten by a constant or the contents of a public memory location; and (iii) that no secret register may be involved in a write instruction to a public memory location. The notion of obliviousness is then parameterized over which part of the input is secret: we say that an algorithm is oblivious if it never executes any instruction that sets its program counter or memory-address operands based on a secret register. Thus, if the entire memory is designated as secret, we recover the standard definition of oblivious computation.

We will also assume that each of the registers and memory locations is labeled as *deterministic* or *nondeterministic*, with the restriction that values labeled *nondeterministic* can never influence those labeled *deterministic*.[2] We assume that machines have a finite number of registers of every type (public/deterministic, public/nondeterministic, secret/deterministic, secret/nondeterministic), and we will measure memory usage of all types together. When machines have access to randomness, we will say that they are provided with two read-only one-way-infinite tapes of uniformly random bits, with one labeled public and one labeled secret, and both labeled nondeterministic.

We can now state three definitions, in increasing order of computational power.

1. **Deterministically data-independent**: *The RAM can only set its control flow based on registers that are both public and deterministic.* This definition corresponds directly to uniform families of Boolean circuits.
2. **Data-independent**: *The RAM can only set its control flow based on registers that are public (though these registers may be either deterministic or nondeterministic).* Even

[1] Our choice of such a powerful machine model is motivated by the thin asymptotic complexity margins inherent in the study of obliviousness. As we note in Section 2.1, any program can be made oblivious with overhead linear in its time complexity, and thus it would not make sense to study a simpler model such as the Turing machine, which may already incur linear overhead just by virtue of being limited to sequential memory access.

[2] As above, this restriction can be effected using standard static information-flow control rules.

though its structure varies based on random bits, the computation can still proceed without seeing the secret data.

3. **Data-oblivious**: *The RAM can only set its control flow based on public registers, but it is equipped with an additional* "declassification" operation *that moves a value from any secret register to any public register. However, the distribution of all declassified values (along with the point during execution at which each declassification occurs) is independent*[3] *of all secret inputs.* Here, the algorithm can see the secret data and branch on it, but its control flow and memory access pattern still must not reveal any secret values. This definition captures the setting of Oblivious RAM with constant client-side storage [7, 10, 3]; the physical memory locations accessed by the Oblivious RAM protocol must be independent of the logical memory locations requested.

Qualitatively, the fundamental difference between these definitions is that *data-independent* machines must satisfy a purely static constraint on the kinds of operations that are permitted on secret locations, while *data-oblivious* machines must only satisfy a dynamic condition, describing the actual effects of declassification operations at runtime. This distinction is fairly subtle. For instance, Beame and Machmouchi [1] give a lower bound for oblivious branching programs that was originally believed to apply to Oblivious RAM; however, as indicated by the authors in a supplementary note [2], their result applies only to data-independent algorithms, not to data-oblivious algorithms such as Oblivious RAM.

We also remark that, in general, deterministic data-independence is a very strong condition. For hardware implementations, it may be the case that the control flow of the algorithm is fixed a priori and cannot depend on any random bits at runtime. In virtually all applications in cryptography and security, however, no such restriction exists in practice. Even in the case of fully homomorphic encryption, the server operating on ciphertexts may generate and encrypt its own random bits.[4] Thus, for the remainder of this paper, we will use the term *data-independent* to refer to probabilistically data-independent algorithms, unless otherwise indicated.

Finally, we will sometimes want to enable the machine to execute cryptographic operations, such as evaluating a pseudorandom function (PRF), at unit cost. While this feature is standard in the Oblivious RAM setting [7, 10], it introduces subtle difficulties. The largest cryptographic key that will fit in the machine's $O(1)$ registers is of size $\Theta(\log n)$—which cannot possibly provide security better than $2^{\Theta(\log n)} = \text{poly}(n)$. So instead, whenever cryptographic operations are relevant, we introduce an additional security parameter λ, and assume that the adversary's advantage is negligible in any relevant cryptographic game when instantiated with λ as the security parameter.[5] In such settings, when we write $O(t(n))$, it should be taken to mean $O(t(n) \cdot (1 + \text{poly}(\lambda / \log n)))$, i.e., $O(t(n))$ standard RAM operations plus $O(t(n))$ cryptographic operations.

---

[3] We extend this definition in the natural way to *statistically data-oblivious*, in which the distributions for two different secret inputs are statistically close; or *computationally data-oblivious*, in which the distributions are computationally indistinguishable. In this case, we refer to the original definition as *perfectly data-oblivious*.

[4] Indeed, even when the server must provide proof that it conducted the computation correctly, it can still simulate the same effect by evaluating a pseudorandom function homomorphically.

[5] In practical settings, this may require a superpolynomial hardness assumption, since we generally always have $\lambda < n$. An alternate approach, suggested by Goldreich and Ostrovsky [7], is to give the machine (but not the adversary) unit-cost access to a random oracle.

2.1 General program transformations

We note that there is a general upper bound on the cost of making an arbitrary algorithm data-independent:

▶ **Theorem 1.** *Any RAM program whose running time is at most $T(n)$, and refers to memory addresses bounded by $S(n)$,* [6] *may be translated to a deterministically data-independent RAM program that uses $\Theta(S(n))$ space and runs in time $\Theta(T(n)S(n))$.*

Proof. The translation is simply by brute force. At each time step, iterate over the entire finite program, and for each possible pair ⟨instruction, memory location⟩, compute the Boolean value b corresponding to whether (a) the program counter resides at the given instruction and (b) the given memory location is requested by that instruction. Then compute the result of executing the instruction, and set the resulting register and/or memory location accordingly, by "arithmetizing" the branch: $A_\ell := (b \cdot \text{new_value}) + ((1 - \text{b}) \cdot \text{A}_\ell)$. ◀

If the algorithm need only be data-oblivious, not necessarily data-independent, then the upper bound is considerably better. A long line of work on the Oblivious RAM problem [7, 15, 10, 17, 6] has produced increasingly efficient data-oblivious simulations of RAMs, achieving the following complexity bound [10]:

▶ **Theorem 2** (Kushilevitz, Lu, and Ostrovsky, 2012)**.** *Assuming one-way functions exist, there is an Oblivious RAM that requires $O(\log^2 n / \log \log n)$ amortized overhead per operation and uses $\Theta(n)$ space.*

2.2 Data-oblivious data structures

We extend the formal treatment above to data structures, by considering each operation on a data structure as an algorithm taking its current internal state and a query as input, and producing as output its result and subsequent internal state.[7]

As a simple example, we consider the case of the array: we must produce an algorithm that takes a query tuple (read/write, index, value), along with some current memory state, and returns a new memory state (and, if the operation was a read, the result of the query). For simplicity, we restrict our attention to arrays that are *operation-secret* as well as *data-secret*—i.e., the identity of the operation being performed (read or write), is labeled as secret, in addition to the values in the array.[8] (Of course, we also restrict our attention to arrays that are *index-secret*—i.e., the index i is labeled as secret—since otherwise the solution is trivial; an ordinary random-access array suffices.)

For general arrays, it is clear that we cannot hope for a nontrivial data-independent construction:

▶ Proposition 3. Any data-independent array of size n requires $\Omega(n)$ space and $\Omega(n)$ time per operation.

Proof. Immediate by an information-theoretic argument. ◀

[6] Technically, we also require T and S to be time- and space-constructible by an oblivious machine, since we must decide when to stop the simulation without inspecting the simulated data values.

[7] Here we remark that our notion of data-oblivious data structures is also distinct from that of Micciancio [12]: under our definition, the obliviousness criterion applies not to the physical representation of the data structure, but rather to the algorithms that implement its operations. Data-obliviousness is also distinct from cache-obliviousness (as in external memory models).

[8] One could also weaken the requirements so that arrays may be *operation-public*, and describe the complexity of reads and writes separately, but this does not change the situation up to constant factors, since we can just execute both a read and a write whenever either is requested.

We may consider relaxing the requirements by specifying only that the array be data-oblivious, rather than data-independent. In this case, the setting coincides with that of Oblivious RAM, and Theorem 2 gives an efficient construction.

Alternatively, we can relax the requirements not by changing the model, but by restricting the features of the data structure itself. Indeed, we now show that for many common uses of arrays—notably, stacks, queues, and priority queues—it is possible to do considerably better than the naive translation, even in the data-independent setting.

3 Stacks and queues, via composition

In some of the earliest work that considers obliviousness as an explicit goal, Pippenger and Fischer give a simulation of a multitape Turing machine by a deterministic data-independent two-tape Turing machine, running in time $O(T(n) \log T(n))$, where $T(n)$ bounds the running time of the original machine [16]. Initially, constructions such as the Pippenger-Fischer simulation were used to refine the time hierarchy theorem, as they permit a more efficient universal Turing machine simulation than the naive $\Theta(T(n)^2)$.

However, from the perspective of oblivious algorithms (even executed on a RAM), we find that the Turing machine tape is a useful data structure in its own right. As above, we will restrict our attention to the nontrivial case, that of *operation-secret* Turing machine tapes: the identity of each operation (i.e., $\{\text{read}, \text{write}\} \times \{\text{left}, \text{right}\}$), as well as each tape symbol, is deemed secret for the purpose of obliviousness. We state the following results:

▶ **Theorem 4.** *There exists a deterministic data-independent Turing machine tape with (pre-specified, public) length n, using $\Theta(n)$ space and $O(\log n)$ amortized time per operation. The tape may be taken to be either (a) toroidal, so that attempting to move off the right end wraps around to the left end, and vice versa; or (b) bounded, so that attempting to move off either end results in no head movement.*

Proof. Follows directly from the Pippenger-Fischer simulation [16]. ◀

▶ **Corollary 5.** *There exists a deterministic data-independent stack with (pre-specified, public) capacity n, using $\Theta(n)$ space and $\Theta(\log n)$ amortized time per operation.*

Proof. Follows from Theorem 4, since a stack may be implemented by writing the elements in sequence on a Turing machine tape, leaving the head at the end. ◀

Fischer et al. also showed that it is possible to simulate a Turing machine with multiple heads using a Turing machine with multiple tapes (but only one head on each tape), with only a constant factor slowdown [5]. This result enables a straightforward construction of efficient oblivious queues, by composition with the stack of the previous section:

▶ **Theorem 6.** *There exists a deterministic data-independent queue with (pre-specified, public) capacity n, using $\Theta(n)$ space and $\Theta(\log n)$ amortized time per operation.*

Proof. Given a two-headed Turing machine tape, a queue can trivially be implemented with overhead $\Theta(1)$ (and no extra space), by keeping the front of the queue at one head and the back of the queue at the other. A two-headed Turing machine tape can be implemented by a constant number of (non-oblivious) single-headed Turing machine tapes, also with linear space and only a constant factor slowdown [5, 11]. Thus, if we use the oblivious Turing machine tape of Pippenger and Fischer (Theorem 4) to implement each, the entire construction uses only $\Theta(\log n)$ amortized time per operation (and linear space). ◀

4 Stacks and queues, directly

In the previous section, we illustrated the ability to build data-oblivious data structures by composition of other data structures. We now show that one may also proceed from first principles, sometimes obtaining constructions that are much simpler and have better constant factors. Generally, efficient data-oblivious data structures share the following traits:

- Data locality – the Turing machine tape, for example, can only visit a neighborhood of size $O(k)$ within k steps.
- Self-similarity or isotropy – the "local context" at any point in the data structure must appear the same, so that the structure can operate obliviously on local data, and can obliviously shift data so that the new local context is correct.

Proceeding from these principles, we arrive at the following results:

▶ **Theorem 7.** *There exists a deterministic data-independent stack with (pre-specified, public) capacity n, requiring linear space and amortized time at most $\sim 8\lceil \lg n \rceil$ per operation.*

▶ **Theorem 8.** *There exists a deterministic data-independent queue with (pre-specified, public) capacity n, requiring linear space and amortized time at most $\sim 11.5\lceil \lg n \rceil$ per operation.*

At a high level, our constructions implement a "b-structure", which operates on b-word blocks, in terms of (i) a buffer of a constant number of b-word blocks, kept fairly close to half-full; and (b) a "$2b$-structure", defined inductively, into which blocks are pushed or pulled as the local buffer becomes too imbalanced. We defer the details to the extended version of this work. We note that the overhead of our oblivious stack is significantly better than the $24\lceil \lg n \rceil$ of the Pippenger-Fischer simulation (Corollary 5), and even yields a tighter oblivious Turing machine tape ($16\lceil \lg n \rceil$, by implementing the tape using two stacks). The constant-factor improvement in the oblivious queue is even more significant, since Theorem 6 invokes not only the Pippenger-Fischer simulation but also a simulation of a two-headed machine.

5 Priority queues

As another example of data structure composition, we now show how to build efficient oblivious priority queues (both operation-secret and operation-public), using oblivious queues as a key component. As above, we will be concerned only with data-secret priority queues, since the data-public case is trivial. However, unlike the Turing machine tape, the oblivious priority queue harbors a nontrivial distinction between operation-secret and operation-public structures, and we will consider both. In what follows, we write a to denote the capacity of the priority queue, n the number of items in the priority queue at any given time, and m the total number of operations that have been performed.

First, we note that if the priority queue is operation-secret, then the time bounds must depend only on m and a (since n depends on whether the operations performed were insertions or deletions.) We also note that in general:

▶ **Lemma 9.** *Given an operation-public priority queue such that remove-min and insert both run in amortized time $f(a, n, m)$ (where f is monotonically increasing and poly(a, n, m)), there exists an operation-secret priority queue such that both operations run in amortized time $O(f(m, m, m))$.*

Proof. By performing dummy operations; we defer the details to the extended version. ◀

▶ **Theorem 10.** *There exists a deterministic data-independent, operation-secret priority queue, using* $\Theta(\min(a, m))$ *space and* $\Theta(\log^2(\min(a, m)))$ *amortized time per operation.*

Proof. For simplicity, we assume the capacity of the priority queue is a power of two, $a = 2^k$, and that all elements are distinct. We use a hierarchical series of k oblivious queues (Section 8), as in standard Oblivious RAM constructions; the i^{th} queue has capacity 2^{i+1}.

We maintain the following invariant: at the beginning of any operation, queue i contains up to 2^i items in sorted order. The operations are then implemented as follows:[9]

1. To remove the minimum: first, find the minimum by examining the front of each queue.[10] Then, for each queue, if its front is the minimum, pop it; otherwise, do nothing. (Since all elements are distinct, only one of the queues will actually be popped.)
2. To insert an item: first, let l be the deepest level such that 2^l divides m (where m, as above, is the number of operations performed so far). Create a temporary buffer queue B of size 2^{l+1}, holding only the new element. Then, for each level $i \in (0, \ldots, l)$, merge the contents of queue i into B. This can be done using the standard merge algorithm: pop whichever of the two queues currently yields the smaller element, accumulating the results in a new buffer B', and finally replace B with B'.

By the properties of the oblivious queue, operation (1) takes amortized time $\Theta(\log(2^i))$ on level i, and thus the total running time is $\sum_{i=0}^{l-1} \Theta(i) = \Theta(l^2) = \Theta(\log^2 \min(a, m))$ (where l is the index of the largest occupied level). On the other hand, operation (2) merges a queue of size $\Theta(2^i)$ into its successor (taking time $\Theta(2^i \log 2^i)$) once after every 2^i operations, and thus the amortized cost of each operation is $\sum_{i=0}^{l}(1/2^i)(\Theta(2^i \log 2^i)) = \Theta(l^2) = \Theta(\log^2 \min(a, m))$ also. ◀

▶ **Theorem 11.** *There exists a deterministic data-independent operation-public priority queue, using* $\Theta(n)$ *space and* $\Theta(\log^2 n)$ *amortized time per operation, where* n *is the size of the priority queue prior to each operation.*

Proof. Similar to the proof of Theorem 10; we defer the details to the extended version. ◀

Theorems 10 and 11 establish efficient constructions of oblivious priority queues in both the operation-secret and the operation-public models. Evidently, for series of operations in which $m = O(n)$ (e.g., inserting $2m/3$ items followed by removing $m/3$ items), the performance in the operation-secret case is no worse asymptotically than in the operation-public case; since the operation-secret queue cannot reveal the pattern of operations, in a sense, this is the best we can hope for, without also obtaining a faster operation-public queue. Thus, the distinction between operation-public and operation-secret priority queues turns out not to matter, at least in the context of our best known constructions.

The constructions above compare favorably with the generic solution in the data-oblivious setting: i.e., representing the priority queues as min-heaps, and using Oblivious RAM to serve as the underlying array. In this case, using the Oblivious RAM of Kushilevitz et al. [10], we would spend $\Theta(\log^3 n / \log\log n)$ time, and $\Theta(\log^2 n)$ sequential communication rounds, per operation (and we would incur either a large constant factor, if the Oblivious RAM uses the AKS sorting network; or a moderate constant factor, with some small probability of error, if the Oblivious RAM uses Goodrich's randomized Shellsort). In contrast, our constructions

[9] In fact, since the priority queue is operation-secret, of course, we simulate both (1) and (2) when either operation is requested.

[10] Strictly speaking, this is not an operation of the queue interface above, but it is a straightforward extension since the pop operation is not destructive.

are not just data-oblivious but data-independent; they operate deterministically, and costs only $\Theta(\log^2 n)$ time per operation, with a very small constant factor.

We note that in recent independent work, Toft [18] has also described a data-oblivious priority queue with the same asymptotic bounds, albeit by a different method. The constructions of this section are much simpler, however, due to the composition of data-oblivious primitives.

6 Data compaction and the partition problem

Goodrich [8] describes a problem called *data compaction*: given an array of length n in which r of the elements are marked as "distinguished", construct a new array of size $O(r)$ containing only the distinguished items. If the resulting array is required to be of size exactly r, the compaction is said to be *tight*; otherwise it is *loose*. If the items are also required to be in their original order, the compaction is additionally said to be *order-preserving*. Goodrich proves the following:

▶ **Theorem 12** (Goodrich, 2011). *There exists a deterministically data-independent algorithm for tight order-preserving data compaction running in time $\Theta(n \log n)$.*

▶ **Theorem 13** (Goodrich, 2011). *There is a data-oblivious algorithm that runs in time $O(n \log^* n)$ and achieves loose data compaction with high probability when $r < n/4$.*

However, for the next section, we will need fast tight compaction. To that end, we first rephrase the problem of tight compaction in terms of the *partition problem*: given an array of n items each tagged with a single bit, separate the items so that all those tagged with a 0 appear to the left of those tagged with a 1.[11]

Now, we will show:

▶ **Theorem 14.** *There is a data-independent algorithm that runs in time $\Theta(n \log \log n)$ and achieves tight data compaction (i.e., solves the partition problem) with high probability.*

Proof. First, we note that we can easily count which tag (0 or 1) constitutes a majority, and obliviously decide whether to invert the tags (and reverse the array) based on that information. Thus, it suffices to give an algorithm to extract a constant fraction of whichever tag constitutes a majority (placing the extracted items at either the beginning or the end of the array, as appropriate), since we may then recur on the rest of the array. At any stage in this recursion, let n denote the size of the entire array, and n' denote the size of the subarray being partitioned at this point.

Now, without loss of generality, suppose the subarray contains more zeroes than ones. Let A denote the leftmost $n'/3$ cells, and B the rightmost $2n'/3$. Let s, the *bucket size*, be $(\log n)^k$ for an arbitrary constant $k > 1$. Then:

1. For each cell a_i in A, pick c cells b_i uniformly at random from B (for some constant c to be determined), and, if b_i contains a 0, swap a_i with b_i.
2. Divide A into $n'/3s$ buckets of s cells each. Partition each bucket using any $\Theta(s \log s)$ algorithm (e.g., Theorem 12).

[11] To show the equivalence, we note that in order to solve the data compaction problem, we can simply mark the distinguished cells with 0, run a partition algorithm, and return the appropriate prefix of the original array. Conversely, to solve the partition problem, we run data compaction twice: once with the 0 elements marked as distinguished, and once with the array in reverse order and the 1 elements marked as distinguished. Then, iterate over the two resulting arrays in parallel, and at each index, copy from whichever array has a distinguished item.

3. Extract and concatenate all of the first halves of the buckets (i.e., $n'/3s$ half-buckets, each of size $s/2$, for a total of $n'/6$ items).

Evidently, this procedure performs a total of $cn'/3+(n'/3s)(s\log s)+n'/6 = \Theta(n'\log s) = \Theta(n'\log\log n)$ operations, giving a running time of $\Theta(n'\log\log n)$ for a subarray of size n'. Since $T(n') = T(5n'/6) + \Theta(n'\log\log n)$ (and noting that we may stop the recursion at $n' = O(s)$, solving the base case directly by the algorithm of Theorem 12 with $T(s) = O(s\log s) = s\log\log n$), we have an overall running time of $T(n) = \Theta(n\log\log n)$.

For correctness, we now claim that for any given subarray of size n', after the final step of the iteration, the first $n'/6$ cells of A are all 0 with high probability. To show this, we first note that since zeroes constitute a majority of the array, B (the rightmost $2n'/3$ cells) must contain at least $n'/6$ zeroes at all times. Thus, in step 1, a_i will become zero with probability at least $(n'/6)/(2n'/3) = 1/4$ on every potential swap, independently of any other events. Since c swaps are performed, the probability that a_i will become zero after the entire step is at least $1-(3/4)^c$; we choose c so that this success probability is bounded away from $1/2$. Thus, by Hoeffding's inequality (and a union bound), the probability that there are more than $s/2$ ones in any bucket (and, thus, the probability that we fail to extract $n'/6$ zeroes after partitioning and splitting) is at most $(n'/3s)e^{-2(2/3-1/2)^2 s} = (n'/6)e^{-\Theta(s)}$. Even after executing this procedure at every level of the recursion (as n' decreases from n down to s), again by a union bound, the overall probability of error is at most $\sum_{n'}(n'/6)e^{-\Theta(s)} = ne^{-\Theta(s)} = n^{-\Theta(\lg^{k-1} n)}$. ◀

Crucially, because this partitioning algorithm is based on swaps (and is data-independent, not just data-oblivious), we can also use it to "un-partition", or intersperse, items:

▶ **Theorem 15.** *Given an array of r items and an array of r binary tags, there is a data-independent algorithm that runs in time $\Theta(r\log\log n)$ and, with probability $1-1/n^{\omega(1)}$, produces a permutation of the array such that every element that was originally in the left half now occupies a location that was tagged with 0. If one-way functions exist, this algorithm can use $O(n)$ words of memory; otherwise, it uses $O(n\log\log n)$ words.*

Proof. Run the algorithm of Theorem 14 on the tag array, and record a "trace" consisting of each pair of indices that it decided to swap, accompanied by a (secret) bit indicating whether those items were actually swapped. (These indices may also include empty scratch cells, not initially containing either a 0 or a 1 item.) Then, run the trace in reverse on the array of actual items.

The space used by the trace is $O(r\log\log n)$ bits, which requires $o(r)$ words; plus $O(r\log\log n)$ pairs of indices, which requires $O(r\log\log n)$ words. However, assuming one-way functions (and hence PRFs), the index pairs can be taken to be the output of a PRF on a short seed and a time step, in which case the entire assembly requires only linear space. ◀

▶ **Lemma 16.** *Given two arrays of size r, each permuted according to a distribution that is computationally indistinguishable from uniform, there is a data-independent algorithm that runs in time $\Theta(r\log\log n)$ and results in a permutation that is computationally indistinguishable from uniform on all $2r$ elements.*

Proof. Follows from Theorem 15. ◀

7 Offline Oblivious RAM

We now show that the data compaction problem is closely related to the question of data-oblivious arrays (i.e., Oblivious RAM). Intuitively, the hierarchical Oblivious RAM

constructions of Goldreich and Ostrovsky [7] and of Kushilevitz et al. [10] must remember (and hide from the adversary) two things: (i) the "level", or epoch, in which each item resides; and (ii) the permutation of the items within each epoch. As a result, they must use techniques such as oblivious cuckoo hashing via sorting [10] or logarithmic-sized buckets [7] in order to make accesses to a level look the same whether an item is present or not. However, we can separate these two issues by considering a variant of the problem in which the RAM need not remember when an item was last accessed—perhaps because the sequence of memory addresses is known in advance, and can be preprocessed before the queries are made.

More precisely, we define a *primed array* to be a data structure that has the same interface as an array, except that each request (read or write) is accompanied by the (logarithm of the) time since the requested cell was last accessed. We note that a primed array generalizes an "offline" array with preprocessed queries, since the preprocessor may annotate each query with the preceding access time to that cell. Now, we can show:

▶ **Theorem 17.** *Assuming one-way functions exist, there is a (computationally) data-oblivious primed array with expected amortized overhead $O((p(n)/n)\log n)$ per operation, where $p(n)$ is the time required for oblivious partitioning (or data compaction) of n items.*

Proof. As in other hierarchical Oblivious RAM constructions, we maintain a series of levels of increasing power-of-two sizes, in which level i contains between zero and $r = 2^i$ items, and is "shuffled" into the next larger level (in a sense specified below) and after every epoch of length r. Here, level i consists of a dictionary D_i of size $\Theta(r)$, mapping each of $2r$ keys to a data item, where each key is either an element of $\{1, \ldots, n\}$ or one of the dummy values $\{\text{dummy}_1^{t_i}, \ldots, \text{dummy}_r^{t_i}\}$, where t_i is the time of the most recent shuffle at level i (superscripts elided below for clarity). (D_i may be implemented by a cuckoo hash table, or even a standard linked-list-based hash table, since we will not depend on its hiding the access pattern—only on the fact that it implements a dictionary in expected amortized constant time.) Further, during any fixed epoch, level i will have an associated injective PRF $\psi_i^{t_i}$ (again, we drop the time index for clarity, and just write ψ_i). For each logical address x present at level i, $D_i(\psi_i(x))$ is the item stored at x; while for each dummy index dummy_s, the key $\psi_i(\text{dummy}_s)$ is present in D_i (and is mapped to a dummy item). Now, upon receiving a query, we find out how long ago the cell was accessed. This tells us in which level it resides (since, as in standard Oblivious RAM constructions, we will "promote" an item to the top level whenever it is accessed). Suppose it resides at level i. We then access a real item in that level only, by querying D_i at $\psi_i(x)$, and query for a unique dummy item in all other levels (e.g., $\psi_j(\text{dummy}_{t \bmod 2^j})$).

When it is time for a level to be merged into the next level down, we first extract (via oblivious partitioning) the items from both levels that have not yet been accessed during this epoch—possibly including some real items and some dummy items (which we overwrite with new dummy indices appropriate to the new, merged level). Then, we randomly intersperse these two lists of items (Section 6). Finally, we fill the resulting $2r$ items into the new level's hash table, keyed by the values of a new PRF (evaluated on either the items' addresses, for real items, or on their new dummy indices).

During this process, the adversary sees the values of the new PRF on all addresses remaining in both levels, appearing in the order they were in after the partition operations. Since these addresses are all distinct, their images under the PRF are indistinguishable from an independently uniform set of size $2r$. Hence the information obtained by the adversary is completely described by the correspondence between these values and the PRF image value queried on each level's dictionary when an item is retrieved (recall that these can match only once, since as soon as an item is found, it is promoted to the top level; while dummy

indices are used only once per epoch). This reveals only the order that an item had when it was initially inserted into its current level during a merge. Thus, if we assume inductively that the ordering of unvisited items in each of the higher levels was indistinguishable from a uniform permutation (of their order of insertion), then it follows from Lemma 16 that the resulting ordering in the current level is also indistinguishable from a uniform permutation, and so is independent of the address request sequence, as desired. ◀

▶ **Corollary 18.** *Assuming one-way functions exist, there is an offline Oblivious RAM with expected amortized overhead* $O(\log n \log\log n)$ *per operation.*

Proof. Immediate from Theorems 14 and 17. ◀

8 Lower bounds via communication complexity

For most problems, it seems very difficult to establish lower bounds on oblivious algorithms, since any nontrivial slowdown due to obliviousness would imply nontrivial uniform circuit lower bounds. For oblivious data structures, however, this is not the case. In fact, by viewing the Oblivious RAM as an oblivious data structure, we immediately obtain a communication complexity lower bound that generalizes both that of Pippenger and Fischer [16], for oblivious Turing machine tapes, and that of Goldreich and Ostrovsky [7], for Oblivious RAMs (we defer the proof to the extended version of this work):

▶ **Theorem 19.** *Any data-oblivious Turing machine tape of length* n *requires* $\Omega(\log n)$ *expected amortized time per operation.*

Now Theorem 19 immediately entails Ostrovsky's lower bound for Oblivious RAM (originally proven by a combinatorial argument [7]):

▶ **Corollary 20.** *Any Oblivious RAM (i.e., any data-oblivious array) with* n *memory words requires* $\Omega(\log n)$ *expected amortized time per operation.*

Proof. Use the array to implement a Turing machine tape and invoke Theorem 19. ◀

Acknowledgements. We thank Dan Boneh and Valeria Nikolaenko for helpful discussions, and the anonymous reviewers for their comments. This work was supported by DARPA PROCEED, under contract #N00014-11-1-0276-P00002, the National Science Foundation, and the Air Force Office of Scientific Research. During this work, Joe Zimmerman has been further supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program (2012-2013), and by the National Science Foundation (NSF) through the National Science Foundation Graduate Research Fellowship Program (GRFP) (2013-2014).

References

1 Paul Beame and Widad Machmouchi. Making branching programs oblivious requires superlogarithmic overhead. In *IEEE Conference on Computational Complexity*, pages 12–22. IEEE Computer Society, 2011.

2 Paul Beame and Widad Machmouchi. Making RAMs oblivious requires superlogarithmic overhead. *ECCC*, 2011.

3 Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure Oblivious RAM without random oracles. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 144–163. Springer, 2011.

4 David Eppstein, Michael T. Goodrich, and Roberto Tamassia. Privacy-preserving data-oblivious geometric algorithms for geographic data. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 13–22, New York, NY, USA, 2010. ACM.

5 Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Real-time simulation of multihead tape units. *J. ACM*, 19(4):590–607, October 1972.

6 Craig Gentry, Kenny A. Goldman, Shai Halevi, Charanjit S. Jutla, Mariana Raykova, and Daniel Wichs. Optimizing ORAM and using it efficiently for secure computation. In Emiliano De Cristofaro and Matthew Wright, editors, *Privacy Enhancing Technologies*, volume 7981 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

7 Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, May 1996.

8 Michael T. Goodrich. Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data. In *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures*, SPAA '11, pages 379–388, New York, NY, USA, 2011. ACM.

9 F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape turing machines. *J. ACM*, 13(4):533–546, October 1966.

10 Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. In Yuval Rabani, editor, *SODA*, pages 143–156. SIAM, 2012.

11 Benton L. Leong and Joel I. Seiferas. New real-time simulations of multihead tape units. *J. ACM*, 28(1):166–180, January 1981.

12 Daniele Micciancio. Oblivious data structures: applications to cryptography. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 456–464, New York, NY, USA, 1997. ACM.

13 John C. Mitchell, Rahul Sharma, Deian Stefan, and Joe Zimmerman. Information-flow control for programming on encrypted data. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 45–60. IEEE, June 2012.

14 Andrew C. Myers and Barbara Liskov. Protecting privacy using the decentralized label model. *ACM Trans. Softw. Eng. Methodol.*, 9(4):410–442, October 2000.

15 Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO'10, pages 502–519, Berlin, Heidelberg, 2010. Springer-Verlag.

16 Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, April 1979.

17 Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious RAM protocol. *IACR Cryptology ePrint Archive*, 2013:280, 2013.

18 Tomas Toft. Secure data structures based on multi-party computation. In Cyril Gavoille and Pierre Fraigniaud, editors, *PODC*, pages 291–292. ACM, 2011.

Higher randomness and forcing with closed sets

Benoit Monin

Université Paris Diderot, LIAFA, Paris, France
benoit.monin@computability.fr

Abstract

Kechris showed in [8] that there exists a largest Π^1_1 set of measure 0. An explicit construction of this largest Π^1_1 nullset has later been given in [6]. Due to its universal nature, it was conjectured by many that this nullset has a high Borel rank (the question is explicitely mentioned in [3] and [16]). In this paper, we refute this conjecture and show that this nullset is merely $\mathbf{\Sigma^0_3}$. Together with a result of Liang Yu, our result also implies that the exact Borel complexity of this set is $\mathbf{\Sigma^0_3}$.

To do this proof, we develop the machinery of effective randomness and effective Solovay genericity, investigating the connections between those notions and effective domination properties.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Effective descriptive set theory, Higher computability, Effective randomness, Genericity

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.566

1 Introduction

We will study in this paper the notion of forcing with closed sets of positive measure and several variants of it. This forcing is generally attributed to Solovay, who used it in [15] to produce a model of $ZF + DC$ in which all sets of reals are Lebesgue measurable. Stronger and stronger genericity for this forcing coincides with stronger and stronger notions of randomness. It is actually possible to express most of the randomness definitions that have been made over the years by forcing over closed sets of positive measure.

In the first section we give a brief overview of the part of algorithmic randomness that we need in the paper. In the second section we make a modification to the usual definition of effective Solovay genericity directly inspired by a notion introduced by Jockusch in [7] about effective genericity for Cohen forcing. This new definition will reveal itself to be interesting for its connections with effective domination properties. In the third section we will give a quick description of what we need of higher computability theory and higher randomness to approach the last section. Finally in the last section we give higher analogues of the Solovay genericity notions studied in section two, and we show again their connections with randomness and higher effective domination properties. This will allow us to conclude with the Borel complexity of the largest Π^1_1 nullset.

2 General Background

In this paper, we will work in the space of infinite sequences of 0's and 1's, called the Cantor space, denoted by 2^ω. We will call **strings** finite sequences of 0's and 1's, **sequences** elements of the Cantor space and **sets** the sets of sequences. For a string σ, we will denote the set of sequences extending σ by $[\sigma]$.

The set of integers W_e will denote the domain of the computable function Φ_e, and $[W_e]$ will denote $\bigcup_{\sigma \in W_e}[\sigma]$, where W_e is seen as a set of strings. We will denote by $\langle , \rangle$ a fixed

© Benoit Monin;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 566–577
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

computable pairing function from $\omega \times \omega$ to ω.

We will consider computable functionals (computable functions using sequences as oracles) as functions from the Cantor space to the Baire space. Then a computable functional Φ is considered define on $X \in 2^\omega$ if $\forall n\ \Phi^X(n)\downarrow$ and we denote by $\operatorname{dom}\Phi$ the set $\{X \mid \forall n\ \Phi^X(n)\downarrow\}$. We say that a function f is computable relative to X or X-computable if there is a computable functional defined on X such that $\Phi^X = f$.

The topology on Cantor space is generated by the basic intervals $[\sigma] = \{X \in 2^\omega \mid X \succ \sigma\}$ for σ a string. For $A \subseteq 2^\omega$ Lebesgue-measurable, $\lambda(A)$ will denote the Lebesgue measure of A, which is the unique Borel measure such that $\lambda([\sigma]) = 2^{-|\sigma|}$ for all strings σ.

2.1 About the arithmetical complexity of sets

In the Cantor space, open sets can be described as countable unions of strings. We call an open set **effective** if it can be described as the union of a computably enumerable set of strings, i.e. if it is equal to $[W_e]$ for some e. Such a set is said to be Σ^0_1. On the other hand, when it is open but not necessarily effectively open, the set is said to be $\mathbf{\Sigma^0_1}$. However, a non-effective open set is always effective relatively to some oracle. If X is such an oracle, we say that the set is $\Sigma^0_1(X)$. A closed set is called effective if its complement is an effective open set, in which case we say that the closed set is a Π^0_1 set. We can then continue to describe the effective Borel sets through the arithmetical hierarchy as effective unions of effective Borel set of lower complexity and as their complements. So a Σ^0_{n+1} set will be an effective union of Π^0_n sets, and a Π^0_{n+1} set will be the complement of a Σ^0_{n+1} set. For example, a set A is Σ^0_4 if we have an integer e such that $A = \bigcup_{m_1 \in W_e} \bigcap_{m_2 \in W_{m_1}} \bigcup_{m_3 \in W_{m_2}} [W_{m_3}]^c$.

We have a canonical surjection from integers to Σ^0_1 sets (The one which associates to e the computably enumerable set $[W_e]$), but also from integers to Σ^0_n sets for a fixed n. In the above example, with $n = 4$ the integer e is associated to the Σ^0_4 set A. In this context e will be called an **index** for the set A.

Also for a computably enumerable set of integers W, we denote by $W[t]$ the enumeration of W up to stage t. We extend this definition to effective open sets: if $O = [W]$, then $O[t] = [W[t]]$. Similarly, if $F = O^c$, $F[t] = O[t]^c$.

2.2 About algorithmic randomness

In 1966, Martin-Löf gave in [10] a definition capturing elements of the Cantor space that can be considered 'random'. Many nice properties of the Martin-Löf random sequences make this notion of randomness one of the most interesting and one of the most studied.

Intuitively a random sequence should not have any atypical property. A property is here considered atypical if the set of sequences having it is of measure 0. It also makes sense to consider only properties which can be described in some effective way (because any X has the property of being in the set $\{X\}$ and thus nothing would be random).

▶ **Definition 1.** An intersection of measurable sets $\bigcap_n A_n$ is said to be **effectively of measure** 0 if the function which to n associates the measure of A_n is bounded by a decreasing computable function whose limit is 0. A **Martin-Löf** test is a Π^0_2 set $\bigcap_n O_n$ effectively of measure 0. We say that $X \in 2^\omega$ is **Martin-Löf random** if it is in no Martin-Löf test.

One can iterate this idea by considering Π^0_n sets effectively of measure 0 for any $n \geq 2$. Martin-Löf randomness is also called **1-randomness**, the use of Π^0_3 sets effectively of

measure 0 gives us **2-randomness**, Π^0_4 sets give us **3-randomness**, and so on. The requirement for a Martin-Löf test to be effectively of measure 0 is important and leads to very nice properties. In particular there exists a universal Martin-Löf test, i.e. a test containing all the others (see [10]). This is not the case anymore if we drop the 'effectively of measure 0' condition. Instead we get a notion known as weak-2-randomness.

▶ **Definition 2.** We say that $X \in 2^\omega$ is **weakly-2-random** if it is in no Π^0_2 nullset.

As a randomness notion, weak-2-randomness is a strictly stronger than 1-randomness, but is strictly weaker than 2-randomness (see [13] section 3.6).

3 Solovay genericity and its variants

Cohen introduced in [4] the general technique of forcing by forcing with all dense open sets of the Cantor space (with the usual topology) in a countable model of ZFC. The most basic effective version of this would be to say that X is generic if it belongs to all dense Σ^0_1 sets, a notion introduced by Kurtz in [9]. Jockusch introduced and studied in [7] a slightly different notion.

▶ **Definition 3** (Kurtz, Jockusch)**.** We say that X is **weakly-1-generic** if it belongs to all dense Σ^0_1 sets. We say that X is **1-generic** if for any Σ^0_1 set U, either X belongs to U or X belongs to some other Σ^0_1 set U' disjoint from U.

We will apply Jockusch's idea behind 1-genericity to forcing with Π^0_1 sets. First note that by definition, the weakly-2-randoms are exactly the sequences which are in all Σ^0_2 sets of measure 1. If we consider the topology generated by Π^0_1 sets of positive measure, because Σ^0_2 sets of measure 1 are then dense open sets for this topology, we also get in some sense a genericity notion.

3.1 Forcing with Π^0_1 sets

Adding a measure requirement to the definition of genericity will always link us to randomness. We study what happens if we drop the measure requirement and if we consider instead the Σ^0_2 sets which are dense for the topology generated by the Π^0_1 sets, i.e. the Σ^0_2 sets which intersect all non-empty Π^0_1 set. It is clear that the Cantor space with this topology is a Baire space, i.e. has the property that an intersection of dense open sets is dense. This directly comes from the fact that a decreasing intersection of non-empty closed sets is non-empty. This justifies the following definition:

▶ **Definition 4.** Let $\{G_i\}_{i\in\omega}$ be the collection of all Σ^0_2 sets which intersect all the Π^0_1 sets. We say that X is **weakly-Π^0_1-generic** if it belongs to $\bigcap_i G_i$.

As the next proposition shows, weak-Π^0_1-genericity has nothing to do with randomness.

▶ **Proposition 5.** *No weakly-Π^0_1-generic sequence is 2-random.*

Proof. We construct uniformly in n a Σ^0_2 set intersecting all Π^0_1 sets and with measure smaller than 2^{-n}. Let $\{F_e\}_{e\in\omega}$ be an enumeration of the Π^0_1 sets. For each e we initialize σ_e to the first string (using lexicographic order) of length $n+e+1$. Our Σ^0_2 set will consist of a computably enumerable set A of indices of Π^0_1 sets. We now describe the algorithm to enumerate elements of A: At stage t, for each substage $e < t$ in increasing order, if the index of $F_e \cap [\sigma_e]$ has not been enumerated yet into A, then enumerate it. After that,

if $(F_e \cap [\sigma_e])[t] = \emptyset$ then reset σ_e to be the string of length $n+e+1$ following σ_e in the lexicographic order. If σ_e is already the last such string, leave it unchanged.

Let us prove that the measure of the Σ^0_2 set represented by A is smaller than 2^{-n}. For each e, if $F_e \cap [\sigma_e] = \emptyset$ then by compactness $(F_e \cap [\sigma_e])[t] = \emptyset$ for some t. Thus at most one string σ_e of length $n+e+1$ such that $F_e \cap [\sigma_e] \neq \emptyset$ has been enumerated into A, and the measure of A is bounded by $\sum_e 2^{-n-e-1} \leq 2^{-n}$. Now our Σ^0_2 set is dense because if F_e is not empty then there exists a string σ_e of length $n+e+1$ such that $F_e \cap [\sigma_e]$ is not empty and then A will intersect F_e.

From this we can then construct a Π^0_3 set effectively of measure 0 and containing all the weakly-Π^0_1-generic sequences. ◀

Following Jockusch's 1-genericity idea we now define Π^0_1-genericity:

▶ **Definition 6.** A sequence X is **Π^0_1-generic** if for all Σ^0_2 sets G, either X is in G or there is a Π^0_1 set F disjoint from G such that X is in F.

We now establish a simple but surprising connection with computability theory, which appears to be previously unknown. We say that a sequence X is **computably dominated** if for every total function $f : \omega \to \omega$, computable relative to X, there exists a total computable function g such that g dominates f (i.e. $\forall n\ f(n) \leq g(n)$).

▶ **Proposition 7.** *A set X is Π^0_1-generic iff it is computably dominated.*

Proof. Suppose X is computably dominated and take any Σ^0_2 set $\bigcup_n F_n$. Suppose that X belongs to its complement, a Π^0_2 set $\bigcap_n O_n$. Let us define the X-computable function $f : \omega \to \omega$ which to n associates the smallest t so that $X \in O_n[t]$. As X is computably dominated, there is a computable function g which dominates f. Then $X \in \bigcap_n O_n[g(n)]$, an effectively closed set disjoint from $\bigcup_n F_n$.

Conversely suppose that X is Π^0_1-generic and consider a functional Φ, defined on X. We have that $\operatorname{dom}\Phi = \{X \mid \forall n\ \Phi^X(n)\downarrow\}$ is a Π^0_2 set containing X. But then as X is Π^0_1-generic, it is contained in a Π^0_1 set F contained in the domain of Φ. Let us now build[1] a computable function f such that $\forall X \in F\ \Phi^X < f$. To compute the value of $f(n)$ we find the smallest pair $\langle m, t\rangle$ such that for all strings σ of size m with $[\sigma] \subseteq F[t]$, the functional Φ halts on n in less than t steps with σ as an oracle (considering that if Φ needs to use bits of the oracle at positions bigger than $|\sigma|$, it does not halt). Then we set $f(n)$ to the sum of all those values plus one. All we need to show is that f is total. Fix n and let us prove there is a m so that for all $X \in F$ we have $\Phi^{X\restriction_m}(n)\downarrow$. Suppose not, then for all m there is $X \in F$ with $\Phi^{\sigma_m}(n)\uparrow$ where $\sigma_m = X\restriction_m$. As $\{\sigma_m\}_{m\in\omega}$ is infinite it has at least one limit sequence Y and as F is closed we have $Y \in F$. Also as $\Phi^{Y\restriction_m}(n)\uparrow$ for all m we have that Φ is not defined on Y which contradicts the hypothesis. Thus for some t we have that $F[t]$ is covered by a finite union $\bigcup_{i\leq k}[\sigma_i]$ such that $\Phi^{\sigma_i}(n)\downarrow$. It follows that for some t and some m we have that $\Phi^\sigma(n)$ halts in less than t steps for all strings σ of size m such that $[\sigma] \subseteq F[t]$. ◀

A direct computation shows that the set of computably dominated sequences is Π^0_4. The above proposition lowers down the Borel complexity to $\mathbf{\Pi^0_3}$: if for every set A we denote by A° the interior of A for the topology generated by Π^0_1 sets, i.e. the union of all Π^0_1 sets included in A, then the set of computably dominated sequences is the intersection over all

[1] One can also directly deduce the existence of such a function f using the fact that the supremum of a computable function, over an effectively compact set, is right-ce.

the Π^0_2 sets P, of $P^\circ \cup P^c$. We now give a lower bound on the Borel complexity of the computably dominated sequences, however we do not know if it can be $\mathbf{\Sigma^0_3}$.

▶ **Proposition 8.** *The set of computably dominated sequences is neither* $\mathbf{\Sigma^0_2}$ *nor* $\mathbf{\Pi^0_2}$.

Proof. Let us show that it is not $\mathbf{\Pi^0_2}$. First note that for any $\mathbf{\Pi^0_2}$ set A, if A is dense (for the usual topology) in some $[\sigma]$ then it contains a weakly-1-generic sequence as defined by Kurtz. Indeed, the intersection of $A \cap [\sigma]$ with all dense Σ^0_1 sets will not be empty and will then contain weakly-1-generic sequences. But by a result of computability theory (see [9]), no weakly-1-generic is computably dominated. Thus a $\mathbf{\Pi^0_2}$ set containing only computably dominated sequences is nowhere dense. But as the set of computably dominated sequences is dense, being closed under finite change of prefixes, such a $\mathbf{\Pi^0_2}$ set cannot contain all of them.

To show that it is not $\mathbf{\Sigma^0_2}$, we adapt a technique that Liang Yu exposed in [16]. Suppose that the set of computably dominated sequences is described as $\bigcup_n F_n$ with each F_n closed. For each n let $B_n = \bigcup\{T \mid T \cap F_n = \emptyset \text{ and } T \text{ is a } \Pi^0_1 \text{ set with no computable member}\}$. Let us prove that the set B_n intersects any non-empty Π^0_1 set with no computable members. Take any non-empty Π^0_1 set G with no computable members. By a classical result of computability theory (see [13] proposition 1.5.12 combined with fact 1.8.36) G contains a non-computably dominated sequence. Thus G contains a sequence X which is not in F_n. Then as F_n is closed there is a string σ such that $X \in G \cap [\sigma]$ but $G \cap [\sigma] \cap F_n = \emptyset$. Thus $G \cap [\sigma]$ is a non-empty Π^0_1 set with no computable sequence, intersecting G and disjoint from F_n. Consequently we have $B_n \cap G \neq \emptyset$ and then each B_n is dense for the topology generated by Π^0_1 sets with no computable member. It follows that $\bigcap_n B_n$ is also dense for this topology. From Proposition 7 the set of computably dominated sequences is also dense for this topology. Then there is a computably dominated sequence in $\bigcap_n B_n$. But we also have by design of the B_n that $\bigcap_n B_n \cap \bigcup_n F_n = \emptyset$, which contradicts the fact that $\bigcup_n F_n$ contains all computably dominated sequences. ◀

3.2 Forcing with Π^0_1 sets of positive measure

We now introduce a notion of genericity which is a measure-theoretic variation of Π^0_1-genericity defined in the previous section. The notion will be interesting for its counterpart in Higher computability. Let us now come back to the topology generated by Π^0_1 sets of positive measure. To obtain weak-2-randomness we consider only Σ^0_2 sets of measure 1. We now consider all Σ^0_2 sets which intersect with positive measure every Π^0_1 set of positive measure.

▶ **Definition 9.** Let $\{G_i\}_{i\in\omega}$ be the collection of all Σ^0_2 sets A such that for any Π^0_1 set F of positive measure we have $\lambda(A \cap F) > 0$. Then we say that X is **weakly-Π^0_1-Solovay-generic** if it belongs to $\bigcap_i G_i$.

▶ **Definition 10.** We say that X is **Π^0_1-Solovay-generic** if for any Σ^0_2 set A, either X is in it or there exists a Π^0_1 set F of positive measure and disjoint from A such that X is in it.

▶ **Proposition 11.** *A set X is Π^0_1-Solovay-generic iff it is weakly-2-random and computably dominated.*

Proof. Suppose that X is weakly-2-random and computably dominated. Take any Σ^0_2 set and suppose that X does not belong to it. By Proposition 7, as X is computably dominated, we have that X belongs to some Π^0_1 set disjoint from the Σ^0_2 set. Also as X is weakly-2-random this Π^0_1 set has positive measure.

Conversely, suppose that X is Π^0_1-Solovay-generic. In particular it is weakly-2-random and Π^0_1-generic. Then by Proposition 7 we have that it is computably dominated. ◀

3.3 A separation for weak and non weak-genericity

We will now prove that weak-genericity is not enough to obtain computable domination. For this we shall adapt a proof of a theorem in [1] saying that for any function f, there is a weakly-2-random X and an X-computable function g not dominated by f. Here we want weak-Π^0_1-Solovay-genericity instead of weak-2-randomness.

▶ **Proposition 12.** *For any function $f : \omega \to \omega$ there is an X weakly-Π^0_1-Solovay-generic computing a function $g : \omega \to \omega$ which is above f infinitely often.*

The reader can see [12] for a proof of proposition 12, that we skip here, due to its length. Using Proposition 12, we have some weakly-Π^0_1-Solovay-generics which are not computably dominated and so not Π^0_1-Solovay-generic. One can prove that weakly-Π^0_1-Solovay-genericity implies weakly-Π^0_1-genericity by showing that any Σ^0_2 set intersecting all the Π^0_1 sets also intersects with positive measure all Π^0_1 sets of positive measure. Take any Σ^0_2 set intersecting all the Π^0_1 sets. Take now a set F of positive measure and consider the Σ^0_2 set $\bigcup_n F_n$ of Martin-Löf randoms (the complement of the universal Martin-Löf test). As it has measure 1, there is some F_n such that $F \cap F_n$ has positive measure. But by hypothesis our Σ^0_2 set intersects $F \cap F_n$. The intersection contains only Martin-Löf random sequences and thus is necessarily of positive measure. Thus there is also some weakly- Pi^0_1-generics which are not Π^0_1-generics.

4 Background on higher computability and higher randomness

We now give a few definitions of higher computability and higher randomness. The Turing reductions are replaced by hyperarithmetical reductions. One intuitive way to understand a hyperarithmetical computation is to think of a standard Turing computation, but with an infinite-time Turing machine. For those machines the computational time is not an integer anymore, but an ordinal. Tapes are infinite and pre-filled with 0's, at a successor stage everything happens as in a regular Turing machine. At a limit stage, the machine changes to a special 'limit' state, the head comes back to the first cell of the first tape and if the value of a cell of a tape does not converge, it is reset to 0 (otherwise it is set to the limit of its previous values). The rest works as usual.

For example, we can build the ordinal time Turing machine which on a tape, at finite computation time $t = \langle s, e\rangle$ write 1 on the cell number e of this tape if the program number e halts in less than s steps. At ordinal time ω we then have the halting problem on this tape. Then stages $\omega + n$ can be used to compute what one could compute with the halting problem. This can be iterated to compute anything that could be computed in a finite jump. But we can even go beyond a finite jump and continue through the ordinal jumps. To formalize this properly we need to fix the notion of notation for computable ordinals.

4.1 Computable ordinals

More details about this section can be found in [14]. An ordinal is defined as the order type of a well-ordered set. When the ordinal is infinite and countable it can be the order-type of a well-ordered set with domain ω. We say that a countable ordinal α is computable if we have a relation $R \subseteq \omega \times \omega$ which is a well-founded linear order of a subset of ω of order-type α and if there is some e such that $(n, m) \in R \leftrightarrow \langle n, m\rangle \in W_e$. In this case we say that e codes for α and we write $|e| = \alpha$. Let us denote by $\mathcal{W}$ the set of integers which code for

computable ordinals and let us denote by $\mathcal{W}_\alpha$ the set of integers which code for computable ordinals strictly smaller than α.

As there are uncountably many countable ordinals, not all of them are computable. Moreover it is known that they form a strict initial segment of the countable ordinals. We denote by ω_1^{ck} the smallest non-computable ordinal. This notion can then be relativised. We say that e is an X-code for the ordinal α if we have a relation $R \subseteq \omega \times \omega$ which is a well-founded linear order of a subset of ω of order-type α and if $(n, m) \in R \leftrightarrow \langle n, m\rangle \in W_e^X$. We then write $|e|^X = \alpha$. We denote by $\mathcal{W}^X$ the set of X-codes for X-computable ordinals, and we denote by $\mathcal{W}_\alpha^X$ the set of X-codes for X-computable ordinals strictly smaller than α. Finally, we call ω_1^X the smallest ordinal which is non-computable relatively to X. Note that any countable ordinal is computable with a representation of itself as an oracle.

4.2 Second order definable sets

We say that a sequence X is hyperarithmetic if for some computable function f and some computable ordinal α we have $n \in X \leftrightarrow f(n) \in \mathcal{W}_\alpha$. One can define the hyperarithmetic sequences equivalently as the sequences we can Turing-compute with sufficiently many successive effective joins and iterations of the jump, constructed by induction over the computable ordinals. Also coming back to the analogy with infinite-time Turing machines we have in [5] a theorem saying that a sequence X is hyperarithmetic iff it can be computed by an infinite-time Turing-machine in a computable ordinal length of time. Similarly we define what is hyperarithmetic for sets. We say that $A \subseteq 2^\omega$ is hyperarithmetic if there exists e and α computable such that $X \in A \leftrightarrow e \in \mathcal{W}_\alpha^X$.

We now define Π_1^1 sequences. While hyperarithmetic sequences can be considered to be the higher counterpart of computable sequences, Π_1^1 sequences can be considered to be the higher counterpart of computably enumerable sequences. They are the sequences one can define with a formula of arithmetic containing arbitrary many first order quantifications and only universal second order quantifications (with no negations in front of them). We have another equivalent definition. A sequence X is Π_1^1 if for some computable function f we have $n \in X \leftrightarrow \exists \alpha < \omega_1^{ck}\ f(n) \in \mathcal{W}_\alpha$. Coming back to the analogy with infinite-time Turing machines, the Π_1^1 sequences also correspond to the sets of integers one can enumerate along computable ordinal length of time with such a machine (when we interpret sequences as sets of integers, considering that n in the set iff the n-th bit of the sequence is one). The Σ_1^1 sequences are their complements (again, when we see sequences as sets of integers), the higher equivalent of co-recursively enumerable sequences. Finally a set A is Π_1^1 if we have an integer e so that $X \in A \leftrightarrow \exists \alpha < \omega_1\ e \in \mathcal{W}_\alpha^X$. We also have a canonical surjection from integers to Π_1^1 sets, so like the arithmetical sets, they can be indexed (in the above example, e is an index for the Π_1^1 set A).

A set is called Δ_1^1 if it is both Σ_1^1 and Π_1^1. By a theorem of Kleene (see chapter 2 in [14]) they are exactly the hyperarithmetical sets. An index for a Δ_1^1 set will consist of a pair of two indices. One expressing it as a Π_1^1 predicate and one expressing its complement as a Π_1^1 predicate.

Note that for Π_1^1 sets, the existential quantification over the ordinals goes up to ω_1. Indeed, if $\omega_1^X > \omega_1^{ck}$ it is possible that $X \in A$ is witnessed by some X-code e for $\alpha \geq \omega_1^{ck}$. This leads us to a Π_1^1 set of great importance for this paper, the set $\{X \mid \omega_1^X > \omega_1^{ck}\}$ (the proof that this set if Π_1^1 can be found in section 9.1 of [13]). We now state two theorems that will be useful for the rest of the paper.

▶ **Theorem 13** (Sacks [14]). *Uniformly in ε and an index for a Δ^1_1 set A, one can compute an index for a Σ^1_1 closed set F so that $F \subseteq A$ and $\lambda(A - F) \leq \varepsilon$. Also one can uniformly from an index of a Δ^1_1 set obtain an index for the Δ^1_1 real being the measure of this set.*

▶ **Theorem 14** (Spector [14]). *If $f : \omega \to \mathcal{W}^X$ is a total $\Pi^1_1(X)$ functional predicate then $\sup_n |f(n)| < \omega_1^X$.*

4.3 Higher randomness

We now introduce notions of randomness which are higher effective variations of the usual randomness notions.

▶ **Definition 15** (Sacks). We say that $X \in 2^\omega$ is **Δ^1_1-random** if it is in no Δ^1_1 nullset.

Martin-Löf was actually the first to promote this notion (see [11]), suggesting that it was the appropriate mathematical concept of randomness. Even if his first definition undoubtedly became the most successful over the years, this other definition got a second wind recently on the initiative of Hjorth and Nies who started to study the analogy between the usual notions of randomness and their higher counterparts. In order to do so they created in [6] a higher analogue of Martin-Löf randomness.

▶ **Definition 16** (Hjorth, Nies). A Π^1_1-Martin-Löf test is given by an effectively null intersection of open sets $\bigcap_n O_n$, each O_n being Π^1_1 uniformly in n. A sequence X is **Π^1_1-ML-random** if it is in no Π^1_1-Martin-Löf test.

This definition is strictly stronger than Δ^1_1-randomness (see Corollary 9.3.5 in [13]). The higher analogue of weak-2-randomness has also been studied (see [3]).

▶ **Definition 17.** We say that X is **weakly-Π^1_1-random** if it belongs to no $\bigcap_n O_n$ with each O_n open set Π^1_1 uniformly in n and with $\lambda(\bigcap_n O_n) = 0$.

Earlier, Sacks gave an even stronger definition, made possible by a theorem of Lusin saying that even though Π^1_1 sets are not necessarily Borel, they remain all measurable.

▶ **Definition 18** (Sacks). We say that $X \in 2^\omega$ is **Π^1_1-random** if it is in no Π^1_1 nullset.

This last definition is of great importance. Kechris proved that there is a universal Π^1_1 nullset, in the sense that it contains all the others (see [8]). Later, Hjorth and Nies gave in [6] an explicit construction of this Π^1_1 nullset. Chong and Yu proved in [3] that weakly-Π^1_1-randomness is strictly stronger than Π^1_1-Martin-Löf-randomness, but it is still unknown whether Π^1_1-randomness coincides with weakly-Π^1_1-randomness.

To separate the two notions, the idea of showing they have different Borel complexity was promoted in [3]. In the next section we show that this will not be possible, by proving that the biggest Π^1_1 nullset has the surprisingly small Borel complexity of $\mathbf{\Sigma^0_3}$. Using results of [17] we will conclude that the Borel complexity of both the weakly-2-randoms and the Π^1_1-randoms, is strictly $\mathbf{\Pi^0_3}$. We now give some important results about higher randomness, that will be needed to achieve this:

▶ **Theorem 19** (Sacks). *The set $\{X \mid \omega_1^X > \omega_1^{ck}\}$ has measure 0.*

Thus no X such that $\omega_1^X > \omega_1^{ck}$ is Π^1_1-random. The following beautiful theorem of Chong, Yu and Nies (see [2]) strengthens Sacks' theorem:

▶ **Theorem 20** (Chong, Yu, Nies). *A sequence X is Π^1_1-random iff it is Δ^1_1-random and $\omega_1^X = \omega_1^{ck}$.*

One could also define the randomness notion introduced by Σ^1_1 nullsets, but this turns out to be equivalent to Δ^1_1-randomness.

▶ **Theorem 21** (Sacks). *A Δ^1_1-random sequence is in no Σ^1_1 nullset. Therefore Σ^1_1-randomness coincides with Δ^1_1-randomness.*

5 Higher Solovay genericity and its variants

▶ **Definition 22.** We say that X is **weakly-Σ^1_1-Solovay-generic** if it belongs to all sets of the form $\bigcup_n F_n$ which intersect with positive measure all the Σ^1_1 closed sets of positive measure, where each F_n is a Σ^1_1 closed set uniformly in n.

▶ **Definition 23.** We say that X is **Σ^1_1-Solovay-generic** if for any set of the form $\bigcup_n F_n$ where each F_n is a Σ^1_1 closed set uniformly in n, either X is in $\bigcup_n F_n$ or X is in some Σ^1_1 closed set of positive measure F, disjoint from $\bigcup_n F_n$.

As in the lower case, one could drop the measure requirement in the definition of Σ^1_1-Solovay-genericity and obtain interesting relations with domination properties. However we will focus in this paper only on (weakly-)Σ^1_1-Solovay-genericity.

Unlike in the lower case, we have that the set of weakly-Σ^1_1-Solovay-generics is of measure 1. We can actually prove easily that they coincide with the weakly-Π^1_1-randoms. Let $\bigcup_n F_n$ be a uniform union of Σ^1_1 closed sets with measure strictly smaller than 1. Let $\bigcap_n O_n$ be its complement. As it is a Π^1_1 set, we have e such that $X \in \bigcap_n O_n \leftrightarrow \exists \alpha < \omega_1 \;\; e \in \mathcal{W}^X_\alpha$. But by Theorem 19 we have that $\{X \mid \exists \alpha \geq \omega_1^{ck} \;\; e \in \mathcal{W}^X_\alpha\} \subseteq \mathcal{S}$ is of measure 0. Thus for some computable α we have that $\{X \mid e \in \mathcal{W}^X_\alpha\}$ has positive measure. As it is a Δ^1_1 set, we can find using Theorem 13 a Σ^1_1 closed set of positive measure contained in it. Thus $\bigcup_n F_n$ does not intersect all Σ^1_1 closed sets of positive measure. Conversely a uniform union of Σ^1_1 closed sets of measure 1 intersects with positive measure any Σ^1_1 closed set of positive measure. Then the weakly-Σ^1_1-Solovay-generics are exactly the weakly-Π^1_1-randoms.

We will now prove that the notion of Σ^1_1-Solovay-genericity is exactly the notion of Π^1_1-randomness. As explained at the end of the section (after Theorem 26), one can also consider this equivalence as the higher counterpart of Proposition 11.

We already know from Theorem 20 that if X is weakly-Π^1_1-random but not Π^1_1-random, then $\omega_1^X > \omega_1^{ck}$. We will show that if X is Σ^1_1-Solovay-generic then $\omega_1^X = \omega_1^{ck}$ which will prove the difficult part of the equivalence.

In order to prove this, we use a technique developed by Sacks and simplified by Greenberg, to show that the set of X with $\omega_1^X > \omega_1^{ck}$ has measure 0. First note that if $\omega_1^X > \omega_1^{ck}$ then there is $o \in \mathcal{W}^X$ such that $|o|^X = \omega_1^{ck}$. In particular for each n we can uniformly restrain the relation coded by o to all elements smaller than n. If $|o|^X$ is a limit ordinal this gives a set of X-codes for ordinals smaller than $|o|^X$ but cofinal (i.e. unbounded) in $|o|^X$. Thus if $\omega_1^X > \omega_1^{ck}$, there is a function $f : \omega \to \mathcal{W}^X$ computable in X such that $\sup_n |f(n)|^X = \omega_1^{ck}$. The idea is the following. Suppose that for some X we have a computable function Φ_e such that:

$$\forall n \;\; \exists \alpha < \omega_1^{ck} \;\; \Phi_e^X(n) \in \mathcal{W}^X_\alpha$$

Suppose also that X is Σ^1_1-Solovay-generic. Then we will show that the supremum of $|\Phi_e^X(n)|$ over $n \in \omega$ is strictly smaller than ω_1^{ck}. To show this we need an approximation lemma, which can be seen as an extension of Theorem 13.

▶ **Lemma 24.** *For a Σ^1_1 predicate $S(X) \leftrightarrow \forall \alpha < \omega_1^{ck} \; e \notin \mathcal{W}^X_\alpha$, uniformly in e and n one can find a Σ^1_1 closed set $F \subseteq S$ with $\lambda(S - F) \leq 2^{-n}$.*

Proof. One can equivalently write $S(X) \leftrightarrow \forall o \in \mathcal{W} \quad e \notin \mathcal{W}^X_{|o|}$. Let S_o be the predicate $e \notin \mathcal{W}^X_{|o|}$. If $o \in \mathcal{W}$ one can uniformly in o and e obtain an index for the Δ^1_1 predicate S_o. The Π^1_1 index for it corresponds to the property : "There exists no bijection from $|e|$ to a strict initial segment of $|o|^X$", and a Π^1_1 index for its complement is : "There exists no infinite backward sequence in $|e|$, and there exists no bijection from $|o|^X$ to an initial segment of $|e|$." Note that if $o \notin \mathcal{W}$, the index is still well defined but does not correspond to anything specific.

Then uniformly in an index for S_o and in n we can find using Theorem 13 a Σ^1_1 closed set F_o such that $F_o \subseteq S_o$ with $\lambda(S_o - F_o) \leq 2^{-o}2^{-n}$. Now let us define $F(X) \leftrightarrow \forall o \in \mathcal{W} \quad X \in F_o$. As an intersection of closed sets, the set F is closed. And as $\mathcal{W}$ is Π^1_1 and F_o is Σ^1_1 uniformly in o, we have that F is Σ^1_1. To conclude we also we have that:

$$\begin{array}{rcl} \lambda(S - F) & = & \lambda(\bigcup_{o\in\mathcal{W}} S - F_o) \\ & \leq & \lambda(\bigcup_{o\in\mathcal{W}} S_o - F_o) \\ & \leq & \sum_{o\in\mathcal{W}} \lambda(S_o - F_o) \leq 2^{-n}. \end{array}$$

◀

We can now prove the desired theorem:

▶ **Theorem 25.** *If Y is Σ^1_1-Solovay-generic then $\omega_1^Y = \omega_1^{ck}$.*

Proof. Suppose that Y is Σ^1_1-Solovay-generic. For any functionnal Φ, consider the set

$$P = \{X \mid \forall n \quad \exists \alpha < \omega_1^{ck} \quad \Phi^X(n) \in \mathcal{W}^X_\alpha\}.$$

Let $P_n = \{X \mid \exists \alpha < \omega_1^{ck} \quad \Phi^X(n) \in \mathcal{W}^X_\alpha\}$ and $P_{n,\alpha} = \{X \mid \Phi^X(n) \in \mathcal{W}^X_\alpha\}$, so $P = \bigcap_n P_n$ and $P_n = \bigcup_{\alpha<\omega_1^{ck}} P_{n,\alpha}$.

From Lemma 24 we can find uniformly in n a uniform union of Σ^1_1 closed sets included in P^c_n with the same measure as P^c_n. From this we can find a uniform union of Σ^1_1 closed sets included in P^c with the same measure as P^c. Suppose that Y is in P, as it is Σ^1_1-Solovay-generic we have a Σ^1_1 closed set F of positive measure containing Y which is disjoint from P^c up to a set of measure 0, formally $\lambda(F \cap P^c) = 0$. In particular for each n we have $\lambda(F \cap P^c_n) = 0$ and then $\lambda(F^c \cup P_n) = 1$. Then let f be the total function which to each pair $\langle n, m\rangle$ associates the smallest code $o_{n,m} \in \mathcal{W}$ such that:

$$\lambda(F^c_{|o_{n,m}|} \cup P_{n,|o_{n,m}|}) > 1 - 2^{-m}$$

where F^c_α is the Δ^1_1 set of strings which are witnessed to be in F^c via an ordinal smaller than α. Using second part of Theorem 13 one can prove that f is Π^1_1. Let $\alpha^* = \sup_{n,m} |f(n,m)|$. By Theorem 14 we have that $\alpha^* < \omega_1^{ck}$. Then we have:

$$\begin{array}{rcl} & \forall n \quad \lambda(F^c_{\alpha^*} \cup \bigcup_{\alpha<\alpha^*} P_{\alpha,n}) & = & 1 \\ \rightarrow & \forall n \quad \lambda(F_{\alpha^*} \cap \bigcap_{\alpha<\alpha^*} P^c_{\alpha,n}) & = & 0 \\ \rightarrow & \forall n \quad \lambda(F - \bigcup_{\alpha<\alpha^*} P_{\alpha,n}) & = & 0 \\ \rightarrow & \lambda(F - \bigcap_n \bigcup_{\alpha<\alpha^*} P_{\alpha,n}) & = & 0 \end{array}$$

As X is Σ^1_1-Solovay-generic it is in particular weakly-Σ^1_1-Solovay-generic and then weakly-Π^1_1-random. Thus by Theorem 21 it belongs to no Σ^1_1 set of measure 0. Then as $F - \bigcap_n \bigcup_{\alpha<\alpha^*} P_{\alpha,n}$ is a Σ^1_1 set of measure 0 we have that X belongs to $\bigcap_n \bigcup_{\alpha<\alpha^*} P_{\alpha,n}$ and then $\sup_n \Phi^X(n) \leq \alpha^* < \omega_1^{ck}$. ◀

We can now prove the equivalence:

▶ **Theorem 26.** *The set of Σ^1_1-Solovay-generics is exactly the set of Π^1_1-randoms.*

Proof. Using Theorem 20 we have that the Σ^1_1-Solovay-generics are included in the Π^1_1-randoms. We just have to prove the reverse inclusion.

Suppose Y is not Σ^1_1-Solovay-generic. If $\omega_1^Y > \omega_1^{ck}$ then Y is not Π^1_1-random. Otherwise $\omega_1^Y = \omega_1^{ck}$ and in this case there is a sequence of Σ^1_1 closed sets $\bigcup_n F_n$ of positive measure such that X is not in $\bigcup_n F_n$ and such that any Σ^1_1 closed set of positive measure which is disjoint from $\bigcup_n F_n$ does not contain Y. The complement of $\bigcup_n F_n$ is a Π^1_1 set P containing Y. Let e be so that $P(X) \leftrightarrow \exists \alpha < \omega_1 \ e \in \mathcal{W}^X_\alpha$. As $\omega_1^Y = \omega_1^{ck}$ and $P(Y)$, we have that $\exists \alpha < \omega_1^{ck} \ e \in \mathcal{W}^Y_\alpha$. But then Y is in a Δ^1_1 set that one can approximate using Theorem 13 by an effective union of Σ^1_1 closed sets of the same measure. Thus as X can be in none of them it is in a Π^1_1 set of measure 0 and then not Π^1_1-random. ◀

The previous theorem gives an interesting corollary, making a connection with another domination property. We say that a sequence X is hyp-dominated if for every total function $f : \omega \to \omega$, Δ^1_1 relative to X, there exists a total Δ^1_1 function g such that g dominates f (i.e. $\forall n \ f(n) \leq g(n)$). Chong, Yu and Nies proved in [2] that all Π^1_1-random sequences are hyp-dominated. It follows from that and from the previous theorem that a sequence X is Σ^1_1-Solovay-generic iff it is weakly-2-random and hyp-dominated. This can be seen as the higher counterpart of Proposition 11.

We have a second corollary, giving a higher bound on the Borel complexity of the Π^1_1-randoms, and then on the biggest Π^1_1 nullset.

▶ **Corollary 27.** *The set of Π^1_1-randoms is $\mathbf{\Pi^0_3}$.*

The $\mathbf{\Pi^0_3}$ set is obtained exactly the same way we obtain the $\mathbf{\Pi^0_3}$ set of computably dominated sequences. The following result of Liang Yu (see [17]) can be used to prove that the set of Π^1_1-randoms is not $\mathbf{\Sigma^0_3}$.

▶ **Theorem 28** (Liang Yu)**.** *Let $\bigcap_n O_n$ be a $\mathbf{\Pi^0_2}$ sets contaning only weakly-Π^1_1-randoms. Then the set $\{F \mid F$ is a Σ^1_1 closed set and $\bigcap_n O_n \cap F = \emptyset\}$ intersects with positive measure any Σ^1_1 closed sets of positive measure.*

It follows that the set of weakly-Π^1_1-randoms cannot be $\mathbf{\Sigma^0_3}$ but also that the set of Π^1_1-randoms cannot be $\mathbf{\Sigma^0_3}$. Indeed, suppose that the set of Π^1_1-randoms is equal to $\bigcup_n \bigcap_m O_{n,m}$ each $O_{n,m}$ being open. For each n let $A_n = \{F \mid F$ is a Σ^1_1 closed set and $\bigcap_m O_{n,m} \cap F = \emptyset\}$. We have $\bigcap_n A_n \cap \bigcup_n \bigcap_m O_{n,m} = \emptyset$, and from Theorem 28 we have that $\bigcap_n A_n$ contains some Solovay-Σ^1_1-generic elements, which contradicts that $\bigcup_n \bigcap_m O_{n,m}$ contains all of them.

The question whether it is possible for X to be weakly-Solovay-Σ^1_1-generic but not Solovay-Σ^1_1-generic (equivalently weakly-Π^1_1-random but not Π^1_1-random) is still open. The technique that we use in the lower case to separate weak genericity from non weak genericity does not seem to work here.

Acknowledgements. I would like to thank Laurent Bienvenu, Noam Greenberg, Paul Shafer and Liang Yu for helpful comments and discussions.

References

1 George Barmpalias, Rod Downey, and Keng Meng Ng. Jump inversions inside effectively closed sets and applications to randomness. *Journal of Symbolic Logic*, 76(2):491–518, 2011.

2 Chi Tat Chong, André Nies, and Liang Yu. Lowness of higher randomness notions. *Israel Journal of Mathematics*, 166:39–60, 2008.

3 Chi Tat Chong and Liang Yu. Randomness in the higher setting. *under refereeing*, `http://math.nju.edu.cn/~yuliang/cy1.pdf`.

4 Paul J Cohen. *Set theory and the continuum hypothesis*. Dover Publications, 1966.

5 Joel David Hamkins and Andy Lewis. Infinite time Turing machines. *Journal of Symbolic Logic*, pages 567–604, 2000.

6 Greg Hjorth and André Nies. Randomness via effective descriptive set theory. *Journal of the London Mathematical Society*, 75(2):495–508, 2007.

7 Carl G Jockusch Jr. Simple proofs of some theorems on high degrees of unsolvability. *Canadian Journal of Mathematics*, 29(5):1072–1080, 1977.

8 Alexander S Kechris. The theory of countable analytical sets. *Transactions of the American Mathematical Society*, 202:259–297, 1975.

9 S Kurtz. Randomness and genericity in the degrees of unsolvability. phd diss., University of Illinois. *Urbana*, 1981.

10 Per Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.

11 Per Martin-Löf. On the notion of randomness. *Studies in Logic and the Foundations of Mathematics*, 60:73–78, 1970.

12 Benoit Monin. Higher randomness and forcing with closed sets, `http://www.liafa.univ-paris-diderot.fr/~benoitm/ressources/papers/paper_stacs_2014.pdf`, 2014.

13 Andre Nies. *Computability and Randomness*. Oxford University Press, 2009.

14 Gerald E Sacks. *Higher recursion theory*. Springer, 2010.

15 Robert M Solovay. A model of set-theory in which every set of reals is Lebesgue measurable. *The Annals of Mathematics*, 92(1):1–56, 1970.

16 Liang Yu. Descriptive set theoretical complexity of randomness notions. *Fundamenta Mathematicae*, 215:219–231, 2011.

17 Liang Yu. LogicBlog2013, Higher randomness, `http://dl.dropboxusercontent.com/u/370127/Blog/Blog2013.pdf`, 2013.

Near-Optimal Generalisations of a Theorem of Macbeath

Nabil H. Mustafa[1] and Saurabh Ray[2]

1 **Université Paris-Est, Laboratoire d'Informatique Gaspard-Monge, Paris, France**
mustafan@esiee.fr

2 **Department of Computer Science, New York University Abu Dhabi, Abu Dhabi, United Arab Emirates**
sr194@nyu.edu

Abstract

The existence of Macbeath regions is a classical theorem in convex geometry ("A Theorem on non-homogeneous lattices", Annals of Math, 1952). We refer the reader to the survey of I. Barany for several applications [3]. Recently there have been some striking applications of Macbeath regions in discrete and computational geometry.

In this paper, we study Macbeath's problem in a more general setting, and not only for the Lebesgue measure as is the case in the classical theorem. We prove near-optimal generalizations for several basic geometric set systems. The problems and techniques used are closely linked to the study of ϵ-nets for geometric set systems.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Epsilon Nets, Cuttings, Union Complexity, Geometric Set systems, Convex Geometry

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.578

1 Introduction

The goal of this paper is to study small, uniform-sized decompositions of geometric range spaces which approximate the range space. This can be seen as a discrete analogue and extension of the classical result of Macbeath [12] in convex geometry, as well as having several basic connections to well-studied problems in discrete geometry.

Classical Macbeath Regions. Given a convex body K in $\mathbb{R}^d$ of unit volume, and a parameter $\epsilon > 0$, the theorem of Macbeath states the existence of disjoint convex bodies of K, each of volume $\Theta(\epsilon)$, called *Macbeath regions*, such that any halfspace containing at least ϵ volume of K completely contains one of these convex objects. Formally, the following theorem follows from their work:

▶ **Theorem 1** (Macbeath Regions). *Given a convex body $K \subset \mathbb{R}^d$ of unit volume, and a parameter $0 < \epsilon < 1/(2d)^{2d}$, there exists a set of convex objects $\mathcal{M}$, $|\mathcal{M}| = O((1/\epsilon)^{1-2/(d+1)})$, such that for any halfspace h with $vol(h \cap K) \geq \epsilon$, there exists $K_i \in \mathcal{M}$ such that $K_i \subset h \cap K$ and*

$$vol(K_i) \geq \frac{1}{(6d)^{3d}} vol(h \cap K)\,.$$


© Nabil H. Mustafa and Saurabh Ray;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 578–589

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


The existence of Macbeath regions was first proven in the paper of Macbeath [12], with several later applications to geometric problems. Edwald, Larmen and Rogers [9] used it for cap coverings, which was later further extended by Barany and Larman [4] (also see Barany [3] for a survey of this and several other results). It was used for lower-bounds on range-searching by Bronniman, Chazelle and Pach [5]. And very recently, Macbeath regions were used in an elegant way by Arya, Fonseca and Mount [2] for computing near-optimal Hausdorff approximations to polytopes.

A fundamental and powerful result in computational geometry is the existence of small-sized ϵ-nets: given a set system $(X, \mathcal{R})$, and a parameter ϵ, an ϵ-net is a subset $X' \subset X$ such that $X' \cap R \neq \emptyset$ for all $R \in \mathcal{R}$ where $|R| \geq \epsilon |X|$. The famous theorem of Haussler-Welzl [10] shows that ϵ-nets of size $O(d/\epsilon \log d/\epsilon)$ exist for set systems $(X, \mathcal{R})$, where d is the VC-dimension of the set system $(X, \mathcal{R})$. This bound was later improved in [11] to an optimal bound of $(1 + o(1))(\frac{d}{\epsilon} \log(1/\epsilon))$. By now ϵ-nets are an indispensable tool in combinatorics and algorithms [20, 6, 8, 15, 1, 19, 17, 16, 18, 2, 11, 13, 7].

Note that Macbeath's original theorem immediately implies an ϵ-net kind of a result: for any convex body C in $\mathbb{R}^d$ of volume V, it is possible to pick $O(\frac{1}{\epsilon})$ points in C which stab all halfspaces containing an ϵ-th fraction of the volume of C. However, the statement itself is much stronger than that: instead of just points, it gives us $O(\frac{1}{\epsilon})$ regions of volume $\Theta(\epsilon V)$ so that each halfspace containing an ϵ fraction of the volume of C contains one of the regions. The same kind of result is not true in general in a discrete setting (with counting measure instead of Lebesgue measure) for halfspaces in $\mathbb{R}^d$. However, it is true for halfspaces in $\mathbb{R}^3$. Given n points in $\mathbb{R}^3$, one can find $O(\frac{1}{\epsilon})$ groups containing $\Theta(\epsilon n)$ points each so that any halfspace containing ϵn points contains one of the groups. This is much stronger than just the existence of ϵ-nets of size $O(\frac{1}{\epsilon})$.

This raises the intriguing question: of the large number of results known for ϵ-nets, which can be optimally strengthened like above?

Combinatorial Macbeath Regions. Given the existence of decomposition of a convex set K into roughly equal-volume subsets with respect to halfspaces, the natural question is to prove the existence of a small-sized set of Macbeath regions for the counting measure (instead of the Lebesgue measure).

So the problem is: given a set P of n points in $\mathbb{R}^d$ and a parameter $\epsilon > 0$, one would like to construct sets $\mathcal{P} = \{P_1, \ldots, P_m\}, P_i \subset P$, such that each set P_i has size $\Omega(\epsilon n)$, and any halfspace containing at least ϵn points contains a set in $\mathcal{P}$.

It turns out that this is implied by a classical result in discrete geometry, called *shallow cuttings*, which states the following [13, 7]. Given a set of n regions $\mathcal{S}$ in $\mathbb{R}^d$ and two parameters r, l, a $(1/r, l)$-shallow cutting w.r.t. $\mathcal{S}$ is a partition of $\mathbb{R}^d$ into cells (of constant descriptive complexity) such that i) each cell is intersected by the boundary of at most n/r regions of $\mathcal{S}$, and ii) the number of cells containing points of depth smaller than l is at most $O((rl/n + 1)^d \cdot n/l \cdot \phi(n/l))$. A set of regions is said to have union complexity $\phi(\cdot)$ if the combinatorial complexity of the union of any r of the regions is at most $r\phi(r)$.

It can be observed that this statement implies a Macbeath-type statement for halfspaces, and more generally, for the following problem for regions of small union complexity: given a set of regions $\mathcal{S}$ of union complexity $\phi(\cdot)$, the objective is to compute a family $\mathcal{U}$ of subsets of $\mathcal{S}$, each of size $\Omega(\epsilon n)$, such that any point contained in at least ϵn objects of $\mathcal{S}$ hits all elements of some set in $\mathcal{U}$.

To construct the Macbeath sets $\mathcal{U}$ for regions in $\mathcal{S}$, fix $l = 2\epsilon n$, $r = 2/\epsilon$, and construct a $(1/r, l)$-shallow cutting for $\mathcal{S}$. For a cell C in the shallow cutting, let $r(C)$ be the set of regions in $\mathcal{S}$ that completely contain C, i.e., $s_i \in r(C)$ iff $C \subset s_i$. Now, for all cells C

that contain a point of depth at most $2\epsilon n$ (called *shallow cells*), add $r(C)$ to $\mathcal{U}$. By the shallow-cutting theorem, the number of cells containing a point of depth ϵn is

$$O((rl/n+1)^d \cdot n/l \cdot \phi(n/l)) = O(4^d \cdot 1/\epsilon \cdot \phi(2/\epsilon))$$

and so $|\mathcal{U}| = O(1/\epsilon \cdot \phi(2/\epsilon))$. To show that sets in $\mathcal{U}$ form the required Macbeath regions, recall that the cutting partitions $\mathbb{R}^d$ into a set of cells such that each cell intersects the boundary of at most $n/r = \epsilon n/2$ objects in $\mathcal{S}$. For a point p hitting ϵn regions, let C be the shallow cell containing p. By the property of shallow-cuttings, of the ϵn regions containing p, at most $\epsilon n/2$ regions can intersect C. The remaining at least $\epsilon n/2$ regions must then completely contain C, and so for the $r(C)$ added to $\mathcal{U}$ that contains p, we have $|r(C)| \geq \epsilon n/2$.

The above shows the existence of $O(1/\epsilon\phi(1/\epsilon))$ sets such that any point contained in $\Theta(\epsilon n)$ sets of $\mathcal{S}$ must hit one of the constructed sets. To make it work for all sets of size at least ϵn, we can iteratively construct sets for increasing values of ϵ, i.e., $\epsilon, 2\epsilon, \ldots, 2^i\epsilon$, and take the union, still obtaining $O(1/\epsilon\phi(1/\epsilon))$ sets.

For our problem of halfspaces, simply dualize each point in P to a halfspace, and apply the above construction. For halfspaces, $r\phi(r) = O(r^{\lfloor d/2 \rfloor})$, and so we get the following combinatorial version of Macbeath: given a set P of n points in $\mathbb{R}^d$, there exists a set $\mathcal{P} = \{P_1, \ldots, P_m\}, P_i \subset P$, with $m = \frac{1}{\epsilon^{\lfloor d/2 \rfloor}}$, such that i) all sets in $\mathcal{P}$ have size $\Omega(\epsilon n)$, and ii) any halfspace containing ϵn points of P contains at least one set in $\mathcal{P}$.

The existence of a number of other 'Macbeath-type' statements for several other range spaces is also implied by the above proof. In particular, for regions with linear union complexity, i.e., $\phi(r) = O(1)$, there exist linear-sized Macbeath regions. This points to the possibility of the existence of such structural partitioning properties for a wide range of sets derived from geometric objects. In this paper we initiate a systematic study of the analogues of Macbeath regions for other commonly studied geometric set-systems.

Our results. Given a set system $(X, \mathcal{R})$ and $\epsilon > 0$, we say that a set system $\mathcal{U}$ over X is an ϵ-Macbeath net (or ϵ-Mnet for short) of $(X, \mathcal{R})$ if i) each set in $\mathcal{U}$ has size $\Omega(\epsilon|X|)$, and ii) for every set $R \in \mathcal{R}$ of size at least $\epsilon|X|$, there exists a set $U \in \mathcal{U}$ such that $U \subseteq R$. The *size* of an ϵ-Mnet $\mathcal{U}$ is $|\mathcal{U}|$. Parameterizing the problem a little further, if each set in $\mathcal{U}$ has size at least $\epsilon n/k$, we call it a $\frac{1}{k}$-heavy ϵ-Mnet .

In the study of ϵ-nets for geometric set systems, there are two types of set-systems that are frequently studied: each of these are defined by a set of points P, and a set of regions $\mathcal{S}$. In the so-called 'primal' set-systems, P is taken as the ground set, and the subsets are induced by the regions in $\mathcal{S}$, where a region R induces the subset $R \cap P$. In the so-called 'dual' set-systems, $\mathcal{S}$ is taken as the ground set, and the subsets are induced by points in P, where a point p induces the subset consisting of the regions in $\mathcal{S}$ containing p. Often in the dual setting P is not mentioned, and is assumed to be the entire Euclidean space.

Earlier we pointed out the existence of ϵ-Mnets for halfspaces of size $O(1/\epsilon^{\lfloor d/2 \rfloor})$. Unfortunately this bound cannot be improved substantially: in Section 3, we show that it is not very far from optimal, that is for any d, there exist a set of n points in $\mathbb{R}^d$ where any $\mathcal{P}$ satisfying conditions i) and ii) has size at least $\Omega(\frac{1}{\epsilon^{\lceil (d-1)/3 \rceil}})$.

The earlier statement in fact proved the existence of Macbeath sets for the dual problem for general regions in terms of their union complexity. Namely, it showed:

▶ **Theorem 2** (ϵ-Mnets for dual set-systems). *Let $\mathcal{S}$ be a set of n regions in $\mathbb{R}^d$ with union complexity $\phi(r)$[1]. Then there exists an ϵ-Mnet for the dual set-system defined by $\mathcal{S}$ and $\mathbb{R}^d$ (i.e., subsets of $\mathcal{S}$ hit by a point in the plane for the set system), of size $O(\frac{1}{\epsilon}\phi(\frac{1}{\epsilon}))$.*

[1] And satisfying certain technical conditions of bounded algebraic complexity. See [8] for a broader discussion on this.

Interestingly, the dependence of $\phi(\cdot)$ in general cannot be reduced to, for example, $\log \phi(\cdot)$, as is the bound for ϵ-nets. A set-system which provides a counter-example follows from our first main result, which considers ϵ-Mnets for the commonly-studied range-space induced by axis-parallel rectangles in the plane. For these, we optimally tighten the result produced by shallow-cuttings by improving the upper-bound, and then providing a matching lower-bound. We prove the following in Section 2:

▶ **Theorem 3** (ϵ-Mnet for rectangles in $\mathbb{R}^2$)**.** *Let R be a set of n rectangles, P a set of n points in $\mathbb{R}^2$, $\epsilon > 0$ a parameter, and $k \geq 2$ an integer:*

1. *There exists a $\frac{1}{2^k}$-heavy ϵ-Mnet for the dual set-system defined by R and $\mathbb{R}^2$, of size $O(4^k/\epsilon^{1+1/k})$. Furthermore, this cannot be significantly improved: there exists a set $\mathcal{R}$ of n axis-parallel rectangles such that any $\frac{1}{k}$-heavy ϵ-Mnet for the dual set-system defined by $\mathcal{R}$ and $\mathbb{R}^2$ has size $\Omega((1/\epsilon)^{1+1/(k-1)})$.*
2. *There exists an ϵ-Mnet for the primal set-system defined by P and R, of size $O((1/\epsilon)\log 1/\epsilon)$. Furthermore, this cannot be significantly improved: there exists a set P of n points in $\mathbb{R}^2$ such that any $\frac{1}{k}$-heavy ϵ-Mnet for the primal set-system defined by P and R has size $\Omega(\frac{1}{\epsilon}\log_k \frac{1}{\epsilon})$.*

Our second main result is to consider the primal case, i.e., where the input is a set of points P, and the ranges are defined by geometric objects such as circles, k-sided polygons, and in general, objects of some fixed descriptive complexity. We prove the following in Section 3:

▶ **Theorem 4.** *Let P be a set of n points in $\mathbb{R}^2$. Then one can construct ϵ-Mnets of size[2]:*

- *$O(1/\epsilon)$ for sets induced by disks in the plane,*
- *$O(1/\epsilon)$ for sets induced by rectangles all intersecting a fixed line l,*
- *$\tilde{O}(1/\epsilon^2)$ for sets induced by lines, cones, strips in the plane,*
- *$\tilde{O}(1/\epsilon^3)$ for sets induced by triangles, and in general k-sided polygons in the plane (the constant in the asymptotic notation depends on k).*

We further show in Section 3 that near-linear bounds (like those achieved for halfspaces in 2 and 3 dimensions, or for the dual set-systems of linear union-complexity) are not possible for even simple primal set-systems: there exist a set P of n points in the plane such that any ϵ-Mnet for lines must have size $\Omega(1/\epsilon^2)$. This implies that for strips or cones in the plane, the same bound holds, ruling out near-linear bounds for even the simplest type of geometric objects.

We conclude our study by observing that the above series of results, while their proofs use different techniques, indicate an intriguing relation between the size of ϵ-nets and the size of ϵ-Mnets . In all cases, they obey the following: if for a range-space (dual, or primal), the ϵ-nets have size $O(1/\epsilon f(1/\epsilon))$, then the size of ϵ-Mnets for the same range-space is $O(1/\epsilon c^{f(1/\epsilon)})$, where c is constant. So for all spaces known to have linear-sized ϵ-nets (which is optimal), our proofs prove the existence of linear-sized ϵ-Mnets (which is optimal). For the primal set-systems of axis-parallel rectangles in the plane, the ϵ-nets have size $O(1/\epsilon \log\log 1/\epsilon)$ (shown to be optimal) [1, 17], and our result show ϵ-Mnets of size $O(1/\epsilon \log 1/\epsilon)$ (which we show to be optimal). And for the remaining ranges which have ϵ-nets of size $O(1/\epsilon \log 1/\epsilon)$, we show the existence of ϵ-Mnets of size $O(1/\epsilon^c)$. It would be interesting to see if there is any connection with the (still) open problem of finding the right bound on the size of ϵ-nets for lines in the plane.

[2] $\tilde{O}(\cdot)$ ignores polylogarithmic factors.

2 Proof of Theorem 3

In this section we prove Theorem 3, which completely resolves the case for rectangles. We start by giving the lower-bounds for the primal and the dual problem, and then give the matching upper-bounds for both.

2.1 Lower Bounds

The following Lemma gives the key insight to studying ϵ-Mnet for rectangles.

▶ **Lemma 5.** *For any integers $r, d \geq 1$, consider the grid $G = [r]^d$ in $\mathbb{R}^d$. Then the set system on G induced by incidences with axis-parallel lines can be realized by point-rectangle incidences in $\mathbb{R}^2$.*

Proof. Let $r \geq 1$ be any integer and let $[r]$ represent the set $\{0, \cdots, r-1\}$. Let $G = [r]^d$ which can be thought of as a finite d-dimensional grid of side length r. For some fixed integers $a_1, \cdots, a_{i-1}, a_{i+1}, \cdots a_d \in [r]$, consider the set of points $S_i(a_1, \cdots, a_{i-1}, a_{i+1}, \cdots a_d) = \{(a_1, \cdots, a_{i-1}, x, a_{i+1}, \cdots a_d) : x \in [r]\}$. We call such a set a *line* in direction i. There are dr^{d-1} lines, with r^{d-1} lines in each of the d directions.

We will show that there exists a mapping $\pi : G \mapsto \mathbb{R}^2$ s.t. for each line l (in any direction i), the smallest (inclusion minimal) axis parallel rectangle containing the image $\pi(l)$ of the points in l does not contain the image of any other points of G. Here is the mapping π that we will use: $\pi((a_1, \cdots, a_d)) = \sum_j a_j \vec{v_j}$, where $\vec{v_j} = (r^j, r^{d+1-j})$. For any point $z \in G$, we will interpret $p = \pi(z)$ both as a vector and as a point, as suitable. When treating it as a vector we will denote it as $\vec{p}$.

For any point $p = (a_1, \cdots, a_d) \in G$, let $\vec{V}_{<i}(p)$ denote the vector $\sum_{j<i} a_j \vec{v_j}$ and $\vec{V}_{>i}(p)$ denote the vector $\sum_{j>i} a_j \vec{v_j}$. Thus we can write $\pi(p)$ as $\vec{V}_{<i}(p) + a_i \vec{v_i} + \vec{V}_{>i}(p)$.

Consider the line $l = S_i(a_1, \cdots, a_{i-1}, a_{i+1}, \cdots a_d)$. The minimal rectangle R containing $\pi(l)$ is defined by the two opposite corners $\pi(u)$ and $\pi(v)$, where $u = (a_1, \cdots, a_{i-1}, 0, a_{i+1}, \cdots a_d)$ and $v = (a_1, \cdots, a_{i-1}, r-1, a_{i+1}, \cdots a_d)$ are the extreme points in l. The width of R is $(r-1)r^i$ and its height is $(r-1)r^{d+1-i}$.

Consider any point $z = (b_1, \cdots, b_d) \in G \setminus l$. Let $p = \pi(z)$ and let q be the point $\sum_{j \neq i} a_j \vec{v_j} + b_i \vec{v_i} \in l$. Now, $\vec{p} - \vec{q} = (\vec{V}_{<i}(p) - \vec{V}_{<i}(q)) + (\vec{V}_{>i}(p) - \vec{V}_{>i}(q))$. Since $\vec{p} \neq \vec{q}$, one of the summands must be non-zero. Without loss of generality assume that the latter summand is non-zero. The other case is symmetric.

Since the vector $\vec{V}_{>i}(p) - \vec{V}_{>i}(q)$ is an integral combinations of the vectors v_j , $j > i$, its x-coordinate has magnitude at least r^{i+1}. On the other hand the x-coordinate of $(\vec{V}_{<i}(p) - \vec{V}_{<i}(q))$ has magnitude at most $\sum_{1 \leq j < i} (r-1)r^j = r^i - r^{i-1}$. Therefore, the horizontal distance between p and q is at least $r^{i+1} - (r^i - r^{i-1})$ which is greater than the width of R. Hence, $p \notin R$. When $(\vec{V}_{<i}(p) - \vec{V}_{<i}(q)) \neq 0$, a similar argument holds for the y-coordinates of p and q showing that their vertical distance is larger than the height of R. ◀

Proof of Theorem 3 part 1) lower-bound. We now show that for any integer constant $d \geq 2$, there exists a set $\mathcal{R}$ of n axis-parallel rectangles such that any $\frac{1}{k}$-heavy ϵ-Mnet for $\mathcal{R}$ w.r.t. points has size $\Omega((1/\epsilon)^{1+1/(k-1)})$.

Proof. Now apply Lemma 5 with $d = k$ and $r = \epsilon^{-\frac{1}{d-1}}$. Let G be the grid $[r]^d$ as before. We set $P = \{\pi(p) : p \in G\}$ and we take $\mathcal{R}$ to be the set of rectangles with $\epsilon n/d$ copies of each of the set $\mathcal{R}'$ of dr^{d-1} rectangles corresponding to the dr^{d-1} lines in G. Note that

$|\mathcal{R}| = \epsilon n/d \cdot dr^{d-1} = n$. Since each of the points in G is contained in d lines (one in each direction), the points in P are contained in d rectangles of $\mathcal{R}'$ and consequently ϵn rectangles of $\mathcal{R}$. Since there is at most one line through two points in G there is at most one rectangle in $\mathcal{R}'$, and hence at most $\epsilon n/d$ rectangles of $\mathcal{R}$ that contain any pair of points $p, q \in P$. Since for any $\frac{1}{k}$-heavy ϵ-Mnet $\mathcal{U}$, each $U \in \mathcal{U}$ has size more than $\epsilon n/k$, it must be that no set in $\mathcal{U}$ can be contained in two sets $\mathcal{R}(p)$ and $\mathcal{R}(q)$ induced by two distinct points p and q in P. Therefore $|\mathcal{U}| \geq |P| = r^d = \epsilon^{-\frac{k}{k-1}}$. ◀

Proof of Theorem 3 part 2) lower-bound. We now show that for any integer constant $k \geq 2$, there exists a set P of n points in $\mathbb{R}^2$ such that any $\frac{1}{k}$-heavy ϵ-*Mnet* of P, w.r.t. axis-parallel rectangles, has size $\Omega(\frac{1}{\epsilon}\log_k \frac{1}{\epsilon})$.

Proof. Apply Lemma 5 with $r = k$, and d such that $r^{d-1} = \frac{1}{\epsilon}$. Let $\mathcal{R}$ be the set of $dr^{d-1} = 1/\epsilon \log_k 1/\epsilon$ rectangles corresponding to the lines of G, and let P be the set of points with $\epsilon n/r$ copies of each $\pi(p), \forall p \in G$. Each of the rectangles in $\mathcal{R}$ contains $r\epsilon n/r = \epsilon n$ points of P. Any two rectangles of $\mathcal{R}$ share at most $\epsilon n/r = \epsilon n/k$ points of P. Thus no two rectangles in $\mathcal{R}$ may share the same set $U \in \mathcal{U}$ of a $\frac{1}{k}$-heavy ϵ-*Mnet* $\mathcal{U}$. Since each of them must contain some $U \in \mathcal{U}$, we have $|\mathcal{U}| \geq |\mathcal{R}|$ and the result follows. ◀

2.2 Upper Bounds

We now give constructions which match the preceding lower-bounds to complete the proof of Theorem 3 part 1. Our argument is based essentially on the technique of boot-strapping; at the cost of worse constant factors, we give a simple exposition below.

Construct a hierarchical subdivision of the rectangles in R by vertical lines, with an integer $k = 1/\epsilon^{1/d}$, as follows. Let $n_i = n/k^i$, and $\epsilon_i = \epsilon(k/2)^i$. At the 0-th level ($i = 0$), let $l_1^0, \ldots, l_k^0$ by a set of k vertical lines such that the number of rectangles of R lying between two consecutive lines ('a slab') is at most n/k. Let R_j^0 be the set of rectangles lying entire in the j-th slab. For each line l_j^0, construct a $\epsilon_i/4$-Mnet for all of the (at most) n rectangles of R intersecting it. Furthermore, recursively construct a ϵ_{i+1}-Mnet for the rectangles in R_j^0 for each j. The recursive construction continues for d steps, where at the i-level, there are k^i total subproblems, each subproblem has at most $n_i = n/k^i$ rectangles, and with $\epsilon_i = \epsilon(k/2)^i$. Finally we use a direct $O(1/\epsilon_d^2)$-sized construction for the ϵ_d-Mnet of the final k^d subproblems at level $i = d$: construct $10/\epsilon_d$ vertical and horizontal lines so that each vertical and horizontal slab contains at most $\epsilon_d n/10$ rectangles, and for each grid cell c, add to $\mathcal{U}$ any $\epsilon_d n_d/2$ rectangles containing c (if possible). Now notice that any point in $\epsilon_d n_d$ rectangles must have at most $\epsilon_d n_d/5$ rectangles intersecting the cell boundary in which it lies, and so at least $\epsilon_d n_d/2$ of the remaining ones would form a set in $\mathcal{U}$. The next two claims show that all these Mnet together form a ϵ-Mnet $\mathcal{U}$ for R of the required size, and we're done.

▶ Claim 1. Each set in $\mathcal{U}$ has size $\Theta(\epsilon n/2^d)$. The size of $\mathcal{U}$ is $O(4^d/\epsilon^{1+1/d})$.

Proof. At the i-level there are k^i subproblems, each of size at most $n_i = n/k^i$ with $\epsilon_i = \epsilon(k/2)^i$. For each such subproblem, we partition its n_i rectangles by k lines, and construct a $\epsilon_i/4$-Mnet for the rectangles intersecting of these k lines. Note that the set of rectangles intersecting any line, and clipped to one side of the line have linear union complexity and by our Theorem on the dual set-systems, there exists a $\epsilon_i/4$-Mnet of size $O(1/\epsilon_i)$. Hence the

total size over all internal subproblems is:

$$\sum_{i=0}^{d} k^i \cdot k \cdot O(\frac{1}{\epsilon_i}) = \sum_{i=0}^{d} k^{i+1} \cdot O(\frac{2^i}{\epsilon k^i}) = \sum_{i=0}^{d} O(\frac{2^i}{\epsilon^{1+1/d}}) = O(\frac{2^d}{\epsilon^{1+1/d}}) .$$

After d steps, we have k^d subproblems, each with at most n/k^d rectangles, and $\epsilon_d = \epsilon(k/2)^d$. Now just use a direct construction which constructs an ϵ-Mnet of size $O(1/\epsilon^2)$, to get the total size of Mnet at the last step to be $O(\frac{k^d}{\epsilon_d^2}) = O(\frac{4^d}{\epsilon^2 k^d}) = O(4^d/\epsilon)$.

At any level i, we construct a ϵ_i-Mnet on a set of at most n/k^i rectangles. So each set in the Mnet has size $\epsilon_i \cdot n/k^i = O(\epsilon n/2^i)$. ◀

▶ Claim 2. Each point in at least ϵn rectangles of R contains a set of $\mathcal{U}$.

Proof. Take a point q lying in at least ϵn rectangles of R. At the 0-th level, say q lies in the vertical slab defined by lines l_j^0 and l_{j+1}^0. If q hits at least $\epsilon n/4$ rectangles intersected by either l_j^0 or l_{j+1}^0, say l_j^0, then it hits at least $\epsilon n/4$ rectangles out of a total of at most n rectangles intersected by l_j^0. So the $(\epsilon_i/4 = \epsilon/4)$-Mnet for l_j^0 will have a set contained by q. Otherwise q hits at least $\epsilon_0 n_0/2 = \epsilon n/2 = \epsilon(k/2)(n/k) = \epsilon_1 n_1$ rectangles of the set R_j^0 of size $n_1 = n_0/k$, and we proceed to this subproblem.

In general, at the i-level, each subproblem has $n_i = n/k^i$ rectangles, with $\epsilon_i = \epsilon(k/2)^i$. Then either q hits at least $\epsilon_i \cdot n_i/4$ rectangles intersecting one of the lines, and so will contain a set from the $\epsilon_i/4$-Mnet constructed for each of the k vertical lines. Or q contains at least $\epsilon_i n_i/2$ rectangles out of a total of n_i/k rectangles lying in one of the slabs defined by the k vertical lines. But as

$$\epsilon_i n_i/2 = \epsilon/2 \cdot (k/2)^i \cdot n/k^i = \epsilon(k/2)^{i+1} n/k^{i+1} = \epsilon_{i+1} n_{i+1} ,$$

q will be covered inductively by the ϵ_{i+1}-Mnet constructed for the $n_{i+1} = n/k^{i+1}$ rectangles in one of the resulting subproblems at level $i+1$. ◀

Finally we present the tight upper-bound for the primal case of axis-parallel rectangles in Theorem 3 part 2.

Assume $P = \{p_1, \ldots, p_n\}$ are sorted by their x-coordinates. Given P, construct the balanced binary subdivision of P with vertical lines: divide P by a vertical line into two equal-sized subsets P_0^0, P_1^0, and then recursively divide each of these sets for $\log 1/\epsilon$ levels. Let P_j^i be the j-th resulting subset of P at level i, i.e., $P_j^i = \{p_{jn/2^i}, \ldots, p_{(j+1)n/2^i - 1}\}$.

For each set P_j^i, and for each of its two bounding lines l_0 and l_1 in the binary subdivision above, construct a $2^{i-1}\epsilon$-Mnet for the following primal set-system: the base set is P_j^i, and given the line $l \in \{l_0, l_1\}$, the sets are induced by rectangles intersecting the line l. Note that all points of P_j^i lie on the same side of l. Let $\mathcal{U}$ be the union of all these Mnets . By Theorem 4, a ϵ-Mnet for such a set-system has size $O(1/\epsilon)$.

We now prove that $\mathcal{U}$ is an ϵ-Mnet of P, w.r.t. axis-parallel rectangles, of size $O(1/\epsilon \log 1/\epsilon)$.

▶ Claim 3. Each set in $\mathcal{U}$ has size $\Theta(\epsilon n)$, and size of $\mathcal{U}$ is $O(1/\epsilon \log 1/\epsilon)$.

Proof. P_j^i has $n/2^i$ points, and so a $(2^{i-1}\epsilon)$-Mnet of P_j^i has sets of size $O(2^{i-1}\epsilon \cdot n/2^i) = O(\epsilon n)$. Each such $2^{i-1}\epsilon$-Mnet has size $O(1/2^i\epsilon)$, there are 2^i sets P_j^i at level i, and a total of $\log 1/\epsilon$ levels. Hence the size of $\mathcal{U}$ is $O(1/2^i\epsilon \cdot 2^i \cdot \log 1/\epsilon) = O(1/\epsilon \log 1/\epsilon)$. ◀

▶ Claim 4. Each axis-parallel rectangle containing ϵn points of P contains a set of $\mathcal{U}$.

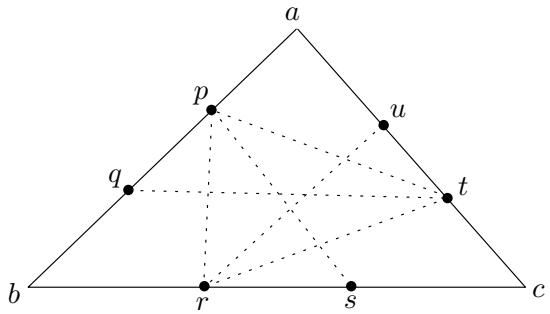


Figure 1 The union of triangles *aqt*,*bsp*, *cur* and *prt* covers the triangle *abc*.

Proof. Let R be an axis-parallel rectangle containing ϵn points of P. Let i be the smallest index such that R intersects exactly one vertical line separarting two sets P^i_j and P^i_{j+1} at level i. Say R intersects the line l separating P^i_j and P^i_{j+1}. Then R must contain at least $\epsilon n/2$ points from either P^i_j or P^i_{j+1}, say P^i_j. Let R' be the part of R on the side of l towards P^i_j. All such R' form a set of psuedo-disks, and so R' must contain at least one set of the $2^{i-1}\epsilon$-Mnet for P^i_j, as

$$|R \cap P^i_j| = |R' \cap P^i_j| \geq \epsilon n/2 = 2^{i-1}\epsilon \cdot n/2^i = 2^{i-1}\epsilon \cdot |P^i_j|\,.$$

◀

3 Proof of Theorem 4

In this section we give the proof of Theorem 4. Given a set P of n points, first we give the proof for the most difficult case, that of the primal set-system induced by triangles, and in general, k-sided polygons in the plane. At the end we sketch out the modifications required for the rest of the cases of strips, cones and disks.

So we are given a set P of n points, and its subsets induced by the family of all k-sided polygons. The objective, as before, is to compute a small-sized ϵ-Mnet . We will assume P to be in general position.

Since a k-sided polygon can be triangulated with k triangles, any k-sided polygon containing ϵn points of P also contains a triangle containing $\epsilon n/k$ points. Hence an ϵ/k-Mnet with respect to triangles is an ϵ-Mnet with respect to k-sided polygons. We can therefore restrict ourselves to triangles.

Consider any triangle T in the plane that contains ϵn points of P. By moving the sides of the triangle we can ensure that each side of T contains at least two points of P and this can be done in such a way that no point outside T enters the interior of P. Some points in the interior of T may have moved to its boundary and some point outside T may also have moved to the boundary. Since at most 6 points may be on the boundary of T, due to P being in general position, the interior of T still contains at least $\epsilon n/2$ points assuming $\epsilon n \geq 12$. For $\epsilon n < 12$, the set P itself is an ϵ-Mnet . Thus we can further restrict ourselves to the interior of triangles each of whose sides contain at least two points. Figure 1 shows a triangle with each side containing two points of P. The points q and r could be identical, they could both be equal at the corner b of the triangle. Similarly s and t could be at c and u and p could be at a. Observe that the triangles *aqt*, *bsp*, *cur* and *prt* cover the triangle T and therefore one of them must contain at least $\epsilon n/4$ points of P. Each of these triangles are of the following type: at least two of the corners are in P and all sides contain at least two

points of P. We call such triangles *anchored* triangles. Thus we can again restrict ourselves to the problem of anchored triangles containing ϵn points.

Let $\mathcal{O}$ be the set of all anchored triangles. Let $\mathcal{O}' = \{o_1, \ldots, o_t\}$ be a maximal set of t triangles from $\mathcal{O}$ such that $o_i \cap P = \epsilon n$ and $|o_i \cap o_j \cap P| \leq \epsilon n/2$.

▶ Claim 5. $|\mathcal{O}'| \leq 2 \cdot f(\frac{c}{\epsilon} \cdot \log 1/\epsilon, 2c \log 1/\epsilon)$, where $f(n, l)$ is the maximum number of $\leq l$-sized subsets induced by objects in $\mathcal{O}$ given any set of n points, and c is some fixed constant.

Proof. The proof is via the probabilistic method. Pick each point of P independently at random with probability $p = c/(2\epsilon n) \cdot \log 1/\epsilon$ to get a random sample S.

▶ Fact 1. With probability at least $1/2$, the sets $o_i \cap S$, $i = 1 \ldots t$, are distinct and $|S| \leq c/\epsilon \cdot \log 1/\epsilon$.

Proof. Consider the range space $(P, \mathcal{R}')$, where $\mathcal{R}' = \{(o_i \setminus o_j) \cap P \mid \forall 1 \leq i < j \leq t\}$. First note that from the definition of $\mathcal{O}'$, we get that each set in $\mathcal{R}'$ has size at least $\epsilon n - \epsilon n/2 = \Theta(\epsilon n)$. Second, we use the fact that ranges induced by polygons with k sides have VC-dimension at most $2k + 1$ [14]; it is easy to see that $\mathcal{R}'$ is a subset of the ranges induced by polygons (or union of polygons) with at most 9 sides (overall), and so the VC-dimension of $\mathcal{R}'$ is at most 19. Then by the Haussler-Welzl theorem [10], for $c > 19 \cdot 4$, with probability at least $3/4$, S is an ϵ-net for $(P, \mathcal{R}')$. Now observe that if $o_i \cap S = o_j \cap S$, then the set $(o_i \setminus o_j) \cap S$ is empty, a contradiction to the fact that S is an ϵ-net for $\mathcal{R}'$.

Finally, from standard concentration estimates from Chernoff bounds, it follows that $|S| \geq c/\epsilon \cdot \log 1/\epsilon$ with probability at most $1/4$. ◀

For each $o_i \in \mathcal{O}'$, let X_i be the random variable which is 1 if $|o_i \cap S| \geq 2c \log 1/\epsilon$, and 0 otherwise. Then

▶ Fact 2. With probability greater than $1/2$, $\sum X_i \leq t/2$.

Proof. For a fixed i, by linearity of expectation:

$$E[|o_i \cap S|] = c/2 \cdot \log 1/\epsilon$$

By Markov's inequality applied to each X_i,

$$Pr[X_i = 1] = Pr[|o_i \cap S| \geq 2c \cdot \log 1/\epsilon] = Pr[|o_i \cap S| \geq 4 \cdot E[|o_i \cap S|]] \leq 1/4$$

Hence the expected value of $Y = \sum X_i$ is:

$$E[\sum X_i] = \sum E[X_i] = \sum Pr[X_i = 1] \leq t/4$$

By Markov's inequality applied to Y, we get that

$$Pr[\sum X_i \geq t/2] \leq E[\sum X_i]/(t/2) \leq 1/2$$

So with probability greater than $1/2$, at least half the sets of $\mathcal{O}'$ contain at most $2c \log 1/\epsilon$ points of S. ◀

Therefore, putting together Fact 1 and Fact 2, there exists a subset S of size $(c/\epsilon) \log 1/\epsilon$ such that $o_i \cap S$ are distinct for all objects in $\mathcal{O}'$, and for at least $|\mathcal{O}'|/2$ of the objects in $\mathcal{O}'$, we have $|o_i \cap S| \leq 2c \log 1/\epsilon$.

Let $f(n, l)$ be the number of distinct subsets of size at most l that can be achieved by intersection with objects in $\mathcal{O}$. These are called $\leq l$-sets (the most extensively studied case

is for halfspaces in $\mathbb{R}^d$). So in our case above, each $o_i \cap S$ formed by these $|\mathcal{O}'|/2$ objects is a $\leq l$-set of S, where $l = 2c \log 1/\epsilon$. By the bound on number of $\leq l$-sets for k-sided polygons, we get

$$|\mathcal{O}'|/2 = f(|S|, l) = f((c/\epsilon)\log 1/\epsilon, 2c \log 1/\epsilon)$$

This gives the required bound on $|\mathcal{O}'|$. ◀

Take this set $\mathcal{O}'$ of maximal objects, each containing ϵn points of P, and every pair of objects in $\mathcal{O}'$ intersecting in less than $\epsilon n/2$ points. For each object $o_i \in \mathcal{O}$, do the following: apply the simplicial partition theorem to $o_i \cap P$ with the parameter t, which is a large enough constant, to get a partition of $o_i \cap P$ into t sets of size $\Theta(|o_i \cap P|/t)$. Add each of these t sets to the ϵ-Mnet $\mathcal{U}$ for P.

▶ Claim 6. $\mathcal{U}$ is an ϵ-Mnet for the primal set-system defined by P and $\mathcal{O}$, of size $O(|\mathcal{O}'|)$.

Proof. First note that each set added to $\mathcal{U}$ had size $\Theta(|o_i \cap P|/t) = \Theta(\epsilon n)$. And the number of such sets is $O(|\mathcal{O}'| \cdot t) = O(|\mathcal{O}'|)$. It remains to show that any object containing ϵn points of P contain one set of $\mathcal{U}$.

Take any object o containing ϵn points of P. By the maximality of $\mathcal{O}'$, there exists $o_i \in \mathcal{O}'$ such that $|o \cap o_i| \geq \epsilon n/2$. Furthermore, of all the sets in the simplicial partition of o_i, each edge of ∂o can intersect only $O(\sqrt{t})$ sets; so in total the boundary of o can intersect at most $O(d\sqrt{t})$ sets. Each of these sets has $O(|o_i \cap P|/t)$ points. So these sets can contribute at most $O(d\sqrt{t} \cdot |o_i \cap P|/t)$ points of o_i to the object o. Taking $t = \alpha d$ for some large enough constant α, this is less than $\epsilon n/2$. Therefore o must contain a point in o_i which lies in a partition for o_i not intersecting ∂o, i.e., the partition lies completely inside o. ◀

▶ Claim 7. $f(n, l) \leq ln^3$.

Proof. The proof is folklore, and follows by standard application of the Clarkson-Shor method [14]. For completeness we sketch it here. An anchored triangle abc can be of two types - either all corners are in P or exactly two corners, say a and b, are in P and there is a point $p \in P$ on ac and another point $q \in P$ on bc. The number of anchored triangles of the first type is clearly at most $\binom{n}{3}$. Thus we only need to bound the number of anchored triangles of the second type with at most l points in the interior. We first consider the case when $l = 0$, i.e., anchored triangles of the second type with no point of P in the interior. For such triangles, observe that fixing the points a, b and p determines q. If there were two points q and q' then it can be easily shown that one of anchored triangles T_1 determined by a, b, p and q and T_2 determined by a, b, p and q' is non-empty - either T_1 contains q' or T_2 contains q. Thus the number of such triangles is at most $\binom{n}{3}$.

Let N denote the number of anchored triangles of the second type with at most l points in the interior. Let Q be a subset of P obtained by picking each point of P independently with probability $p = 1/l$. The expected number of empty anchored triangles of the second type determined by Q is at most the expected number of triples in Q which is $p^3\binom{n}{3}$ since every triple in P appears as a triple in Q with probability p^3. At the same time, each of the N anchored triangles with at most l points in the interior becomes an empty anchored triangles in Q with probability $p^4(1-p)^l$. Thus the expected number of empty anchored triangles in Q is at least $Np^4(1-p)^l$. Thus $Np^4(1-p)^l \leq p^3\binom{n}{3}$. Since $p = 1/l$, it follows that $N = O(ln^3)$. ◀

Triangles and k-sided polygons. Finally, the proof for the size of ϵ-Mnet for triangles and k-sided polygons follows from Claims 5, 6 and 7. We now sketch the proof of the other cases along the above lines.

Lines, strips, cones. For sets induced by lines, strips, cones in the plane, one can follow the above proof. The function $f(n,l)$ correspondingly denotes the number of subsets of size l induced by the objects of the appropriate type (lines, strips, cones). For lines, $f(n,l) = O(n^2)$, for strips it is $O(n^2 l)$ and for cones it is $O(n^2 l^2)$. The proof then follows from the above claims.

Rectangles intersecting a common line l. As each rectangle contains ϵn points and intersects l, for each rectangle R, take the portion of the rectangle on the side of l that contains at least $\epsilon n/2$ points. We can construct $\epsilon n/2$-Mnets for the two sides of l separately.

So consider the rectangles anchored on l lying on the same side containing $\epsilon n/2$ points of P. Call this set $\mathcal{O}$. As before, let $\mathcal{O}'$ be the maximal subset of $\mathcal{O}$ such that i) every pair of rectangles in $\mathcal{O}'$ share at most $\epsilon n/10$ points, and ii) each rectangle contains $\epsilon n/2$ points. These form *pseudo-disks* (i.e., no two rectangles pierce each other) and by the result of [18], $|\mathcal{O}'| = O(1/\epsilon)$. Now Claim 6 implies that one can construct $\epsilon/2$-Mnet of size $O(1/\epsilon)$.

Disks. By standard Veronese map, points P and disks D can be lifted to halfspaces H in $\mathbb{R}^3$ such that each point is lifted to a point in $\mathbb{R}^3$ and each disk is lifted to a halfspace in $\mathbb{R}^3$ in such a way that their incidences are preserved. Now the required bound follows from the result for halfspaces in $\mathbb{R}^3$.

Lower-bounds

▶ **Theorem 6.** *For every $\epsilon > 0$ and k an integer, there exists a set P of n points in the plane, and a set $\mathcal{D}$ of $\Omega(\frac{1}{\epsilon^{d+1}})$ curves of degree at most d, such that i) each curve contains ϵn points of P and, ii) no two curves share more than $\epsilon n/k$ points of P.*

Proof. For the lower-bound on the size of $\frac{1}{k}$ heavy ϵ-$Mnet$, consider the grid $G = [dk] \times [\frac{1}{\epsilon}]$ in the plane for some $d \geq 1$, where $[r]$ denotes the set $\{0, \cdots, \lceil r \rceil - 1\}$. Now, consider univariate functions of x of the form $y = \sum_{i=0}^{d} a_i x^i$ where each a_i is an integer in $[\frac{1}{\epsilon(d+1)(dk)^i}]$. Clearly there are at least $\Omega(\prod_{i=0}^{d} \frac{1}{\epsilon(d+1)(dk)^i}) = \Omega(\frac{1}{\epsilon^{d+1}})$ of these polynomials. Since for each value of $x \in [dk]$, the value of y is in $[\frac{1}{\epsilon}]$, each of these curves contain dk points of G. Also, since these are curves of degree at most d, no two intersect in more than d points. Let P be the set of n points containing $\epsilon n/dk$ copies of each of the points in G. We thus get a set of $\Omega(\frac{1}{\epsilon^{d+1}})$ curves of degree at most d, each of which contain ϵn points of P and no two of which share more than $\epsilon n/k$ points of P. ◀

▶ **Corollary 7.** *This gives a lower bound of $\Omega(\frac{1}{\epsilon^{d+1}})$ for $\frac{1}{k}$-heavy ϵ-$Mnets$ for range spaces induced by curves of degree at most d in the plane.*

Note that this immediately implies that for sets induced by lines in the plane, ϵ-Mnets must have size $\Omega(1/\epsilon^2)$. Which in turn is a special case for strips and cones in the plane.

▶ **Corollary 8.** *Any ϵ-Mnet for sets induced by lines, strips and cones in the plane must have size $\Omega(1/\epsilon^2)$.*

Finally, using standard linearization [14] (with Veronese maps), it is possible to lift a set of polynomial curves of degree d and a set of points to R^{3d+2} so that each point in the plane is lifted to a point in R^{3d+2} and each curve is lifted to a halfspace (i.e., the curve $y = f(x)$ becomes $(y - f(x))^2 \leq 0$, and each monomial of this expansion can be treated as a different coordinate for the linearization). Thus we have the following:

▶ **Corollary 9.** *Any ϵ-Mnet for sets induced by halfspaces in $\mathbb{R}^d$ must have size $\Omega(\frac{1}{\epsilon^{\lceil (d-1)/3 \rceil}})$.*

References

1. B. Aronov, E. Ezra, and M. Sharir. Small-size ϵ-nets for axis-parallel rectangles and boxes. *SIAM J. Comput.*, 39(7):3248–3282, 2010.
2. S. Arya, G. Dias da Fonseca, and D. M. Mount. Optimal area-sensitive bounds for polytope approximation. In *Symposium on Computational Geometry*, pages 363–372, 2012.
3. I. Barany. Random polytopes, convex bodies, and approximation. In W. Weil, editor, *Stochastic Geometry*, pages 77–118. Springer, 2007.
4. I. Barany and D. G. Larman. Convex bodies, economic cap coverings, random polytopes. *Mathematika*, 35:274—-291, 1988.
5. H. Bronnimann, B. Chazelle, and J. Pach. How hard is halfspace range searching. *Discrete Comput. Geom.*, 10:143—155, 1993.
6. T. M. Chan, E. Grant, J. Könemann, and M. Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *SODA*, 2012.
7. C. Chekuri, K. L. Clarkson, and S. Har-Peled. On the set multi-cover problem in geometric settings. In *Symposium on Computational Geometry*, pages 341–350, 2009.
8. K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. In *Symposium on Computational Geometry*, pages 135–141, 2005.
9. G. Ewald, D. G. Larman, and C. A. Rogers. The directions of the line segments and of the r-dimensional balls on the boundary of a convex body in euclidean space. *Mathematika*, 17:1—-20, 1970.
10. D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.
11. J. Komos, J. Pach, and G. Woeginger. Almost tight bounds for epsilon nets. *Discrete & Computational Geometry*, pages 163–173, 1992.
12. A. M. Macbeath. A theorem on non-homogeneous lattices. *Annals of Math*, 56:269—-293, 1952.
13. J. Matousek. Reporting points in halfspaces. *Computational Geometry*, 2(3):169 – 186, 1992.
14. J. Matousek. *Lectures in Discrete Geometry*. Springer-Verlag, New York, NY, 2002.
15. N. H. Mustafa and S. Ray. An optimal generalization of the centerpoint theorem, and its extensions. In *Symposium on Computational Geometry*, pages 138–141, 2007.
16. N. H. Mustafa and S. Ray. Weak ϵ-nets have a basis of size $O(1/\epsilon \log 1/\epsilon)$. *Comp. Geom: Theory and Appl.*, 40(1):84–91, 2008.
17. J. Pach and G. Tardos. Tight lower bounds for the size of epsilon-nets. In *Symposium on Computational Geometry*, pages 458–463, 2011.
18. E. Pyrga and S. Ray. New existence proofs epsilon-nets. In *Symposium on Computational Geometry*, pages 199–207, 2008.
19. S. Ray. *Weak and strong ϵ-nets for Geometric Range Spaces*. PhD thesis, Saarland University, 2009.
20. K. Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 641–648, 2010.

Non-autoreducible Sets for NEXP

Dung T. Nguyen and Alan L. Selman

University at Buffalo, The State University of New York, NY, US
{dtn3,selman}@buffalo.edu

Abstract

We investigate autoreducibility properties of complete sets for NEXP under different polynomial-time reductions. Specifically, we show that under some polynomial-time reductions there are complete sets for NEXP that are not autoreducible. We show that settling the question whether every $\leq^p_{dtt}$-complete set for NEXP is $\leq^p_{NOR\text{-}tt}$-autoreducible either positively or negatively would lead to major results about the exponential time complexity classes.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Autoreducibility, NEXP, diagonalization, structural complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.590

1 Introduction

Autoreducibility was first introduced by Trakhtenbrot [11]. A set A is autoreducible if A is reducible to A via an oracle Turing machine M such that M never queries x on input x. Ambos-Spies [1] introduced the polynomial-time variant of autoreducibility, where the oracle Turing machine now runs in polynomial time. Each notion of polynomial-time reduction induces the corresponding notion of autoreducibility.

The main question that has drawn many researchers' attention is whether complete sets for various complexity classes are polynomial-time autoreducible. Over many years, many results about autoreducibility of complete sets of different classes have been discovered. Glaßer et al. [7] showed that all m-complete sets of the following complexity classes are many-one autoreducible: NP, PSPACE, EXP, NEXP, $\Sigma^{\mathrm{P}}_{\mathrm{k}}$, $\Pi^{\mathrm{P}}_{\mathrm{k}}$, and $\Delta^{\mathrm{P}}_{\mathrm{k}}$ for $k \geq 1$. Beigel and Feigenbaum [10] showed that all Turing complete sets for any class $\Sigma^{\mathrm{P}}_{\mathrm{k}}$, $\Pi^{\mathrm{P}}_{\mathrm{k}}$, $\Delta^{\mathrm{P}}_{\mathrm{k}}$, $k \geq 0$, are Turing autoreducible. Also, all Turing complete sets for NP are Turing autoreducible.

Resolving some open questions about autoreducibility would lead to major class separation results. Buhrman et al. [2] proved various autoreducibility results for many different complexity classes and demonstrated strong evidence that studying structural properties of the complete sets, especially the autoreducibility property, might be an important tool to separate complexity classes. For example, if there exists a Turing complete set of NEXP that is not Turing autoreducible, then EXP is different from NEXP.

We reinforce this belief with the following result. Let hypothesis A be the assertion that every $\leq^p_{dtt}$-complete set for NEXP is $\leq^p_{NOR\text{-}tt}$-autoreducible. We prove that hypothesis A is true if and only if NEXP = coNEXP. It follows immediately that $\neg A$ implies NEXP $\neq$ EXP. We see that settling hypothesis A either positively or negatively solves important problems about these classes.

With this motivation in mind, we study autoreducibility questions for NEXP. Buhrman et al. [2] extensively studied autoreducibility for EXP. It is known that under many-one, 1-tt, 2-tt, and Turing reductions, all complete sets for EXP are autoreducible. Also for any $k \geq 3$, under $\leq^p_{k\text{-}tt}$-reduction, there exists a complete set for EXP that is not autoreducible. For NEXP, it is known that all many-one complete sets are autoreducible. Moreover, Glaßer



© Dung T. Nguyen and Alan L. Selman;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 590–601



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



et al. [6] took a next step to show that under 2-tt, disjunctive-truth-table, and conjunctive-truth-table reductions, all complete sets for NEXP are autoreducible. We make progress in this paper by proving non-autoreducibility of complete sets for NEXP under certain polynomial-time reductions. In particular, we obtain the following results. (All definitions to follow.)

- For any positive integers s and k such that $2^s - 1 > k$, there is a $\leq^p_{s\text{-}T}$-complete set for NEXP that is not $\leq^p_{k\text{-}tt}$-autoreducible.
- There is a $\leq^p_T$-complete set for NEXP that is not $\leq^p_{tt}$-autoreducible.
- There is a $\leq^p_{3\text{-}tt}$-complete set for NEXP that is not honest $\leq^p_{3\text{-}tt}$-autoreducible.
- For any positive integer k, there is a $\leq^p_{k\text{-}tt}$-complete set for NEXP that is not weakly $\leq^p_{k\text{-}tt}$-autoreducible.

Proofs typically require intricate diagonalization arguments.

This paper is organized as follows. Section 2 contains notation and definitions about many different polynomial-time reductions and autoreducibilities. In section 3, we obtain our non-autoreducibility results for many different complete sets in NEXP. Section 4 contains our result about hypothesis A. In section 5, we will show negative results in relativized worlds for some open questions.

2 Preliminaries

Most notation and definitions are standard [9]. Strings are elements of $\{0,1\}^*$. For every string x, denote $|x|$ to be the length of x. For every Turing machine M, $L(M)$ denotes the language accepted by the machine M. We denote M^B to be an oracle Turing machine M that accesses the oracle B. Also for every input x, $M(x)$ is the outcome of the computation of M on input x; i.e., $M(x) = 1$ if and only if M accepts input x. We assume that the pairing function $\langle \ldots \rangle$ is a one-to-one, polynomial-time computable function that can take any finite number of inputs and its range does not intersect with 0^*. For every set A, the characteristic function of A is denoted by A; that is, $A(x) = 1$ if $x \in A$ and $A(x) = 0$ otherwise. Also $|A|$ denotes the cardinality of A.

For any two sets A and B, A is *Turing-reducible* to B *in polynomial time*, $A \leq^p_T B$, if there exists a deterministic polynomial-time-bounded oracle Turing machine M such that $A = L(M^B)$. Similarly, $A \leq^p_{k\text{-}T} B$ if there exists a deterministic polynomial-time-bounded oracle Turing machine M such that $A = L(M^B)$ and M asks no more than k queries for any input x. In this paper, if we do not mention explicitly the running time of a reduction, then that reduction is a polynomial time reduction. The reduction is *nonadaptive*, $A \leq^p_{tt} B$, if the queries are independent of the oracle and so they do not depend on the answers to the previous queries. Other notions of reductions are also considered. A set A is *k-truth-table-reducible* to B, $A \leq^p_{k\text{-}tt} B$, if there exists a nonadaptive oracle Turing machine M^B that accepts A such that for any input x, the computation of M^B on input x asks no more than k queries. A set A is *bounded-truth-table-reducible* to B, $A \leq^p_{btt} B$, if there exists some integer k such that $A \leq^p_{k\text{-}tt} B$. A set A is *disjunctive-truth-table reducible* to B in polynomial time, $A \leq^p_{dtt} B$, if there exists a polynomial time computable function f such that for any x, $f(x) = \langle q_1, \ldots, q_k \rangle$, and $x \in A \iff B(q_1) \vee \cdots \vee B(q_k) = 1$. Similarly, a set A is *conjunctive-truth-table reducible* to B in polynomial time, $A \leq^p_{ctt} B$, if there exists a polynomial time computable function f such that for any x, $f(x) = \langle q_1, \ldots, q_k \rangle$, and $x \in A \iff B(q_1) \wedge \cdots \wedge B(q_k) = 1$. Other notions $\leq^p_{k\text{-}dtt}$ and $\leq^p_{k\text{-}ctt}$ are defined analogously. For any k-ary Boolean function α, a set A is *α-truth-table reducible* to B in polynomial

time, $A \leq^p_{\alpha tt} B$, if there exists a polynomial time computable function f such that for any x, $f(x) = \langle q_1, \ldots, q_k \rangle$, and $x \in A \iff \alpha(B(q_1), \ldots, B(q_k)) = 1$.

EXP $= \bigcup\{\text{DTIME}(2^{p(n)}) \mid p \text{ is a polynomial}\}$ is the class of languages that can be decided by a deterministic Turing machine in exponential time.

NEXP $= \bigcup\{\text{NTIME}(2^{p(n)}) \mid p \text{ is a polynomial}\}$ is the class of languages that can be decided by a nondeterministic Turing machine in exponential time.

Throughout this paper, let $\{\text{NEXP}_i\}_{i \geq 1}$ be an enumeration of all nondeterministic exponential time Turing machines. Also we assume that the computation of NEXP_j on input x has running time that is bounded by $2^{|x|^j}$.

Let $K = \{\langle i, x, l \rangle \mid \text{NEXP}_i \text{ accepts input } x \text{ within } l \text{ steps}\}$ be a canonical complete set for NEXP, where l is encoded in a binary string.

For any oracle Turing machine M^B, let $Q(M^B, x)$ denote the set of all queries of the computation of M^B on input x.

▶ **Definition 1** (Autoreducibility)**.** For any reduction $\leq$, a set A is $\leq$-autoreducible if $A \leq A$ via an oracle Turing machine M^A such that for any x, $x \notin Q(M^A, x)$. We call M an autoreduction of A by $\leq$-reduction. The reduction $\leq$ can apply to any reductions, specifically, all those that we mention above, such as $\leq^p_T, \leq^p_{tt}, \leq^p_{k\text{-}tt}, \leq^p_{dtt}$, etc.

Honest reductions are discussed in [8] and [5]. Informally, in honest reductions, the strings queried to the oracle cannot be too short compared to the input length. In this paper, we use a stronger notion of honest reductions, where strings queried cannot be either too short or too long compared to the input length. Its formal definition is as follows.

▶ **Definition 2** (Honest truth-table reduction)**.** Given any two sets A and B and an arbitrary positive number $c \geq 1$, we define an honest truth-table reduction $\leq^{h\text{-}c}_{tt}$ as follows: $A \leq^{h\text{-}c}_{tt} B$ if there exists a nonadaptive Turing machine M with oracle B such that M^B accepts x if and only if $x \in A$ and for any input x, all queries q made to oracle B have length satisfying $|x|^{1/c} \leq |q| \leq |x|^c$.

▶ **Definition 3** (NOR-reduction)**.** Given any two sets A and B, we define a NOR-truth-table reduction $\leq^p_{NOR\text{-}tt}$ as follows: $A \leq^p_{NOR\text{-}tt} B$ if there exists a nonadaptive Turing machine M with oracle B such that for any input x, letting $q_1, \ldots, q_k$ be all queries of M^B on input x, then $x \in A \iff q_1 \notin B \wedge \cdots \wedge q_k \notin B$.

▶ **Definition 4** (Weak-reduction)**.** Given any two sets A and B, we define a weak truth-table reduction $\leq^p_{tt\text{-}w}$ as follows: $A \leq^p_{tt\text{-}w} B$ if and only if there exist two polynomial computable functions f and g such that for any input x, $f(x) = \langle q_1, \ldots, q_k \rangle$, $g(x) = h(\alpha_1, \ldots \alpha_k)$ is a Boolean function with k variables $\alpha_1, \ldots \alpha_k$ such that h is neither an OR nor a NOR Boolean function, and $x \in A \iff h(B(q_1), \ldots, B(q_k)) = 1$.

3 Non-autoreducible sets for NEXP

▶ **Theorem 5.** *For any positive integers s and k such that $2^s - 1 > k$, there is a $\leq^p_{s\text{-}T}$-complete set for* NEXP *that is not $\leq^p_{k\text{-}tt}$-autoreducible.*

Proof. Let $\{M_j\}_{j \geq 1}$ be an enumeration of all $\leq^p_{k\text{-}tt}$-reductions. Assume that M_j on input x runs in time $|x|^j$. We will construct a set B such that $K \leq^p_{s\text{-}T} B$ but B is not $\leq^p_{k\text{-}tt}$-autoreducible. Recall that K, which is defined in the Preliminaries section, is a canonical complete set for NEXP.

The $\leq^p_{s\text{-}T}$-reduction from K to B will be as follows: we build a full binary tree of height s. This tree has exactly $2^s - 1$ nodes. We number the nodes from top to bottom, left to right,

by using numbers $0, 1, \ldots, 2^s - 2$; i.e. the root node will be numbered 0, then its two children will be 1 and 2, etc. Then for any string x, each node i will be labeled by the pair $\langle x, i\rangle$. From now on, for every such x, $\mathcal{T}(x)$ is such a query tree, and for every node $\mathcal{N}$, $\mathcal{N}$ is referred as a node itself or its label interchangeably. Also for any two nodes $\mathcal{N}_1$ and $\mathcal{N}_2$ such that one node is an ancestor of another node, denote $\mathcal{P}(\mathcal{N}_1, \mathcal{N}_2)$ to be a unique path from $\mathcal{N}_1$ to $\mathcal{N}_2$. For every node $\mathcal{N}$, denote the left path $\mathcal{L}(\mathcal{N})$ to be a path from $\mathcal{N}$ to a leaf node by just traversing left. The right path $\mathcal{R}(\mathcal{N})$ is defined similarly. Those labels are possible queries that can be asked to the oracle B by this reduction. Specifically, start at the root node, and if the current query is node $\mathcal{N}$, if the answer is YES, i.e. $\mathcal{N} \in B$, then the next query will be $\mathcal{N}$'s left child; otherwise the right child will be asked. The reduction accepts if and only if the last query (certainly, it is one of the leaf nodes) belongs to B. Define the sequence $\{y_n\}_{n\geq 0}$ such that $y_0 = 1$ and $y_{n+1} = 2^{y_n^n} + 1$ for every $n \geq 0$. Now we construct such a set B that satisfies the above reduction. At the same time, we want to diagonalize against all M_n^B such that M_n^B accepts 0^{y_n} if and only if $0^{y_n} \notin B$. The set B is constructed in each stage as follows. Initially we set $B = \emptyset$.

At stage n, suppose that the set B has been constructed such that all strings of length up to y_{n-1}^{n-1} have already been encoded into B appropriately to make the above reduction work. We will encode all strings of length between $y_{n-1}^{n-1} + 1$ and y_n^n into B in this stage.

Compute Q that is the set of all queries q of M_n on input 0^{y_n} such that $q = \langle x, i\rangle$, $i \leq 2^s - 1$, and $y_{n-1}^{n-1} + 1 \leq |x| \leq y_n^n$. Denote P to be the set of all x such that $\langle x, i\rangle \in Q$ for some $0 \leq i \leq 2^s - 1$. And for each $x \in P$, denote P^x to be the set of all $\langle x, i\rangle$ such that $\langle x, i\rangle \in Q$ and $0 \leq i \leq 2^s - 1$.

For each x in P, consider set P^x. Notice that $|P^x| \leq k < 2^s - 1$. Consider the query tree $\mathcal{T}(x)$:

- **Case 1:** If all leaf nodes are in P^x, then there are some internal nodes such that they are not in P_x. Let $\mathcal{N}$ be the smallest node in the set of those nodes. Put $\mathcal{N}$ into B if and only if $x \in K$. Also for every node $\mathcal{N}'$ in $\mathcal{L}(\mathcal{N})$ and $\mathcal{N}' \neq \mathcal{N}$, add $\mathcal{N}'$ to B. Finally for every node $\mathcal{N}'$ in the path $\mathcal{P}(Root, \mathcal{N})$, add $\mathcal{N}'$ to B if and only if its left child node is in the path.
- **Case 2:** If there are some leaf nodes that are not in P^x, let $\mathcal{N}$ be the smallest node in the set of those nodes. Add $\mathcal{N}$ to B if and only if $x \in K$. For every node $\mathcal{N}'$ in $\mathcal{P}(Root, \mathcal{N})$, add $\mathcal{N}'$ to B if and only if its left child is in that path.

For every $x \notin P$ such that $y_{n-1}^{n-1} + 1 \leq |x| \leq y_n^n$, put $\langle x, 2^s - 1\rangle$ into B if and only if $x \in K$.

After all those steps are done, put 0^{y_n} into B if and only if M_n^B rejects 0^{y_n}.

That is how B is constructed. It is straightforward to see that the construction satisfies two properties: $K \leq_{s\text{-}T}^p B$ and B is not $\leq_{k\text{-}tt}^p$-autoreducible.

▶ **Claim 6.** $B \in \mathrm{NEXP}$

Proof. Notice that all elements of B have one of two forms 0^* and $\langle x, i\rangle$ where $0 \leq i \leq 2^s - 1$. For any input of any other form, it just rejects immediately.

Given an input b, consider the following cases:

- $b = 0^{y_n}$ for some n (otherwise, $b \notin B$). Then by the construction,

$$0^{y_n} \in B \iff M_n^B \text{ rejects } 0^{y_n}.$$

So if we know how to resolve all queries made to oracle B then it is easy to determine whether M_n^B accepts 0^{y_n} in exponential time. Now notice that in the above construction, for every query q, it can be resolved by considering the query tree and it does not depend

on the membership of some x in K. In this case membership in B can be answered deterministically in exponential time.

- $b = \langle x, i\rangle$ for some $0 \le i \le 2^s - 1$. By considering the query tree $\mathcal{T}(x)$, there are two cases:
  - The membership of b in B can be determined straightforwardly, based on the above construction, and does not depend on whether $x \in K$ or not.
  - $b \in B \iff x \in K$. In this case, we can simulate the machine to accept K on an input x. Notice that $|x| < |b|$, so it can be done nondeterministically in exponential time.

Thus, $B \in$ NEXP ◀

Hence, B is a $\le^p_{s\text{-}T}$-complete set for NEXP that is not $\le^p_{k\text{-}tt}$-autoreducible. ◀

Glaßer et al. [6] showed that every $\le^p_{2\text{-}tt}$-complete set for NEXP is $\le^p_{2\text{-}tt}$-autoreducible. Theorem 5 is somehow "tight" in case $s = 2$ and $k = 2$. The following corollary separates the notions of $\le^p_{2\text{-}T}$ and $\le^p_{2\text{-}tt}$.

▶ **Corollary 7.** *There is a $\le^p_{2\text{-}T}$-complete set for* NEXP *that is not $\le^p_{2\text{-}tt}$-complete.*

It has been known that there is a Turing complete set for EXP that is not $\le^p_{tt}$-autoreducible [4]. We want to remark that Buhrman et al. [3] showed that there is a set that is Turing complete but not $\le^p_{tt}$-complete for NEXP. Moreover, their construction technique can be adapted to show that for any positive integers s and k such that $2^{s-2} > k$, there is a $\le^p_{s\text{-}T}$-complete set for NEXP that is not $\le^p_{k\text{-}tt}$-autoreducible, which is weaker than what Theorem 5 states. By adding a minor trick to the proof in Theorem 5 or cleverly adapting the technique in [3], we can separate the Turing-completeness notion from the $\le^p_{tt}$-autoreducibility notion in NEXP, as opposed to Turing-completeness versus $\le^p_{tt}$-completeness in [3].

▶ **Corollary 8.** *There is a Turing complete set for* NEXP *that is not $\le^p_{tt}$-autoreducible.*

Proof. Notice that in this case, the M_n autoreduction will not ask just k queries on input 0^{y_n}, but it can ask up to y_n^n queries, because its running time on input 0^{y_n} is bounded by y_n^n. Another modification is that the reduction from K to B will now need to ask more queries, say $|x|^2$ adaptive queries; that also means the query tree will have height $|x|^2$. With this trick in mind, in the construction algorithm of B at stage n, for every x in P, $|P^x| \le y_n^n = (2^{y_{n-1}^{n-1}} + 1)^n < 2^{y_{n-1}^{2(n-1)}} < 2^{|x|^2}$. So the number of nodes in the query tree $\mathcal{T}(x)$ will be bigger than the number of queries of M_n on input 0^{y_n}. In cases 1 and 2 the construction will work similarly. ◀

Now we consider the more difficult question of whether every $\le^p_{3\text{-}tt}$-complete set for NEXP is $\le^p_{3\text{-}tt}$-autoreducible. Notice that the above technique cannot be used, because the number of options to encode every x in K into B is no more than the number of queries of M_n^B on input 0^{y_n}; both are equal 3 in this case. This difficulty arises because we have no "room" for the encoding and diagonalization at the same time. We need to use a different technique to resolve that issue.

▶ **Theorem 9.** *For any number c, there is a $\le^p_{2\text{-}T}$-complete set for* NEXP *that is not $\le^{h\text{-}c}_{3\text{-}tt}$-autoreducible.*

Proof. Let $\{M_i\}_{i\ge1}$ be a standard enumeration of all $\le^{h\text{-}c}_{3\text{-}tt}$-autoreductions clocked such that M_i runs in time n^i. We will construct a $\le^p_{2\text{-}T}$-complete set B for NEXP incrementally in each stage and diagonalize against all autoreductions M_i. We define the sequence $\{y_n\}_{n\ge1}$ recursively as follows: $y_1 = 1$ and $y_{n+1} = \max(y_n^n, y_n^{c^2}) + 1$ for all $n \ge 1$.

In each stage, we construct B such that the following procedure is the $\leq^p_{2\text{-}T}$-reduction that reduces K to B. Given any input x, ask a query 0^m to oracle B, where m is a number that is bounded by some polynomial in $|x|$. If the answer is YES, then accept x if and only if $\langle 0, x\rangle \in B$. If the answer is NO, then accept x if and only if $\langle 1, x\rangle \in B$. Obviously if B satisfies this condition, then B is $\leq^p_{2\text{-}T}$-hard for NEXP.

The detail of how B is constructed will be as follows.

- Initially $B = \emptyset$.
- Suppose at stage n, the set B is constructed up to length $y_n - 1$. At this stage, we will add appropriate strings of length between y_n and $y_{n+1} - 1$ to accomplish two things: encoding K into B and diagonalize, using the string $0^{y_n^c}$, against the autoreduction M_n that asks no more than 3 queries. Therefore, in the following steps, if M_n asks more than 3 queries, then the diagonalization task will be skipped to the next stage.

Consider the following case where queries of M_n^B on input $0^{y_n^c}$ are $\langle 0, q_1\rangle$, $\langle 1, q_2\rangle$, and $\langle 1, q_3\rangle$ and the Boolean truth-table function is $f(a, b_1, b_2)$. In other words, M_n^B accepts $0^{y_n^c}$ if and only if $f(B(\langle 0, q_1\rangle), B(\langle 1, q_2\rangle), B(\langle 1, q_3\rangle)) = 1$. (Lack of space does not permit a complete proof of this Theorem.)

▶ **Lemma 10.** *For any Boolean function $f(a, b_1, b_2)$, at least one of the following statements must be true:*

- *There exist two Boolean functions $g_1(a)$ and $g_2(a)$, where $g_1(a)$ and $g_2(a)$ are one of $a, 0$, or 1, such that $f(a, g_1(a), g_2(a)) = 0$ for every a.*
- *There exists a Boolean function $h(b_1, b_2)$, where $h(b_1, b_2)$ is one of $0, 1, b_1, b_2, b_1 \wedge b_2$, or $b_1 \vee b_2$, such that $f(h(b_1, b_2), b_1, b_2) = 1$ for every b_1 and b_2.*

Suppose that we have $f(b_1 \vee b_2, b_1, b_2) = 1$, for every b_1 and b_2 (in this case, we are considering Statement 2 in the above lemma). Then if we set $B(\langle 1, q_2\rangle) = 1$ if $q_2 \in K$, $B(\langle 1, q_3\rangle) = 1$ if $q_3 \in K$, and $B(\langle 0, q_1\rangle) = B(\langle 1, q_2\rangle) \vee B(\langle 1, q_3\rangle)$. Also $B(0^{y_n^c}) = 0$. It is easy to verify that $0^{y_n^c} \notin B$ and M_n^B accepts $0^{y_n^c}$. So the diagonalization can be achieved by this fact.

Moreover by this setting, the reduction $K \leq^p_{2\text{-}T} B$ can be obtained correctly too. Notice that $0^{y_n^c} \notin B$. Thus, $q_2 \in K$ if and only if $\langle 1, q_2\rangle \in B$. Similarly for $\langle 1, q_3\rangle$. This fact is correctly reflected in the above setting.

Last but not least, we need B to be in NEXP. Consider whether $\langle 1, q_2\rangle \in B$. Notice that it is equivalent to the question whether $q_2 \in K$, which can be solved nondeterministically in exponential time. A more difficult question is whether $\langle 0, q_1\rangle$ is in B. By B's construction, $B(\langle 0, q_1\rangle) = B(\langle 1, q_2\rangle) \vee B(\langle 1, q_3\rangle)$. By this fact, $\langle 0, q_1\rangle$ is in B if one of the two strings q_2 and q_3 is in B. This condition can also be solved nondeterministically in exponential time. In summary, B is in NEXP.

By our construction we obtain three properties: $\leq^p_{2\text{-}T}$-hardness of B, B is in NEXP, and B is not autoreducible. That is, B is the set that we want to construct to prove this theorem. ◀

We note that Lemma 10 cannot be generalized to a Boolean function of 4 variables a_1, a_2, b_1, b_2 or more because we found a counterexample in that case. We obtained the counterexample by writing a program to list all possible Boolean functions of 4 variables, and then for each function checking whether it satisfies the two statements in Lemma 10. So the proof of Theorem 9 cannot be generalized to work with $\leq^p_{k\text{-}tt}$-reductions for $k \geq 4$. Nevertheless, the following theorem will show non-autoreducibility for $\leq^p_{k\text{-}tt}$-reductions if we

reduce the power of the $\leq^p_{k\text{-}tt}$-autoreduction by not allowing the truth-table function to be an OR or a NOR.

▶ **Theorem 11.** *For any positive integer k, there is a $\leq^p_{k\text{-}tt}$-complete set for* NEXP(EXP) *that is not weakly $\leq^p_{k\text{-}tt\text{-}w}$-autoreducible.*

Proof. Let $\{M_j\}_{j\geq 1}$ be an enumeration of polynomial-time weak $\leq^p_{k\text{-}tt}$-autoreductions. For each $j \geq 1$, assume that M_j on input x runs in time $|x|^j$. Denote $\alpha_1, \ldots, \alpha_k$ to be the lexicographically first k strings of length $\lceil \log k \rceil$. We will construct a set B with the following property: $x \in K \iff$ there exists a j, $1 \leq j \leq k$, and $\langle \alpha_j, x\rangle \in B$, which ensures that $K \leq^p_{k\text{-}tt} B$, and then B is $\leq^p_{k\text{-}tt}$-hard for NEXP. We also need B so that for any $n \geq 1$, the following property holds: $0^{y_n} \in B \iff M_n^B$ rejects input 0^{y_n}, which ensures that M_n is not an autoreduction of B. (The value of y_n will be chosen later in the proof) Then we can conclude that B is not autoreducible.

We construct B in stages. In each stage, we will encode K into B and diagonalize against all weak $\leq^p_{k\text{-}tt}$-reductions using the string 0^{y_n} simultaneously to obtain those above two properties.

Before going into detail of how B is constructed, we define the sequence $\{y_n\}_{n\geq 0}$ such that $y_0 = 1$ and $y_{n+1} = 2^{y_n^n} + 1$ for every $n \geq 0$. B is constructed in each stage as follows.

Initially we set $B = \emptyset$. At stage n, suppose that B is already constructed up to strings of length y_{n-1}^{n-1}. We will encode appropriately all strings of length between $y_{n-1}^{n-1} + 1$ and y_n^n into B.

Let Q be the set of all queries q of M_n on input 0^{y_n} such that $|q| > y_{n-1}^{n-1}$. Let $P = \{x \mid$ there exists a $1 \leq j \leq k$ such that $\langle \alpha_j, x\rangle \in Q\}$. For every $x \in P$, denote $P^x = \{\langle \alpha_j, x\rangle \mid \langle \alpha_j, x\rangle \in Q\}$.

Now we consider the following cases:

- If $|P^x| < k$ for all x, then for every $x \in P$, denote t to be the smallest number such that $\langle \alpha_t, x\rangle \notin Q$. Put $\langle \alpha_t, x\rangle$ into B if and only if $x \in K$. Also for every $x \notin P$, put $\langle \alpha_1, x\rangle$ into B if and only if $x \in K$. Finally, put 0^{y_n} into B if and only if M_n^B rejects 0^{y_n}.
- If $|P^x| = k$ for some x, consider the Boolean truth-table function g of M_n on input 0^{y_n}, we have two following cases:
  - If $g(0, 0, \ldots, 0) = 0$, then let $c_1, \ldots, c_k$ be the lexicographically smallest non-zero value such that $g(c_1, \ldots, c_k) = 0$. For every c_i such that $c_i = 1$, put $\langle \alpha_i, x\rangle$ into B if and only if $x \in K$. Also for every $x \notin P$, put $\langle \alpha_1, x\rangle$ into B if and only if $x \in K$. Finally put 0^{y_n} into B.
  - If $g(0, 0, \ldots, 0) = 1$, then let $c_1, \ldots, c_k$ be the lexicographically smallest non-zero value such that $g(c_1, \ldots, c_k) = 1$. For every c_i such that $c_i = 1$, put $\langle \alpha_i, x\rangle$ into B if and only if $x \in K$. Also for every $x \notin P$, put $\langle \alpha_1, x\rangle$ into B if and only if $x \in K$.

This concludes the construction of B. The following lemma claims the time complexity of B.

▶ **Lemma 12.** $B \in$ NEXP.

Proof. Given an input b, one of the following cases can happen:

- **Case 1:** If b has the form 0^*: if $|b| \neq y_n$ for all n then reject. Otherwise, compute the set Q of all queries when running a Turing machine M_n^B on input 0^{y_n}. Notations of P and P^x are defined similarly to B's construction above.
  - If $|P^x| < k$ for all x, then simulate the Turing machine M_n^B on input 0^{y_n}. Whenever a query q is asked, the answer from oracle B will be resolved as follows:
    * If $|q| > y_{n-1}^{n-1}$, then answer NO.
    * Otherwise, check whether $q \in B$ recursively.

- If $|P^x| = k$ for some x. Then let g be a Boolean truth-table function of M_n^B on input 0^{y_n}.
  - If $g(0,\ldots,0) = 0$ then accept.
  - Otherwise, reject.

- **Case 2:** $b = \langle \alpha_i, x\rangle$ for some α_i (if $b \neq \langle \alpha_j, x\rangle$ for all j then just reject)
  Compute the number n such that $y_{n-1}^{n-1} < |b| \leq y_n^n$.
  Consider sets Q, P, and P^x as above when running M_n^B on input 0^{y_n}. We have the following cases:
  - If $|P^y| < k$ for every $y \in P$: If $b \in Q$ then reject. Otherwise, accept if and only if $i = 1$ and $x \in K$.
  - If $|P^y| = k$ for some $y \in P$: If $x \neq y$ then accept if and only if $i = 1$ and $x \in K$. Otherwise, if $x = y$, let g be a Boolean truth-table function of M_n on input 0^{y_n}. Consider the two following cases:
    - If $g(0,\ldots,0) = 0$. Let $c_1,\ldots,c_k$ be the lexicographically smallest non-zero value such that $g(c_1,\ldots,c_k) = 0$. Accept if and only if $c_i = 1$ and $x \in K$.
    - If $g(0,\ldots,0) = 1$. Let $c_1,\ldots,c_k$ be the lexicographically smallest non-zero value such that $g(c_1,\ldots,c_k) = 1$. Accept if and only if $c_i = 1$ and $x \in K$.

Now we will analyze the running time of the above tasks. The most expensive tasks will be described as follows:

- The number n can be determined in polynomial time in terms of length of input b.
- The query set Q and the truth-table function g can be computed in time y_n^n, which is no more than $O(2^{|b|^2})$.
- In case 1, to recursively check whether the query q of length smaller than y_{n-1}^{n-1} belongs to B or not deterministically takes time $2^{2^{y_{n-1}^{n-1}}}$, which is no more than $2^{y_n} = 2^{|b|}$ (Recall that in this case, $b = 0^{y_n}$).
- Determining whether x belongs to K can be done nondeterministically in $2^{|x|} < 2^{|b|}$.

We conclude that $B \in$ NEXP. ◀

▶ **Lemma 13.** *$K \leq_{k\text{-}tt}^p B$.*

Proof. In B's construction, for every x that is in K, we encode at least one of the following strings $\langle \alpha_1, x\rangle \ldots \langle \alpha_k, x\rangle$ into B. Strings that do not belong to K are not encoded into B. It follows that $K \leq_{k\text{-}tt}^p B$. ◀

It is not hard to see that B is not weakly $\leq_{k\text{-}tt}^p$-autoreducible, so by Lemma 12 and Lemma 13, B is a $\leq_{k\text{-}tt}^p$-complete set for NEXP that is not weakly $\leq_{k\text{-}tt}^p$-autoreducible. ◀

The above proof also yields the following corollary:

▶ **Corollary 14.** *For any positive integer k, there is a $\leq_{k\text{-}dtt}^p$-complete set for* NEXP(EXP) *that is not weakly $\leq_{k\text{-}tt}^p$-autoreducible.*

4 Implications

We begin with the following theorem.

▶ **Theorem 15.** *Every $\leq_{dtt}^p$-complete set for* EXP *is $\leq_{NOR\text{-}tt}^p$-autoreducible.*

Glaßer et al. [6] also showed that every $\leq_{dtt}^p$-complete set for EXP is $\leq_{dtt}^p$-autoreducible. Then by Theorem 15, Corollary 14 is somehow "tight" for EXP.

Algorithm 1 Algorithm to decide B. Input is of the form $\langle 0^i, x\rangle$.

```
Q := Q(M_i, ⟨0^i, x⟩) // Set of all queries of M_i on input ⟨0^i, x⟩
If (x ∉ Q) Then
   If (x ∉ A) Then
       Accept
   Else
       Reject
   EndIf
Else
   Reject
EndIf
```

Algorithm 2 Autoreduction algorithm for A. Input string is x.

```
Q := {q_1, ..., q_k} := Q(M_j, ⟨0^j, x⟩)
If x ∉ Q Then
   If ((q_1 ∉ A)&&(q_2 ∉ A)&& ... &&(q_k ∉ A)) Then
      Accept
   Else
      Reject
   EndIf
Else
   Reject
EndIf
```

Proof. Let A be a $\leq^p_{dtt}$-complete set for EXP. We will show that A is also $\leq^p_{NOR\text{-}tt}$-autoreducible.

Let $\{M_i\}_{i\geq 1}$ be a standard enumeration of all $\leq^p_{dtt}$-reductions such that M_i runs in time $p_i(n) = n^i$ on inputs of size n.

Consider a set B containing elements of the form $\langle 0^i, x\rangle$ that are decided by Algorithm 1. Obviously $B \in$ EXP.

Since A is the $\leq^p_{dtt}$-complete set for EXP, $B \leq^p_{dtt} A$ by some disjunctive truth-table reduction M_j. For any x, if x is one of queries of M_j on input $\langle 0^j, x\rangle$, then $\langle 0^j, x\rangle \notin B$. This fact implies that for all queries q, including x, $q \notin A$. Then $x \notin A$. If x is not one of the queries $q_1, \ldots, q_k$ of M_j on input $\langle 0^j, x\rangle$, then $x \in A \iff \langle 0^j, x\rangle \notin B \iff q_i \notin A$ for all i.

Based on that observation, we have the autoreduction algorithm for A described in Algorithm 2.

Observe that this is a $\leq^p_{NOR\text{-}tt}$-autoreduction. Thus A is $\leq^p_{NOR\text{-}tt}$-autoreducible. ◀

Recall that every $\leq^p_{k\text{-}dtt}$-complete set for NEXP is $\leq^p_{k\text{-}dtt}$-autoreducible [6]. Also every $\leq^p_{k\text{-}dtt}$-complete set for EXP is both $\leq^p_{k\text{-}dtt}$-autoreducible [6] and $\leq^p_{NOR\text{-}k\text{-}tt}$-autoreducible. We want to know whether the same holds for NEXP; that is, whether every $\leq^p_{k\text{-}dtt}$-complete set for NEXP is also $\leq^p_{NOR\text{-}k\text{-}tt}$-autoreducible. Settling this question would lead to important complexity class results.

▶ **Theorem 16.** *For any positive integer k, every $\leq^p_{k\text{-}dtt}$-complete set for* NEXP *is $\leq^p_{NOR\text{-}k\text{-}tt}$-autoreducible if and only if* NEXP = coNEXP.

Proof. Suppose every $\leq^p_{k\text{-}dtt}$-complete set for NEXP is $\leq^p_{NOR\text{-}k\text{-}tt}$-autoreducible. Notice that K, the canonical complete set of NEXP, is also $\leq^p_{k\text{-}dtt}$-complete. By the assumption, K is $\leq^p_{NOR\text{-}k\text{-}tt}$-autoreducible.

Algorithm 3 NOR-Autoreduction algorithm for A. Input string is x.

```
⟨q_1, ..., q_k⟩ ← f(x)
For i:= 1 to k do
   If (x = q_i) Then
      Reject and Terminate
   EndIf
EndFor
If ((q_1 ∉ A)&& ... &&(q_k ∉ A)) Then
   Accept
Else
   Reject
EndIf
```

Let f be the autoreduction of K. That is, for every x, $f(x) = \langle q_1, \ldots, q_k \rangle$, $x \neq q_i$ for all i, and $x \in K \iff q_1 \notin K \wedge \cdots \wedge q_k \notin K$. We have the following fact:

$$x \in \overline{K} \iff x \notin K \iff q_1 \in K \vee \cdots \vee q_k \in K.$$

So $\bar{K} \leq^p_{k\text{-}dtt} K$. Because $K \in$ NEXP, we have $\overline{K} \in$ NEXP. Therefore, NEXP = coNEXP.

To prove the other direction, suppose NEXP = coNEXP. Let A be any $\leq^p_{k\text{-}dtt}$-complete set for NEXP. We show that A is also $\leq^p_{NOR\text{-}k\text{-}tt}$-autoreducible. Note that $\overline{A} \in$ NEXP. Hence, $\overline{A} \leq^p_{k\text{-}dtt} A$ by some polynomial-time function f. In other words, for any x, $f(x) = \langle q_1, \ldots, q_k \rangle$ and $x \in \overline{A} \iff q_1 \in A \vee q_2 \in A \vee \cdots \vee q_k \in A$.

Rewriting this, we have $x \in A \iff q_1 \notin A \wedge q_2 \notin A \wedge \cdots \wedge q_k \notin A$. Observe that if there is some i, $i = 1, \ldots, k$ such that $q_i = x$ then $x \notin A$. Because otherwise, it contradicts to the preceding fact. Based on these observations, we have the $\leq^p_{NOR\text{-}k\text{-}tt}$-autoreduction for A described in Algorithm 3. Hence, A is $\leq^p_{NOR\text{-}k\text{-}tt}$-autoreducible. ◀

▶ **Corollary 17.** *For any positive integer k, if there is a $\leq^p_{k\text{-}dtt}$-complete set for* NEXP *that is not $\leq^p_{NOR\text{-}k\text{-}tt}$-autoreducible, then* NEXP $\neq$ EXP.

Proof. The proof follows directly from either Theorem 15 or Theorem 16. ◀

In the following section, we will show a partial result about NOR-autoreducibility for a $\leq^p_{dtt}$-complete set for NEXP in the relativized world.

5 Relativization

While the question whether every $\leq^p_{dtt}$-complete set for NEXP is $\leq^p_{NOR\text{-}tt}$-autoreducible is still open, we can prove that it does not hold in a relativized world.

▶ **Theorem 18.** *Relative to some oracle B, there is a $\leq^{p^B}_m$-complete set for* NEXP^B *that is not $\leq^{p^B}_{NOR\text{-}tt}$-autoreducible.*

Proof. Let $\{M^B_j\}_{j\geq 1}$ be an enumeration of polynomial-time $\leq^{p^B}_{NOR\text{-}tt}$-autoreductions. Notice that M^B_j can now access oracle B.

Let $\{\text{NEXP}^B_i\}_{i\geq 1}$ be an enumeration of all nondeterministic exponential time oracle Turing machines. For each $j \geq 1$, suppose that n^j bounds the running time of M^B_j and 2^{n^j} bounds the running time of NEXP^B_j. Let $K^B = \{\langle i, x, l \rangle \mid \text{NEXP}^B_i \text{ accepts input } x \text{ within } l \text{ steps}\}$, where l is encoded in a binary string, be a canonical complete set for NEXP^B.

We will construct sets A and B with the property $x \in K^B \iff \langle 0, x\rangle \in A$, which ensures that $K^B \leq_m^p A$, and then A is $\leq_m^p$-hard for NEXP^B. We also need A and B so that for any $n \geq 1$, the following property holds: $0^{y_n} \in A \iff M_n^{B,A}$ rejects input 0^{y_n} (the value of y_n will be chosen later in the proof). These properties guarantee that M_n^B is not an autoreduction of A. Then we can conclude that A is not autoreducible for NEXP^B.

We construct A and B together in stages. In each stage, we encode K^B into A and diagonalize against all $\leq_{NOR\text{-}tt}^{p^B}$-reductions using the string 0^{y_n} simultaneously to obtain those above two properties.

We define the sequence $\{y_n\}_{n\geq 0}$ such that $y_0 = 1$ and $y_{n+1} = y_n^n + 1$ for every $n \geq 0$.

Suppose at stage n that the set A has already been constructed up to length $y_n - 1$. At this stage, we will construct A for strings of length between y_n and $y_{n+1} - 1$. Now consider all queries q of M_n^B on input 0^{y_n} made to oracle A when $|q| \geq y_n$ and $q = \langle 0, x\rangle$ for some j. Let Q be the set of all such queries q.

Consider the following cases:

1. If there is a query q' such that $|q'| < y_n$ and $q' \in A$. Then put 0^{y_n} into A and $\langle 0^{2^{y_n}}, 0^{y_n}\rangle$ into B. Finally, for all strings $s = \langle 0, x\rangle$ and $y_n \leq |s| < y_{n+1}$, put s into A if and only if $x \in K^B$.
2. Otherwise, ignore all queries of length smaller than y_n. For every $q' = \langle 0, x\rangle \in Q$ such that $x \in K^B$, choose any accepting path of K^B on input x and denote $Q^{q'}$ to be the set of all queries made in that path. Consider the following cases:
   a. If no such q' exists, then for all strings $s = \langle 0, x\rangle$ and $y_n \leq |s| < y_{n+1}$, put s into A if and only if $x \in K^B$.
   b. Otherwise, let P be the union of $Q^{q'}$ for all such q'. Notice that there are no more than y_n^n such q', and for every q', $|Q^{q'}| \leq 2^{|x|} < 2^{y_n^n}$. Then, $|P| < y_n^n 2^{y_n^n} < 2^{2^{y_n}}$. Therefore, there exists a string t of length 2^{y_n} such that $t \notin P$. Put $\langle t, 0^{y_n}\rangle$ into B. Put 0^{y_n} into A. Finally, for all strings of $s = \langle 0, x\rangle$ and $y_n \leq |s| < y_{n+1}$, put s into A if and only if $x \in K^B$.

This concludes the construction of sets A and B.

Now we will briefly show that A belongs to NEXP^B. To determine membership of an input 0^y, we just need to guess one string t of length 2^y and ask one query $\langle t, 0^y\rangle$ to oracle B; accept if and only if the answer is YES. For other input of the form $\langle 0, x\rangle$, accept if and only if $x \in K^B$. So $A \in \mathrm{NEXP}^B$.

To see that A is not reduced to itself by any $\leq_{NOR\text{-}tt}^{p^B}$-autoreduction, we will show that for any M_n, $M_n^{A,B}$ accepts 0^{y_n} if and only if $0^{y_n} \notin A$. In case (1), because there is one query q' such that $q' \in A$, by the NOR-tt reduction, $M_n^{A,B}$ rejects 0^{y_n}. Notice that putting $\langle 0^{2^{y_n}}, 0^{y_n}\rangle$ into B does not affect the membership of q' in A. In case (2a), $M_n^{A,B}$ accepts 0^{y_n} and in this case 0^{y_n} is not put into A, and then it makes M_n^B not reduce A to itself. In case (2b), $M_n^{A,B}$ does not accept 0^{y_n} and notice that putting $\langle t, 0^{y_n}\rangle$ into B does not affect the memberships of all q' in K^B. And finally 0^{y_n} is added to A to make M_n^B not reduce A to itself.

It is easy to see that $K^B \leq_m^p A$ because we encode all strings $x \in K^B$ by $\langle 0, x\rangle$ into A and nothing else, except the strings of form 0^*. Hence, A is the many-one complete set for NEXP^B that is not $\leq_{NOR\text{-}tt}^{p^B}$-autoreducible. ◀

We note that Theorem 16 actually relativizes. So we have the following familiar corollary:

▶ **Corollary 19.** *There is a set B such that relative to the oracle B, $\mathrm{NEXP}^B \neq \mathrm{coNEXP}^B$.*

Buhrman et al. [2] showed that relative to some oracle, there is a $\leq^p_{2\text{-}T}$-complete set for EXP that is not Turing autoreducible. Their technique also works for NEXP. I.e., we have the following theorem:

▶ **Theorem 20.** *Relative to some oracle, there is a* $\leq^p_{2\text{-}T}$*-complete set for* NEXP *that is not Turing autoreducible.*

6 Open Questions

We know for any positive integers s and k such that $2^s - 1 > k$ that there is a $\leq^p_{s\text{-}T}$-complete set for NEXP that is not $\leq^p_{k\text{-}tt}$-autoreducible. We do not know what happens when $2^s - 1 \leq k$. It is not known whether every Turing-complete set is Turing-autoreducible. Referring to Theorem 9, the situation for $\leq^p_{k\text{-}tt}$-reductions for $k \geq 4$ is still open.

Acknowledgements. Our thanks to Nils Wisiol and Benedikt Budig for useful discussions and feedback on drafts of this paper. Special thanks to Leen Torenvliet for his extensive comments and corrections of an earlier version.

References

1 K. Ambos-Spies. P-mitotic sets. In E. Börger, G. Hasenjäger, and D. Roding, editors, *Logic and Machines, Lecture Notes in Computer Science 177*, pages 1–23. Springer-Verlag, 1984.
2 H. Buhrman, L. Fortnow, D. van Melkebeek, and L. Torenvliet. Using autoreducibility to separate complexity classes. *SIAM Journal on Computing*, 29(5):1497–1520, 2000.
3 H. Buhrman, S. Homer, and L. Torenvliet. Completeness for nondeterministic complexity classes. *Mathematical Systems Theory*, 24(3):179–200, 1991.
4 H. Buhrman and L. Torenvliet. On the structure of complete sets. In *IEEE Structure in Complexity Theory Conference, 1994., Proceedings of the Ninth Annual*, pages 118–133, 1994.
5 R. Downey, S. Homer, W. Gasarch, and M. Moses. On honest polynomial reductions, relativizations, and P=NP. In *IEEE Structure in Complexity Theory Conference*, pages 196–207. IEEE Computer Society, 1989.
6 C. Glaßer, D. Nguyen, C. Reitwießner, A. Selman, and M. Witek. Autoreducibility of complete sets for log-space and polynomial-time reductions. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP (1)*, volume 7965 of *Lecture Notes in Computer Science*, pages 473–484. Springer, 2013.
7 C. Glaßer, M. Ogihara, A. Pavan, A. Selman, and L. Zhang. Autoreducibility, mitoticity, and immunity. *J. Comput. Syst. Sci.*, 73(5):735–754, 2007.
8 S. Homer. Minimal degrees for polynomial reducibilities. *J. ACM*, 34(2):480–491, 1987.
9 R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.
10 J. Feigenbaum R. Beigel. On being incoherent without being very hard. *Computational Complexity*, 2:1–17, 1992.
11 B. Trahtenbrot. On autoreducibility. *Dokl. Akad. Nauk SSSR*, 192, 1970. Translation in Soviet Math. Dokl. 11: 814– 817, 1970.

Differentiability of polynomial time computable functions

André Nies

Department of Computer Science, University of Auckland, Auckland, New Zealand
andre@cs.auckland.ac.nz

Abstract

We show that a real z is polynomial time random if and only if each nondecreasing polynomial time computable function is differentiable at z. This establishes an analog in feasible analysis of a recent result of Brattka, Miller and Nies, who characterized computable randomness in terms of differentiability of nondecreasing computable functions.

Further, we show that a Martin-Löf random real z is a density-one point if and only if each interval-c.e. function is differentiable at z. (To say z is a density-one point means that every effectively closed class containing z has density one at z. The interval-c.e. functions are, essentially, the variation functions of computable functions.)

The proofs are related: they both make use of the analytical concept of porosity in novel ways, and both use a basic geometric fact on shifting dyadic intervals by 1/3.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Polynomial time randomness, feasible analysis, differentiability, porosity

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.602

1 Main results

Recent research in algorithmic randomness has focussed on its interactions with computable analysis. Theorems from analysis stating the well-behaviour of a function almost everywhere (in the sense of measure) form a rich source of such interactions: effective versions of such theorems usually correspond to algorithmic randomness notions that have been studied in other contexts. For instance, Brattka, Miller and Nies showed the following effective version of a classical theorem due to Lebesgue.

▶ **Theorem 1** ([5], Thm. 4.1)**.** *Let $z \in [0,1]$. Then z is computably random $\Leftrightarrow$ $f'(z)$ exists for each nondecreasing computable function $f\colon [0,1] \to \mathbb{R}$.*

Here, a real z is computably random if no computable betting strategy can make unbounded profit when betting on the bits of a binary expansion of z; a nondecreasing function f is computable if and only if f is continuous and $f(q)$ is a computable real uniformly in a rational q. A result of Demuth [8] set in constructive language can be interpreted as the first theorem of this kind: Martin-Löf randomness of a real z corresponds to the differentiability at z of all computable functions of bounded variation. Other results along these lines are in [16, 17, 12].

An algorithm is called feasible if it can be carried out with bounded resources, which often means a running time that is polynomial in the size of the input. In feasible randomness/ feasible analysis, the underlying algorithmic concepts are re-interpreted in terms of feasible algorithms. For instance, a real $z \in [0,1]$ is called polynomial time random if no polynomial time betting strategy can make unbounded profit on the initial segments of its binary

© André Nies;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 602–613
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

expansion. Despite its naturalness and potential applications, this concept is still poorly understood. First studied by Wang [18], its base-invariance was only recently shown [11]. Base-invariance means that to show the real z is non-random, we can equivalently bet on the symbols in a base-b expansion of z, for any $b > 2$. (The proof used lower derivatives, a concept from analysis.)

Our first main result, Theorem 4 below, is the full analog of Theorem 1 in the polynomial time setting. We use a particular case of the base invariance proved in [11], namely that polynomial time randomness is invariant under adding or subtracting 1/3.

Our second main result, Theorem 7 below, also starts from Theorem 1, but now relaxes the effectiveness hypothesis on the nondecreasing functions f considered. Instead of being computable, we only require that f is interval-c.e., which means that $f(0) = 0$ and the ternary relation "$q < f(y) - f(x)$", for $q, x, y \in \mathbb{Q} \cap [0, 1]$ and $x < y$, is computably enumerable. We show that the corresponding randomness notion obtained through Lebesgue's theorem is also one that had been previously studied: Martin-Löf randomness together with being a density-one point. Another classical result, the Lebesgue density theorem [13] asserts that for almost every point z in a measurable class $\mathcal{C} \subseteq [0, 1]$, the class is "thick" around z in that the relative measure of $\mathcal{C}$ converges to 1 as one "zooms in" on z. $\mathcal{C}$ is called effectively closed if its complement is an effective union of open intervals with rational endpoints. We say that a real z is a density-one point if the assertion of this theorem holds for every effectively closed class $\mathcal{C}$. In Theorem 4 we show that *a ML-random real z is a density-one point* $\Leftrightarrow$ *$f'(z)$ exists for each interval-c.e. function $f\colon [0, 1] \to \mathbb{R}$*. In fact we formulate Theorem 7 via a randomness condition that is known to be equivalent to being a Martin-Löf random density-one point (Andrews et al.; see [10]): every left-c.e. betting strategy (technically: a martingale as defined below) converges along the binary expansion of z.

The implication "$\Leftarrow$" is not hard to see: if a ML-random real z is not a density-one point as shown by an effectively closed class $\mathcal{P} \subseteq [0, 1]$, then the interval-c.e. function $f(x) = \lambda([0, x] - \mathcal{P})$ is not differentiable at z (where λ denotes Lebesgue measure). Below, we give an alternative argument using the martingale formulation.

Despite being at very different levels of effectiveness, our two main results can be proved by similar methods. They can be broadly described as "geometric", in the sense that measure is not needed, because it suffices to talk about interaction of classes with intervals. One main concept used is the following: a class $\mathcal{C}$ of reals is porous at a real z if $\mathcal{C}$ has 'holes' of fixed positive proportion in arbitrarily small intervals containing z (see Subsection 2.1). Both results rest on the fact that ill-behaviour of a function f at z (such as non-differentiability in a particular way) means that a class related to f is porous at z. This implies that z is not random in the appropriate sense. For instance, in the feasible case, porosity can be used directly to construct a polynomial time betting strategy that succeeds on z.

The other 'geometric' ingredient was observed for instance by Morayne and Solecki [14]: the endpoints of a basic dyadic interval of length 2^{-n}, and the shift by 1/3 of another basic dyadic interval of the same length, have to be apart by at least $2^{-n}/3$ (Subsection 3.2).

I thank Santiago Figueira and Alexander Galicki for helpful comments. I thanks Santiago and his parents for providing their apartment in Miramar, Argentina where most of this research was carried out.

1.1 Polynomial time randomness and differentiability

A Cauchy name is a sequence $(p_i)_{i\in\mathbb{N}}$ of rationals such that $|p_i - p_k| \leq 2^{-i}$ for each $k > i$. It is used to represent the real $\lim_i p_i$. For feasible analysis, one uses a compact set of Cauchy names. A *special* Cauchy name is given by an infinite sequence $b_0, b_1, \ldots$ from $\{-1, 0, 1\}^\omega$.

We let $p_i = \sum_{k=0}^{i} b_k 2^{-k}$. We call the b_k the *symbols* of the special Cauchy name. If we want to ensure that the represented real is in $[0,1]$, we ask that the sequence is 0^∞, or starts with $0^k 1 \ldots$ for some $k \in \mathbb{N}$, or starts with $10^k(-1)\ldots$, or is 10^∞. Since we have a 3-element input alphabet, a Turing machine (which has to rely on a fixed alphabet of symbols) can process the initial segments of such a sequence.

A martingale is a function $M\colon 2^{<\omega} \to \mathbb{R}_0^+$ such that $2M(\sigma) = M(\sigma 0) + M(\sigma 1)$ for each string σ. For a bit sequence $Z \in \{0,1\}^\omega$ we let $Z\restriction_n$ denote the initial segment of length n. We say that M *succeeds* on Z if $\limsup_n M(Z\restriction_n) = \infty$.

▶ **Definition 2.** A martingale M is called *polynomial time computable* if from a string σ and $i \in \mathbb{N}$ we can in time polynomial in $|\sigma| + i$ compute the first i symbols of a special Cauchy name for $M(\sigma)$.

We say that a bit sequence Z is *polynomial time random* if no polynomial time martingale succeeds on Z. Polynomial time randomness was first studied by Wang [18]. For a recent publication on it that also provides background see [11].

▶ **Definition 3** (see e.g. [19])**.** A function $g\colon [0,1] \to \mathbb{R}$ is called *polynomial time computable* if there is a polynomial time Turing machine turning every special Cauchy name for $x \in [0,1]$ into a special Cauchy name for $g(x)$.

In more detail, the first n symbols of $g(x)$ can be computed in time polynomial in n, thereby using polynomially many symbols of the oracle tape holding a special Cauchy name for x. Commonly occurring functions such as $e^x, \sin x$ are polynomial time computable, essentially because analysis gives us rapidly converging approximation sequences, such as $e^x = \sum_n x^n/n!$. We can extend the definition in an obvious way to functions $g\colon [0,1]^n \to \mathbb{R}$. The use of special Cauchy names ensures that basic functions such as addition and multiplication are polynomial time computable. Our first main result is:

▶ **Theorem 4.** *Let $z \in [0,1]$. Then z is polynomial time random $\Leftrightarrow$ $f'(z)$ exists for each nondecreasing polynomial time computable function $f\colon [0,1] \to \mathbb{R}$.*

We note that the implication "$\Rightarrow$" was independently announced by Miyabe and Kawamura (2013), who directly adapted the proof of [5, Thm. 4.1] to the polynomial time setting.

1.2 Left-c.e. martingales and differentiability of interval c.e. functions

A real z is called *left-computably-enumerable* (left-c.e. for short) if the set $\{q \in \mathbb{Q} : q < z\}$ is computably enumerable. A martingale $M\colon 2^{<\omega} \to \mathbb{R}_0^+$ is called left-c.e. if $M(\sigma)$ is a left-c.e. real uniformly in σ.

Consider a real $z \in [0,1] - \mathbb{Q}$. If a martingale M converges to a finite value at the binary expansion of z, we write $M(z)$ for this value.

▶ **Definition 5.** We say that z is a *convergence point for left-c.e. martingales* if $M(z)$ exists for each left-c.e. martingale M.

Recall that for a function $f\colon [0,1] \to \mathbb{R}$, the variation function at $x \in [0,1]$ is the supremum of the sums $\sum_i |f(t_{i+1}) - f(t_i)|$ for finer and finer partitions $0 = t_1 < \ldots < t_n = x$ of $[0,x]$. In order to identify the variation functions of computable functions, Freer, Kjos-Hanssen, Nies and Stephan [12] studied a class of non-decreasing functions which they called *interval-c.e.*

▶ **Definition 6.** A non-decreasing function $f\colon [0,1] \to \mathbb{R}$ is *interval-c.e.* if $f(0) = 0$, and $f(y) - f(x)$ is a left-c.e. real, uniformly in rationals $x < y$.

Note that the variation function of each computable function of bounded variation is continuous and interval-c.e. Freer et al. [12], together with Rute, showed that conversely, every continuous interval-c.e. function is the variation function of a computable function. For continuous functions, in Def. 6 we can drop the condition that x, y are rationals and instead require the seemingly stronger condition that $f(y) - f(x)$ is a left-c.e. real relative to Cauchy names for $x < y$ [12]. Our second main result is:

▶ **Theorem 7.** *$z \in [0,1]$ is a convergence point for left-c.e. martingales $\Leftrightarrow$ $f'(z)$ exists for each interval-c.e. function $f\colon [0,1] \to \mathbb{R}$.*

2 Preliminaries

2.1 Porosity and density

The proofs of our two main results use the notion of *porosity* which originates in the work of Denjoy. See for instance [6, Ex. 7:9.12], or [4, 5.8.124] (but note the typo in the definition there).

▶ **Definition 8.** We say that a set $\mathcal{C} \subseteq \mathbb{R}$ is *porous at z* via the constant $\varepsilon > 0$ if there exist arbitrarily small $\beta > 0$ such that $(z - \beta, z + \beta)$ contains an open interval of length $\varepsilon\beta$ that is disjoint from $\mathcal{C}$. We say that $\mathcal{C}$ is *porous at z* if it is porous at z via some $\varepsilon > 0$.

For the definitions below we follow [3]. Let λ denote Lebesgue measure. The *lower density* of a set $\mathcal{C} \subseteq \mathbb{R}$ at a point z is :

$$\varrho(\mathcal{C}|z) = \liminf_{z \in I \wedge |I| \to 0} \frac{\lambda(I \cap \mathcal{C})}{|I|},$$

where I ranges over intervals. The *lower dyadic density* $\varrho_2(\mathcal{C}|z)$ is the variant one obtains when one only considers basic dyadic intervals containing z. Clearly $\varrho_2(\mathcal{C}|z) \geq \varrho(\mathcal{C}|z)$. We say that z is a (full) *density-one point* if $\varrho(\mathcal{C}|z) = 1$ for every effectively closed class $\mathcal{C}$ containing z; z is a dyadic density-one point if $\varrho_2(\mathcal{C}|z) = 1$ for every effectively closed class $\mathcal{C}$ containing z.

▶ **Proposition 9** (Khan and Miller, see [10], Part 3)**.** *For a Martin-Löf random real z, being a dyadic density-one point implies being a full density-one point.*

The convergence points for left-c.e. martingales coincide with the Martin-Löf random density-one points. This was obtained by 2012 work of a group in Madison consisting of U. Andrews, M. Cai, D. Diamondstone, S. Lempp, and J. S. Miller. The implication "left-c.e. martingale convergence $\Rightarrow$ density one point" was already pointed out in [2]. The converse is harder to prove. See [10, Part 3] for a write-up due to Nies.

2.2 Slopes and martingales

First we need notation and a few definitions, mostly taken from [5] or [3]. For a function $f\colon [0,1] \to \mathbb{R}$, the *slope* at a pair a, b of distinct reals in its domain is

$$S_f(a,b) = \frac{f(a) - f(b)}{a - b}.$$

For a nontrivial interval A with endpoints a, b, we also write $S_f(A)$ instead of $S_f(a,b)$.

We let σ, τ range over (binary) strings. For such a string σ, by $[\sigma]$ we denote the closed basic dyadic interval $[0.\sigma, 0.\sigma + 2^{-|\sigma|}]$. The corresponding open basic dyadic interval is denoted (σ).

▶ **Fact 10.** *Let f be a non-decreasing polynomial time computable function. Then the function M_f given by $\sigma \to S_f([\sigma])$ is a martingale that is polynomial time computable.*

Proof. To compute the i-th symbol of a special Cauchy name for $M(\sigma)$, it suffices to compute the first $(|\sigma| + i + c)$ symbols of special Cauchy names for $f(0.\sigma)$ and $f(0.\sigma + 2^{-|\sigma|})$, where c is an appropriate constant. This can be done in time polynomial in $|\sigma| + i$. ◀

Derivatives. If z is in an open neighborhood of the domain of f, the *upper* and *lower derivatives* of f at z are

$$\overline{D}f(z) = \limsup_{h\to 0} S_f(z, z+h) \quad \text{and} \quad \underline{D}f(z) = \liminf_{h\to 0} S_f(z, z+h),$$

where as usual, h ranges over positive and negative values. The derivative $f'(z)$ exists if and only if these values are equal and finite.

We will also consider the upper and lower *pseudo*-derivatives defined by:

$$\widetilde{D}f(x) = \limsup_{h\to 0^+} \{S_f(a,b) \mid a \le x \le b \wedge 0 < b - a \le h\},$$
$$\underset{\sim}{D}f(x) = \liminf_{h\to 0^+} \{S_f(a,b) \mid a \le x \le b \wedge 0 < b - a \le h\},$$

where a, b range over rationals in $[0, 1]$. We only use them because in our arguments it is often convenient to consider (rational) intervals containing x, rather than intervals with x as an endpoint.

▶ Remark. Brattka et al. [5, after Fact 2.4] verified that $\underline{D}f(z) \le \underset{\sim}{D}f(z) \le \widetilde{D}f(z) \le \overline{D}f(z)$ for any real $z \in [0, 1]$; furthermore, in [5, Fact 7.2] they showed that for continuous functions with domain $[0, 1]$, the lower and upper pseudo-derivatives of f coincide with the usual lower and upper derivatives.

These two pseudo-derivatives also coincide with the usual ones if f is nondecreasing. To show $\overline{D}f(z) \le \widetilde{D}f(z)$, fix an arbitrarily small $\epsilon > 0$. Given $h \neq 0$, choose rationals $a \le z$, $z + h \le b$ such that $(b - a) \le (1 + \epsilon)|h|$. Then $S_f(z, z + h) \le (1 + \epsilon)S_f(a, b)$. To show $\underset{\sim}{D}f(z) \le \underline{D}f(z)$, choose $[a, b]$ inside the interval given by $z, z + h$ with $|h| \le (1 + \epsilon)(b - a)$ and verify that $S_f(a, b) \le (1 + \epsilon)S_f(z, z + h)$.

We will use the subscript 2 to indicate that all the limit operations are restricted to the case of basic dyadic intervals containing z. Thus,

$$\widetilde{D}_2 f(x) = \limsup_{|A|\to 0} \{S_f(A) \mid x \in A \wedge A \text{ is basic dyadic interval}\},$$
$$\underset{\sim}{D}_2 f(x) = \liminf_{|A|\to 0} \{S_f(A) \mid x \in A \wedge A \text{ is basic dyadic interval}\}.$$

3 Lemmas on comparing derivatives, and on shifting intervals

3.1 A pair of analytical lemmas

The proofs of our main results combine effectiveness considerations with a pair of purely analytical lemmas. We show that discrepancy of dyadic and full upper/lower derivatives at z implies that some closed set is porous at z. The proof extends the idea in the proof of Proposition 9 due to Khan and Miller.

We denote by $\sigma \preceq \tau$ that σ is an initial segment of τ; $\sigma \prec \tau$ denotes that σ is a proper initial segment of τ; $\sigma \prec Z$ that σ is an initial segment of the infinite bit sequence Z.

▶ **Lemma 11.** *Suppose $f\colon [0,1] \to \mathbb{R}$ is a nondecreasing function. Suppose for a real $z \in [0,1]$, with binary representation $z = 0.Z$, there is rational p such that*

$$\widetilde{D}_2 f(z) < p < \widetilde{D} f(z).$$

Let $\sigma^* \prec Z$ be any string such that $\forall \sigma\, [\sigma^* \preceq \sigma \prec Z \Rightarrow S_f([\sigma]) \leq p]$. Then the closed set

$$\mathcal{C} = [\sigma^*] - \bigcup\{(\sigma) \mid S_f([\sigma]) > p\}, \tag{1}$$

which contains z, is porous at z.

Proof. Suppose $k \in \mathbb{N}$ is such that $p(1 + 2^{-k+1}) < \widetilde{D} f(z)$. We show that there exists arbitrarily large n such that some basic dyadic interval $[a, \tilde{a}]$ of length 2^{-n-k} is disjoint from $\mathcal{C}$, and contained in $[z - 2^{-n+2}, z + 2^{-n+2}]$. In particular, we can choose 2^{-k-2} as a porosity constant.

By choice of k there is an interval $I \ni z$ of arbitrarily short positive length such that $p(1 + 2^{-k+1}) < S_f(I)$. Let n be such that $2^{-n+1} > |I| \geq 2^{-n}$. Let a_0 be greatest of the form $\ell 2^{-n-k}$, $\ell \in \mathbb{Z}$, such that $a_0 < \min I$. Let $a_v = a_0 + v2^{-n-k}$. Let r be least such that $a_r \geq \max I$. Since f is nondecreasing and $a_r - a_0 \leq |I| + 2^{-n-k+1} \leq (1 + 2^{-k+1})|I|$, we have

$$S_f(I) \leq S_f(a_0, a_r)(1 + 2^{-k+1}),$$

and therefore $S_f(a_0, a_r) > p$. Then, by the averaging property of slopes at consecutive intervals of equal length, there is a $u < r$ such that

$$S_f(a_u, a_{u+1}) > p.$$

Since $(a_u, a_{u+1}) = (\sigma)$ for some string σ, this gives the required 'hole' in $\mathcal{C}$ which is near $z \in I$ and large on the scale of I: in Definition 8 (porosity), let $\beta = 2^{-n+2}$ and note that we have $[a_u, a_{u+1}] \subseteq [z - 2^{-n+2}, z + 2^{-n+2}]$ because $z \in I$ and $|I| < 2^{-n+1}$. ◀

There is a dual lemma for lower derivatives. Note that it can *not* simply be obtained from the preceding lemma by taking $-f$, because the function in the dual lemma is still non*de*creasing. In fact, now the shortish dyadic intervals we choose in the proof are all *contained in* I. (So we can achieve a porosity constant of 2^{-k-1}.)

▶ **Lemma 12.** *Suppose $f\colon [0,1] \to \mathbb{R}$ is a nondecreasing function. Suppose for a real $z \in [0,1]$, with binary representation $z = 0.Z$, there a rational q such that*

$$\underline{D} f(z) < q < \underline{D}_2 f(z).$$

Let $\sigma^* \prec Z$ be any string such that $\forall \sigma\, [\sigma^* \preceq \sigma \prec Z \Rightarrow S_f([\sigma]) \geq q]$. Then the closed set

$$\mathcal{C} = [\sigma^*] - \bigcup\{(\sigma) \mid S_f([\sigma]) < q\},$$

which contains z, is porous at z.

Proof. The argument is similar to the preceding one. We will show that we can choose as a porosity constant 2^{-k-1} where $k \in \mathbb{N}$ is such that $\underline{D} f(z) < q(1 - 2^{-k+1})$. There is an interval $I \ni z$ of arbitrarily short positive length such that $S_f(I) < q(1 - 2^{-k+1})$. As before, let n be such that $2^{-n+1} > |I| \geq 2^{-n}$. Let a_0 be least of the form $\ell 2^{-n-k}$, $\ell \in \mathbb{Z}$, such that $a_0 \geq \min(I)$. Let $a_v = a_0 + v2^{-n-k}$. Let r be greatest such that $a_r \leq \max(I)$.

Since f is nondecreasing and $a_r - a_0 \geq |I| - 2^{-n-k+1} \geq (1 - 2^{-k+1})|I|$, we have

$S_f(I) \geq S_f(a_0, a_r)(1 - 2^{-k+1})$,

and therefore $S_f(a_0, a_r) < q$. Then there is $u < r$ such that

$$S_f(a_u, a_{u+1}) < q.$$

As before, this gives the required 'hole' in $\mathcal{C}$ near $z \in I$. ◀

3.2 Basic dyadic intervals shifted by $1/3$

We prove the hard directions "⇒" in our main results by contraposition. We need to transform a condition formulated in the setting of real analysis (that a function is not differentiable at a real z) into a condition in Cantor space (that a martingale succeeds on the binary expansion Z of the real). To do so, we use a basic 'geometric' fact for instance observed by Morayne and Solecki [14]. For $m \in \mathbb{N}$ let $\mathcal{D}_m$ be the collection of intervals of the form

$$[k2^{-m}, (k+1)2^{-m}]$$

where $k \in \mathbb{Z}$. Let $\widehat{\mathcal{D}}_m$ be the set of intervals $(1/3) + I$ where $I \in \mathcal{D}_m$.

▶ **Lemma 13.** *Let $m \geq 1$. If $I \in \mathcal{D}_m$ and $J \in \widehat{\mathcal{D}}_m$, then the distance between an endpoint of I and an endpoint of J is at least $1/(3 \cdot 2^m)$.*

To see this, assume that $|k2^{-m} - (p2^{-m} + 1/3)| < 1/(3 \cdot 2^m)$. This yields $|3k - 3p - 2^m|/(3 \cdot 2^m) < 1/(3 \cdot 2^m)$, and hence $3|2^m$, a contradiction.

In order to apply Lemma 13, we may need values of nondecreasing functions $f\colon [0,1] \to \mathbb{R}$ at endpoints of any such intervals, which may lie outside $[0,1]$. So we think of f as extended to $[-1,2]$ via $f(x) = f(0)$ for $-1 \leq x < 0$ and $f(y) = f(1)$ for $1 < y \leq 2$. The effectiveness properties we consider here, polynomial time computable or interval-c.e. (defined in Section 2), are preserved by this. For the interval-c.e. functions, this is clear because it suffices to determine values of the function at rationals. In the polynomial time case, to represent reals in $[-1,2]$ by special Cauchy names (see Subsection 1.1), we now also allow sequences in $\{-1,0,1\}^\omega$ starting with $0^k(-1)\ldots$ and $10^k1\ldots$. To compute a value of the extended function for such a sequence, we let the Turing machine internally replace an input of the form $0^k(-1)\ldots$ by 0^∞ (which yields as an overall output a Cauchy name for $f(0)$), and an input of the form $10^k1\ldots$ by 10^∞ (which yields $f(1)$).

4 Proof of Theorem 4

We prove Theorem 4: a real z is polynomial time random ⇔ $f'(z)$ exists for each nondecreasing polynomial time computable function $f\colon [0,1] \to \mathbb{R}$.

Proof. ⇐: Suppose z is not polynomial time random. Then some polynomial time martingale succeeds on the binary expansion Z of z. A martingale M has the savings property if $M(\tau) \geq M(\sigma) - 2$ for each strings $\sigma \prec \tau$. By [11, Lemma 6], there is a polynomial time martingale M with the savings property that succeeds on Z.

Let μ_M be the corresponding measure given by $\mu_M([\sigma]) = 2^{-|\sigma|}M(\sigma)$. Let $f = \mathsf{cdf}_M$ be the cumulative distribution function of μ_M given by $\mathsf{cdf}_M(x) = \mu_M[0,x)$. Then $\underline{D}_2 f(z) = \infty$, so $f'(z)$ does not exist.

To show f is polynomial time computable, observe that by [11, Lemma 13], for each dyadic rational p, $f(p)$ is a dyadic rational that can be computed from p in polynomial time. Since M has the savings property, by [11, Prop. 12], f satisfies an 'almost-Lipschitz condition': there is $\epsilon > 0$ such that for every $x, y \in [0,1]$, if $x \leq y \leq x+\epsilon$, then $f(y) - f(x) = O((y-x)\cdot \log(1/y-x)$. This implies that f is polynomial time computable: Suppose we are given a special Cauchy name $(p_i)_{i\in\mathbb{N}}$ for a real z. We know that $|z - p_{n+\log n}| = O(2^{-n-logn})$. So by the almost-Lipschitz condition, we have $|f(z) - f(p_{n+\log n})| = O(2^{-n})$. Thus, a Turing machine can determine in polynomial time from the first $n + \log n$ symbols of the special Cauchy name for z the first n symbols of a special Cauchy name for $f(z)$.

$\Rightarrow$: We may assume $z > 1/2$. By the hypothesis on f and Fact 10, the martingale $M(\sigma) = S_f([\sigma])$ is polynomial time computable. Recall that a Cauchy name is a sequence $(p_i)_{i\in\mathbb{N}}$, $p_i \in \mathbb{Q}$, such that $\forall k > i\, |p_i - p_k| \leq 2^{-i}$. We denote by $M(\sigma)_u$ the u-th term of this Cauchy name, so that $|M(\sigma) - M(\sigma)_u| \leq 2^{-u}$.

Let Z be the bit sequence such that $z = 0.Z$. Since z is polynomial time random, $\lim_n M(Z\upharpoonright_n)$ exists. This is a polynomial time version of the Doob martingale convergence theorem; see, for instance [9, Thm. 7.1.3]. Returning to the language of slopes, the convergence of M on Z means that $\underset{\sim}{D}_2 f(z) = \widetilde{D}_2 f(z) < \infty$. Suppose now that $f'(z)$ fails to exist. Then by the remark near the end of Subsection 2.2, we have $\underset{\sim}{D} f(z) < \underset{\sim}{D}_2 f(z)$ or $\widetilde{D}_2 f(z) < \widetilde{D} f(z)$ since f is nondecreasing. We will show that Z is not polynomial time random for a contradiction.

First suppose that $\widetilde{D}_2 f(z) < \widetilde{D} f(z)$. Choose rationals r, p such that $\widetilde{D}_2 f(z) < r < p < \widetilde{D} f(z)$. Choose $u \in \mathbb{N}$ so large that $\widetilde{D}_2 f(z) < r - 2^{-u}$ and $r + 2^{-u} < p$. As usual let $Z \in \{0,1\}^\omega$ be such that $z = 0.Z$. Let n^* be sufficiently large so that $S_f(A) \leq r - 2^{-u}$ for each basic dyadic interval A containing z and of length $\leq 2^{-n^*}$. Choose k with $p(1 + 2^{-k+1}) < \widetilde{D} f(z)$. Then Lemma 11 applies via the string $\sigma^* = Z\upharpoonright_{n^*}$ (and the same value of k as in its proof).

We define polynomial time rational-valued martingales L, L' such that L succeeds on Z, or L' succeeds on Y, where $0.Y$ is the binary expansion of $z - 1/3$. By the base invariance of polynomial time randomness [11, Thm. 14], if the second case applies, the expansion of z in base 3 is not polynomial time random, and hence neither is Z, its expansion in base 2. Thus, in either case, Z is not polynomial time random.

Defining L. It suffices to consider strings $\sigma \succeq \sigma^*$. Let $L(\sigma^*) = 1$. Suppose $\eta \succeq \sigma^*$ and $L(\eta)$ has been defined. Check whether there is a string α of length $k+4$ such that $M(\eta\alpha)_u > r$.

If so, decrease the capital to 0 on $\eta\alpha$ (we know that $\eta\alpha \not\prec Z$, so this won't make us lose along Z). In return, increase the capital by a factor of $2^{k+4}/(2^{k+4}-1)$ along all strings $\eta\widehat{\alpha}$ such that $|\widehat{\alpha}| = k+4$ and $\widehat{\alpha} \neq \alpha$. Continue the strategy with all strings $\eta\widehat{\alpha}$.

If no such α exists, don't bet, that is, let $L(\eta 0) = L(\eta 1) = L(\eta)$. Continue with the strings $\eta 0$ and $\eta 1$.

Defining L'. Let $\rho^* = Y\upharpoonright_{n^*+1}$. It suffices to consider strings $\rho \succeq \rho^*$.

Let $L'(\rho^*) = 1$. Suppose $\rho \succeq \rho^*$ and $L'(\rho)$ has been defined. Check if there is a string β of length $k+5$ such that $[\rho\beta] + 1/3 \subseteq [\tau]$ for a string τ of length $|\rho\beta| - 1$, and $M(\tau)_u > r$.

If so, decrease the capital to 0 on $\rho\beta$ (we know that $\rho\beta \not\prec Y$). Increase the capital by a factor of $2^{k+5}/(2^{k+5}-1)$ along all strings $\rho\widehat{\beta}$ such that $|\widehat{\beta}| = k+5$ and $\widehat{\beta} \neq \beta$. Continue the strategy with all strings $\rho\widehat{\beta}$.

If no such β exists, don't bet, that is, let $L'(\rho 0) = L'(\rho 1) = L'(\rho)$. Continue with the strings $\rho 0$ and $\rho 1$.

We check that the martingale L can be computed in polynomial time. The rational $\gamma = (2^{k+4}-1)/2^{k+4}$ is dyadic of length $k+4$. First assume that σ is not intermediate between η and $\eta\alpha$ as above, that is, we don't have $\eta \prec \sigma$ and $|\sigma| < |\eta| + k + 4$. We can

efficiently decide whether $L(\sigma) = 0$. If $L(\sigma) \neq 0$, for an appropriate $\ell \leq |\sigma|/k$ that we can compute from σ, we have $L(\sigma) = \gamma^{-\ell}$. We can compute γ^r using a polynomial in $|\sigma|$ number of operations. Hence, since division is computable in polynomial time, we can compute in time polynomial in $|\sigma| + i$ the i-th component of a special Cauchy name for $\gamma^{-\ell}$.

If we do have $\eta \prec \sigma$ and $|\sigma| < |\eta| + k + 4$, we simply compute $L(\eta\gamma)$ for all γ of length $k + 4$ with $\sigma \prec \eta\gamma$, and output the average of these values.

By a similar argument, the martingale L' can be computed in polynomial time. We now show that L succeeds on Z, or L' succeeds on Y. Let $\mathcal{C}$ be the class from (1) in Lemma 11. Consider $n \geq n^* + 4$ and a 'hole' $[a, \tilde{a}] \cap \mathcal{C} = \emptyset$ where $[a, \tilde{a}]$ is a basic dyadic interval of length 2^{-n-k}, and $[a, \tilde{a}] \subseteq [z - 2^{-n+2}, z + 2^{-n+2}]$.

▶ **Claim 14.** *One of the following is true.*
(i) *$z, a, \tilde{a}$ are all contained in a single interval A taken from $\mathcal{D}_{n-4}$.*
(ii) *$z, a, \tilde{a}$ are all contained in a single interval A' taken from $\widehat{\mathcal{D}}_{n-4}$.*

To see this note that $\{a, \tilde{a}, z\}$ is contained in an interval of length 2^{-n+2}. Apply Lemma 13 and that $2^{-n+4}/3 > 2^{-n+2}$.

In case (i) let $A = [\eta]$, so that $\eta \prec Z$ (recall that $z \notin \mathbb{Q}$ so z is not an endpoint of A). Let $[a, \tilde{a}] = \eta\alpha$ where $|\alpha| = k + 4$. We have $z \notin [a, \tilde{a}]$, and L increases its capital by a factor of $2^{k+4}/(2^{k+4} - 1)$ along all strings $\eta\hat{\alpha}$ as above.

Now suppose case (ii) applies. Let ρ be the string such that $A' = [\rho] + 1/3$. There is an interval $[b, \tilde{b}]$ in $\widehat{\mathcal{D}}_{n+k+1}$ with $[b, \tilde{b}] \subseteq [a, \tilde{a}]$. Since (ii) holds we have $[b, \tilde{b}] = [\rho\beta]$ for some string β of length $k + 5$. We have $z \notin [b, \tilde{b}]$ and L' increases its capital by a factor of $2^{k+5}/(2^{k+5} - 1)$ along all strings $\rho\widehat{\beta}$ as above.

Note that the capital of L along Z, and of L' along Y, never decreases, because there is no basic dyadic interval $[\tau]$ containing z with $|\tau| \geq n^*$ and $S_f(\tau)_u \geq r$. Suppose that L fails on Z. Then for all sufficiently small holes $[a, \tilde{a}]$ case (ii) applies, so for sufficiently long $\gamma \prec Y$ we can find ρ with $\gamma \preceq \rho \prec Z$ such that L' increases its capital by a fixed factor > 1 on the next $k + 5$ bits of Y. So L' succeeds on Y.

The case $\underline{D}f(z) < \underline{D}_2 f(z)$ is analogous, using Lemma 12 instead of Lemma 11. ◀

A bit sequence is called computably stochastic if no computable selection rule can lead to an asymptotic imbalance of 0s and 1s; see e.g. [15, 7.6.2] or [9] for the formal definition. Ambos-Spies et al. [1] also studied the polynomial time version of this notion. They showed that $X \in \{0, 1\}^\omega$ is computably [polynomial time] stochastic iff no computable [polynomial time] martingale that uses only finitely many, positive rational betting factors can win on X. The martingales L, L' constructed above are of this kind after a slight modification in order to avoid betting capital 0.

▶ **Corollary 15.** *Suppose that a binary expansion of a real z is polynomial time stochastic. Then for each nondecreasing polynomial time computable function $f\colon [0, 1] \to \mathbb{R}$, we have $\widetilde{D}_2 f(z) = \widetilde{D} f(z)$ and $\underline{D}_2 f(z) = \underline{D} f(z)$.*

5 Proof of Theorem 7

Theorem 7 states that a real z is a convergence point for left-c.e. martingales $\Leftrightarrow$ $f'(z)$ exists for each interval-c.e. function $f\colon [0, 1] \to \mathbb{R}$.

The implication "$\Leftarrow$" is the easier one as already noted above. For a proof in the language of martingales, suppose a left-c.e. martingale M diverges along the binary expansion of z.

Let μ_M be the measure on $[0,1]$ corresponding to M, and let $\mathsf{cdf}_M(x) = \mu_M[0,x)$. Then cdf_M is interval-c.e. and $\mathsf{cdf}'_M(z)$ fails to exist.

5.1 Porosity and upper derivatives

Recall that in Definition 8 we introduced the notion that a class of reals is porous at a real.

▶ **Definition 16** ([3]). We call a real $z \in [0,1]$ a *porosity point* if some effectively closed class to which z belongs is porous at z. Otherwise, z is a *non-porosity point*.

For instance, every density-one point in the sense of Subsection 2.1 is a non-porosity point. The converse fails: every Turing incomplete Martin-Löf random real is a non-porosity point by [3], but not necessarily a density-one point [7].

▶ **Proposition 17.** *Let $f\colon [0,1] \to \mathbb{R}$ be interval-c.e. Then $\widetilde{D}_2 f(z) = \widetilde{D} f(z)$ for each non-porosity point z.*

Proof. Assume $\widetilde{D}_2 f(z) < \widetilde{D} f(z)$. Since f is interval-c.e., the function $\sigma \to S_f([\sigma])$ is a left-c.e. martingale. In particular, the class $\mathcal{C}$ defined in (1) in Lemma 11 is effectively closed. This class is porous at z for a contradiction. ◀

▶ Remark. If f is interval *right*-c.e. (in the obvious sense), we can apply the dual Lemma 12 to conclude that $\underset{\sim}{D} f(z) = \underset{\sim}{D}_2 f(z)$ for each non-porosity point z. For instance, let f be the Lipschitz function given by $f(x) = \lambda([0,x] \cap \mathcal{P})$ for an effectively closed class $\mathcal{P}$. Then we may conclude that the (lower) dyadic density of $\mathcal{P}$ at a non-porosity point x coincides with the (lower) full density, a variation on Proposition 9.

5.2 From dyadic to full derivative

We proceed to the proof of the implication "⇒". We may assume $z > 1/2$. The real z is a dyadic density-one point, hence a (full) density-one point by Prop. 9. Then $z - 1/3$ is also a ML-random density-one point. So, using the work of the Madison group discussed at the end of Subsection 2.1, the real $z - 1/3$ is also a c.e. martingale convergence point. In particular, both z and $z - 1/3$ are non-porosity points.

By the hypothesis on z and since S_f is a left-c.e. martingale, we have $\underset{\sim}{D}_2 f(z) = \widetilde{D}_2 f(z)$. By Proposition 17, we have $\widetilde{D}_2 f(z) = \widetilde{D} f(z)$. To complete the proof of "⇒" in Theorem 7, it remains to be shown that

$$\underset{\sim}{D} f(z) = \underset{\sim}{D}_2 f(z). \tag{2}$$

Then, since f is nondecreasing, by the remark near the end of Subsection 2.2 $f'(z)$ exists.

The plan is to show for a contradiction that if $\underset{\sim}{D} f(z) < \underset{\sim}{D}_2 f(z)$, then one of z, $z - 1/3$ is a porosity point. Note that in Cantor space we can apply notions of porosity via the usual transfer to $[0,1]$ given by the binary expansion; further, if a class $\mathcal{G} \subseteq \{0,1\}^\omega$ is porous at $Y \in \{0,1\}^\omega$, then its image in $[0,1]$ is porous at $0.Y$. We will actually show one of z, $z - 1/3$ is a porosity point in the sense of Cantor space, via Π^0_1 classes $\mathcal{E}$ and $\widehat{\mathcal{E}}$ defined below.

As in Fact 10, let $M = M_f$ be the martingale given by $\sigma \to S_f([\sigma])$. Note that M converges on z by hypothesis (recall that we write $M(z)$ for the limit). Thus $\underset{\sim}{D}_2 f(z) = \widetilde{D}_2 f(z) = M(z)$.

Let $\widehat{f}(x) = f(x + 1/3)$, and let $\hat{M} = M_{\widehat{f}}$. We now show that $\hat{M}$ converges on $z - 1/3$, and that the limits coincide.

▶ **Claim 18.** $M(z) = \hat{M}(z - 1/3)$.

As remarked above, $z - 1/3$ is also a convergence point for c.e. martingales. So $\hat{M}$ converges on $z - 1/3$. If $M(z) < \hat{M}(z - 1/3)$ then $\widetilde{D}_2 f(z) < \widetilde{D} f(z)$. However, z is a non-porosity point, so this contradicts Proposition 17. If $\hat{M}(z - 1/3) < M(z)$ we argue similarly using that $z - 1/3$ is a non-porosity point. This establishes the claim.

Assume for a contradiction that (2) fails. We extend the method in the proof of Lemma 12, taking into account both dyadic intervals, and dyadic intervals shifted by 1/3. For this recall the notation in Subsection 3.2. Also recall that $\underset{\sim}{D}_2 f(z) = M(z)$.

We can choose rationals p, q such that

$$\underset{\sim}{D} f(z) < p < q < M(z) = \hat{M}(z - 1/3).$$

Let $k \in \mathbb{N}$ be such that $p < q(1 - 2^{-k+1})$. Let u, v be rationals such that

$$q < u < M(z) < v \text{ and } v - u \leq 2^{-k-3}(u - q).$$

Let $n^* \in \mathbb{N}$ be such that for each $n \geq n^*$ and any interval $A \in \mathcal{D}_n \cup \widehat{\mathcal{D}}_n$ containing z, we have $S_f(A) \geq u$. Let

$$\begin{aligned} \mathcal{E} &= \{X \in \{0,1\}^\omega \colon \forall n \geq n^* M(X\restriction_n) \leq v\} \\ \widehat{\mathcal{E}} &= \{W \in \{0,1\}^\omega \colon \forall n \geq n^* \hat{M}(W\restriction_n) \leq v\} \end{aligned}$$

Since f is interval-c.e., M and $\hat{M}$ are left-c.e. martingales, so these classes are effectively closed. Let Z be the bit sequence such that $z = 0.Z$. By the choice of n^* we have $Z \in \mathcal{E}$. Let Y be the bit sequence such that $0.Y = z - 1/3$. We have $Y \in \widehat{\mathcal{E}}$.

Consider an interval $I \ni z$ of positive length $\leq 2^{-n^*-3}$ such that $S_f(I) \leq p$. Let n be such that $2^{-n+1} > |I| \geq 2^{-n}$. Let a_0 [resp., b_0] be least of the form $w2^{-n-k}$ [resp., $w2^{-n-k} + 1/3$], where $w \in \mathbb{Z}$, such that a_0 [resp., b_0] $\geq \min(I)$. Let $a_i = a_0 + i2^{-n-k}$ and $b_j = b_0 + j2^{-n-k}$. Let r, s be greatest such that $a_r \leq \max(I)$ and $b_s \leq \max(I)$.

As before, since f is nondecreasing and $a_r - a_0 \geq |I| - 2^{-n-k+1} \geq (1 - 2^{-k+1})|I|$, we have $S_f(I) \geq S_f(a_0, a_r)(1 - 2^{-k+1})$, and therefore $S_f(a_0, a_r) < q$. Then there is an $i < r$ such that $S_f(a_i, a_{i+1}) < q$. Similarly, there is $j < s$ such that $S_f(b_j, b_{j+1}) < q$.

▶ **Claim 19.** *One of the following is true.*

(i) *z, a_i, a_{i+1} are all contained in a single interval taken from $\mathcal{D}_{n-3}$.*

(ii) *z, b_j, b_{j+1} are all contained in a single interval taken from $\widehat{\mathcal{D}}_{n-3}$.*

For suppose that (i) fails. Then there is an endpoint of an interval $A \in \mathcal{D}_{n-3}$ (that is, a number of the form $w2^{-n+3}$ with $w \in \mathbb{Z}$) between $\min(z, a_i)$ and $\max(z, a_{i+1})$. Note that $\min(z, a_i)$ and $\max(z, a_{i+1})$ are in I. By Fact 13 and since $|I| < 2^{-n+1}$, there can be no endpoint of an interval $\hat{A} \in \widehat{\mathcal{D}}_{n-3}$ in I. Then, since $b_j, b_{j+1} \in I$, (ii) holds. This establishes the claim.

Suppose I is an interval as above and $2^{-n+1} > |I| \geq 2^{-n}$, where $n \geq n^* + 3$. Let $\eta = Z\restriction_{n-3}$ and $\hat{\eta} = Y\restriction_{n-3}$.

If (i) holds for this I then there is a string α of length $k + 3$ (where $[\eta\alpha] = [a_i, a_{i+1}]$) such that $M(\eta\alpha) < q$. So by the choice of $q < u < v$ and since $M(\eta) \geq u$ there is β of length $k + 3$ such that $M(\eta\beta) > v$. (The decrease along $\eta\alpha$ of the martingale M must be balanced by an increase along some $\eta\beta$.) This yields a hole in $\mathcal{E}$, large and near Z on the scale of I, which is required for porosity of $\mathcal{E}$ at Z. Similarly, if (ii) holds for this I, then there is a string α of length $k + 3$ (where $[\hat{\eta}\alpha] = [b_j, b_{j+1}]$) such that $M(\hat{\eta}\alpha) < q$. So by the choice of

$q < u < v$ and since $\hat{M}(\hat{\eta}) \geq u$ there is a string β of length $k+3$ such that $\hat{M}(\hat{\eta}\beta) > v$. This yields a hole large and near Y on the scale of I required for porosity of $\widehat{\mathcal{E}}$ at Y.

Thus, if case (i) applies for arbitrarily short intervals I, then $\mathcal{E}$ is porous at Z, whence z is a porosity point. Otherwise (ii) applies for intervals below a certain length. Then $\widehat{\mathcal{E}}$ is porous at Y, whence $z - 1/3$ is a porosity point.

References

1 K. Ambos-Spies, E. Mayordomo, Y. Wang, and X. Zheng. Resource-bounded balanced genericity, stochasticity and weak randomness. In *STACS 96 (Grenoble, 1996)*, volume 1046 of *Lecture Notes in Comput. Sci.*, pages 63–74. Springer, Berlin, 1996.

2 L. Bienvenu, N. Greenberg, A. Kučera, A. Nies, and D. Turetsky. K-triviality, Oberwolfach randomness, and differentiability. Mathematisches Forschungsinstitut Oberwolfach, preprint, 40 pages, 2012.

3 L. Bienvenu, R. Hölzl, J. Miller, and A. Nies. Demuth, Denjoy, and Density. Submitted, available at `http://arxiv.org/abs/1308.6402`, 2013.

4 V. I. Bogachev. *Measure theory. Vol. I, II.* Springer-Verlag, Berlin, 2007.

5 V. Brattka, J. Miller, and A. Nies. Randomness and differentiability. Submitted, `http://arxiv.org/abs/1104.4465`.

6 A. Bruckner, J. Bruckner, and B. Thomson. *Real Analysis.* Prentice Hall (Pearson), 2007.

7 A. R. Day and J. S. Miller. Density, forcing and the covering problem. Submitted, `http://arxiv.org/abs/1304.2789`, 2013.

8 O. Demuth. The differentiability of constructive functions of weakly bounded variation on pseudo numbers. *Comment. Math. Univ. Carolin.*, 16(3):583–599, 1975. Russian.

9 R. Downey and D. Hirschfeldt. *Algorithmic randomness and complexity.* Springer-Verlag, Berlin, 2010. 855 pages.

10 A. Nies (editor). Logic Blog 2013. Available at `http://dl.dropbox.com/u/370127/Blog/Blog2013.pdf`, 2013.

11 S. Figueira and A. Nies. Feasible analysis, randomness, and base invariance. Theory of Computing Systems, published electronically Oct 2013, DOI 10.1007/s00224-013-9507-7.

12 C. Freer, B. Kjos-Hanssen, A. Nies, and F. Stephan. Effective aspects of Lipschitz functions. To appear in Computability.

13 H. Lebesgue. Sur les intégrales singulières. *Ann. Fac. Sci. Toulouse Sci. Math. Sci. Phys. (3)*, 1:25–117, 1909.

14 M. Morayne and S. Solecki. Martingale proof of the existence of Lebesgue points. *Real Anal. Exchange*, 15(1):401–406, 1989/90.

15 A. Nies. *Computability and randomness*, volume 51 of *Oxford Logic Guides.* Oxford University Press, Oxford, 2009.

16 N. Pathak. A computational aspect of the Lebesgue differentiation theorem. *J. Log. Anal.*, 1:Paper 9, 15, 2009.

17 N. Pathak, C. Rojas, and S. G. Simpson. Schnorr randomness and the Lebesgue differentiation theorem. *Proc. Amer. Math. Soc.*, 142(1):335–349, 2014.

18 Y. Wang. *Randomness and Complexity.* PhD dissertation, University of Heidelberg, 1996.

19 K. Weihrauch. *Computable Analysis.* Springer, Berlin, 2000.

2-Stack Sorting is polynomial *

Adeline Pierrot[1] and Dominique Rossin[2]

1 **Institute of Discrete Mathematics and Geometry, TU Wien, Wien, Austria**
2 **LIX UMR 7161, École Polytechnique and CNRS, Palaiseau, France**

Abstract

This article deals with deciding whether a permutation is sortable with two stacks in series. Whether this decision problem lies in P or is NP-complete is a longstanding open problem since the introduction of serial compositions of stacks by Knuth in *The Art of Computer Programming* [6] in 1973. We hereby prove that this decision problem lies in P by giving a polynomial algorithm to solve it. This algorithm uses the concept of pushall sorting, which was previously defined and studied by the authors in [8, 9].

1998 ACM Subject Classification G.2.1 Combinatorics: Permutations and Combinations, E.1 Data Structures: Lists, Stacks, and Queues

Keywords and phrases permutation, stack, sort, NP-complete

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.614

1 Introduction

Stack sorting has been studied first by Knuth in the sixties [5]. Characterizing the stack-sortable permutations is a historical problem, which led to define permutation *patterns*, an active research domain in combinatorics (see the book [4]). Stack-sorting was then generalized by Tarjan, who introduced sorting networks [10] allowing to sort more permutations, and many variations of this problem have been studied afterwards (see [3] for a summary).

Here we study the decision problem "Is a given permutation σ sortable by two stacks connected in series?". It is cited many times in the literature: in [3], Bóna gives a summary of advances on stack-sorting and mentions this problem as possibly NP-complete; more recently, it is also cited as possibly NP-complete in [1]. Surprisingly, both conjectures exist: in [2], the authors conjecture it is NP-complete, while Murphy in [7] conjectures it is polynomial.

In this article, we solve this problem that stayed open for several decades by giving a polynomial decision algorithm. Details of the proofs can be found in [8].

The difficulty of this problem, whose statement is however very simple, lies in the fact that both stacks are considered at once, which gives a great liberty on which operation to apply on the permutation at each step, and yields an exponential naive algorithm.

There are two key ideas in this article: a/, limit the number of sortings to consider by proving that if a permutation σ is sortable, then there is a sorting process of σ respecting some condition denoted $\mathcal{P}$. b/, encode a possibly exponential number of sortings by a sequence of graphs called sorting graphs, using pushall stack configurations introduced in [8, 9].

The article is organized as follows: Section 2 studies general properties of two-stack sorting thanks to stack words and stack configurations and limits the number of sortings to consider by introducing Property (P). Section 3 introduces the sorting graph $\mathcal{G}^{(i)}$ which encodes possible stack configurations at a given time t_i and gives an algorithm to compute

* With the support of ANR project ANR BLAN-0204_07 MAGNUM and SFB project SFB F50.



© Adeline Pierrot and Dominique Rossin;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 614–626



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



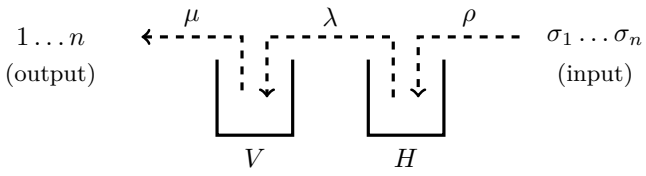


Figure 1 Sorting with two stacks in series.

this graph iteratively for all i from 1 to the number of right-to-left minima, leading to an algorithm deciding whether a permutation is 2-stack sortable. Then Section 4 proves that the resulting algorithm is polynomial.

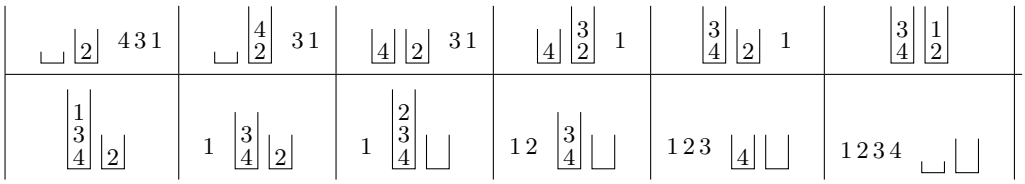
2 Study of two-stack sorting processes

2.1 Definitions and general problem statement

A permutation of size n is a word of n letters $\sigma = \sigma_1\sigma_2\dots\sigma_n$ on the alphabet $[1..n]$ containing each letter from 1 to n exactly once. Given two stacks H and V in series (see Figure 1) and a permutation σ, we want to sort the elements of σ using the stacks. We take σ as input: the elements σ_i are read one by one, from σ_1 to σ_n. We have three different operations (see Figure 1):

ρ: Take the next element of σ still in the input and push it on top of the first stack H.
λ: Pop the topmost element of stack H and push it on top of the second stack V.
μ: Pop the topmost element of stack V and write it to the output.

If there is a sequence $w = w_1 \dots w_k$ of operations ρ, λ, μ leading to the identity $1 \dots n$ as output, the permutation σ is said 2-stack sortable. In that case, we define the sorting word associated to this sorting process as the word w on the alphabet $\{\rho, \lambda, \mu\}$. Note that w must have n times each letter ρ, λ and μ and thus $k = 3n$. For example, 2431 is sortable using the following process:



This sorting process is encoded by the word $w = \rho\rho\lambda\rho\lambda\rho\lambda\mu\lambda\mu\mu\mu$. We can also decorate the word to specify the element on which each operation is performed. The *decorated word* for w and 2431 is $\hat{w} = \rho_2\rho_4\lambda_4\rho_3\lambda_3\rho_1\lambda_1\mu_1\lambda_2\mu_2\mu_3\mu_4$. Note that we have the same information in (σ, w) and in $\hat{w}$. Nevertheless, in a decorated word each letter ρ_i, λ_i or μ_i appears only once. The decorated word associated to (σ, w) is denoted $\hat{w}^\sigma$.

Not all permutations are 2-stack sortable (the smallest non-sortable ones are of size 7, e.g. $\sigma = 2435761$). The question of interest here is to decide whether a permutation is sortable.

There is a naive algorithm for this: given a permutation σ of size n, a sorting process corresponds to a word on the alphabet $\{\rho, \lambda, \mu\}$ of size $3n$. It is thus enough to test all words of size $3n$ and check if one of them yields the identity permutation on the output when taking σ as input. But this decision algorithm is exponential since there are 3^{3n} words to test.

The number of words to test can be reduced by noting that not all words correspond to a sorting process: a necessary condition is to contains n times each letter. But some permutations have an exponential number of sorting processes. For instance, it is easy to see that the decreasing permutation $n(n-1)\dots 1$ admits 2^{n-1} sorting processes.

A natural solution would be to define a canonical sorting process among all possible sorting processes of a permutation, but researches in this direction have been unsuccessful. Several greedy algorithms for 2-stack sorting have been defined, (cf. [11] and [2]) but none is able to sort all 2-stack sortable permutations. A key idea of our polynomial algorithm is to limit the number of sortings to consider by studying stack words and stack configurations.

2.2 Stack words and stack configurations

Not all words on the alphabet $\{\rho, \lambda, \mu\}$ describe sorting processes.

▶ **Definition 1** (stack word and sorting word)**.** Let w be a word on the alphabet $\{\rho, \lambda, \mu\}$ and $\alpha \in \{\rho, \lambda, \mu\}$. Then $|w|_\alpha$ denotes the number of occurrences of α in w.

A *stack word* is a word $w \in \{\rho, \lambda, \mu\}^*$ such that for any prefix v of w, $|v|_\rho \geq |v|_\lambda \geq |v|_\mu$.

A *sorting word* is a stack word w such that $|w|_\rho = |w|_\lambda = |w|_\mu$.

For any permutation σ, a sorting word *for* σ is a sorting word encoding a sorting process with σ as input (leading to the identity of size $|\sigma|$ as output).

Intuitively, stack words describe a sequence of operations ρ, λ, μ that can be carried out starting with empty stacks (and arbitrarily long input), whereas sorting words encode a complete sorting process (stacks are empty at the beginning and at the end of the process).

Another way of describing sorting processes is, instead of focusing on the operations made, to focus on the description of which element lies in each stack (and their order in the stacks) at each step of the process. Such a description for one step is called a *stack configuration*. For example, the figure on the right is a stack configuration which is a part of the sorting process $\rho\rho\lambda\rho\lambda\rho\lambda\mu\lambda\mu\mu\mu$ of 2431.



Stack configurations and stack words describing a sorting process are linked:

▶ **Definition 2.** Let w be a stack word. Starting with a permutation σ as input, the stack configuration reached after performing operations described by the word w is denoted $c_\sigma(w)$. A stack configuration c is *reachable* for σ if there exists a stack word w such that $c = c_\sigma(w)$. A stack configuration is *poppable* if the elements in stacks H and V can be output in increasing order using operations λ and μ.

Any stack configuration which is a part of a sorting process of a permutation σ has to be reachable for σ and poppable. We describe necessary or sufficient conditions for a stack configuration to be reachable or poppable.

▶ **Lemma 3.** *Let c be a stack configuration. If c is poppable, then the values of the elements of V are in decreasing order from bottom to top. If c is reachable for a permutation σ, then the elements of H have increasing indices (as letters of σ) from bottom to top.*

Poppable stack configurations have been characterized in [9] by the following Lemma. Recall first that a permutation $\pi = \pi_1\pi_2\dots\pi_k$ is a *pattern* of $\sigma = \sigma_1\sigma_2\dots\sigma_n$ if there exists indices $1 \leq i_1 < i_2 < \dots < i_k$ such that $\sigma_{i_1}\sigma_{i_2}\sigma_{i_3}\dots\sigma_{i_k}$ is order-isomorphic to π.

▶ **Lemma 4.** *A stack configuration c is poppable if and only if:*

- *Stack V does not contain the pattern* 12 *(seen from bottom to top).*

- *Stack H does not contain the pattern* 132 *(seen from bottom to top).*
- *Stacks (V, H) do not contain the pattern* $|2|13|$.

Plus, there is a unique sequence of stack operations to pop the elements out in increasing order.

The first two conditions are usual pattern relations (note that the first one corresponds to the first part of Lemma 3). The third one means that there are no elements i, j, k with i in V and j, k in H (k above j) such that $j < i < k$.

A stack configuration is usually associated to a permutation, implying that the elements in the stacks are a subset of those of the permutation. In particular a *total stack configuration of σ* is a stack configuration in which the elements of the stacks are exactly all those of σ.

▶ **Definition 5** (pushall configuration)**.** A stack configuration is a *pushall* stack configuration of σ if it is poppable, total and reachable for σ.

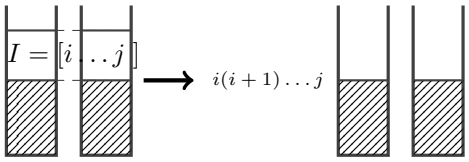
Pushall stack configurations, which were defined and studied in [8] and [9], play a key role in our polynomial algorithm. Indeed, a permutation which ends with its smallest element is 2-stack sortable if and only if it admits a pushall stack configuration. Moreover we have:

▶ **Theorem 6** ([8, 9])**.** *One can compute in time $O(n^2)$ the set of pushall stack configurations of any permutation of size n.*

2.3 Restrict the number of sortings to focus on: Property (P)

Some permutations have an exponential number of sorting processes. To obtain a polynomial algorithm, we restrict the number of sortings to focus on. The following lemma shows that we can focus on sorting processes where smallest elements are popped out "as soon as possible".

▶ **Lemma 7.** *Let σ be a 2-stack sortable permutation and $w = uv$ be a sorting word for σ. Assume that after performing the operations of u, the elements $1 \dots i-1$ have been output and the elements $i \dots j$ are at the top of the stacks. Then there exists a sorting word $w' = uu'u''$ for σ such that u' consists only of moving the elements $i \dots j$ from the stacks to the output in increasing order without moving any other elements.*



Now we add some other constraints on the sortings, using the block-decomposition of permutations. A block B of a permutation $\sigma = \sigma_1\sigma_2 \dots \sigma_n$ is a factor $\sigma_i\sigma_{i+1} \dots \sigma_j$ of σ such that the set of values $\{\sigma_i, \dots, \sigma_j\}$ is an interval. Given two blocks B and B' of σ, we say that $B < B'$ if and only if $\sigma_i < \sigma_j$ for all $\sigma_i \in B$, $\sigma_j \in B'$. A permutation σ is $\ominus$-decomposable if it can be written as $\sigma = B_1 \dots B_k$ such that $k \geq 2$ and for all i, $B_i > B_{i+1}$ in terms of blocks. Otherwise we say that σ is $\ominus$-indecomposable. When each B_i is $\ominus$-indecomposable, we write $\sigma = \ominus[B_1, \dots, B_k]$ and call it the $\ominus$-decomposition of σ. Note that we do not renormalize the elements of B_i, thus, except B_k, the B_i are not permutations. Nevertheless, B_i can be seen as a permutation by subtracting $|B_{i+1}| + \dots + |B_k|$ to all its elements.

The RTL (right-to-left) minima of a permutation are the elements σ_k such that there is no j with $j > k$ and $\sigma_j < \sigma_k$. We denote by σ_{k_i} the i^{th} RTL minimum of σ. If σ has r RTL minima, then $\sigma = \dots \sigma_{k_1} \dots \sigma_{k_2} \dots \sigma_{k_r}$ with $\sigma_{k_1} = 1$ and $k_r = n$.

Take for example the permutation $\sigma = 6\,5\,8\,7\,4\,1\,3\,2$. The $\ominus$-decomposition of σ is $\sigma = \ominus[6\,5\,8\,7, 4, 1\,3\,2]$. Furthermore, σ has 2 RTL-minima which are $\sigma_6 = 1$ and $\sigma_8 = 2$.

We denote $\sigma^{(i)} = \{\sigma_j \mid j < k_i \text{ and } \sigma_j > \sigma_{k_i}\}$ the restriction of σ to elements in the upper left quadrant of the i^{th} RTL minimum σ_{k_i}. The $\ominus_i$-decomposition of σ is the $\ominus$-decomposition

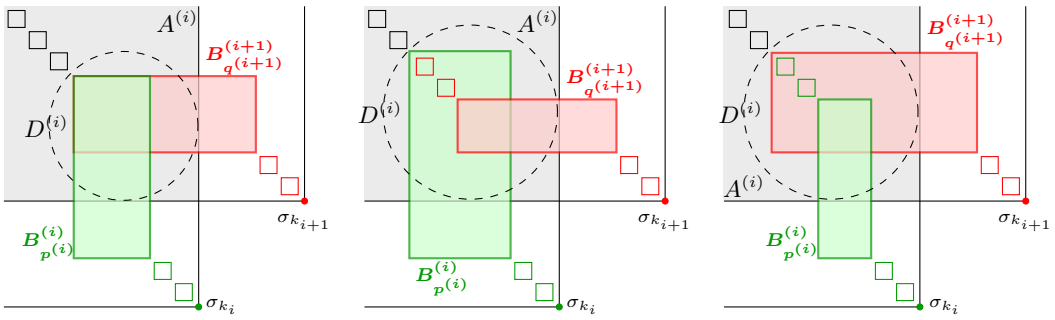


Figure 2 The $\ominus$-decomposition of $\sigma^{(i)}$ and of $\sigma^{(i+1)}$ visualized in the diagram of σ (set of the points at coordinates (i, σ_i)) resp. when $p^{(i)} = q^{(i+1)}$, $p^{(i)} < q^{(i+1)}$ and $p^{(i)} > q^{(i+1)}$).

of $\sigma^{(i)} = \ominus[B_1^{(i)}, \ldots, B_{s_i}^{(i)}]$. In the following, s_i always denotes the number of blocks of $\sigma^{(i)}$ and $B_j^{(i)}$ the j^{th} block in the $\ominus_i$-decomposition.

We denote by $A^{(i)}$ the common part of $\sigma^{(i)}$ and $\sigma^{(i+1)}$, i.e., $A^{(i)} = \sigma^{(i)} \bigcap \sigma^{(i+1)} = \{\sigma_j \mid j < k_i \text{ and } \sigma_j > \sigma_{k_{i+1}}\}$. This sub-permutation $A^{(i)}$ intersects $\ominus$-indecomposable blocks of $\sigma^{(i)}$ and $\sigma^{(i+1)}$. Let $p^{(i)}$ (resp. $q^{(i+1)}$) be the index such that $B^{(i)}_{p^{(i)}}$ (resp. $B^{(i+1)}_{q^{(i+1)}}$) contains the smallest value of $A^{(i)}$. Let $D^{(i)} = \left(B^{(i)}_{p^{(i)}} \bigcup B^{(i+1)}_{q^{(i+1)}}\right) \bigcap A^{(i)}$ (see Figure 2).

▶ **Definition 8** (Properties (P_i) and (P))**.** Let w be a sorting word for a permutation σ. We say that w verifies (P_i) if and only if the corresponding decorated word $\hat{w}$ satisfies:

(i) μ_{σ_j} appears before $\rho_{\sigma_{k_i}}$ for all $\sigma_j < \sigma_{k_i}$,

(ii) $\rho_{\sigma_{k_i}} \lambda_{\sigma_{k_i}} \mu_{\sigma_{k_i}}$ is a factor of $\hat{w}$,

(iii) All operations μ_{σ_ℓ} with $\sigma_\ell \in B_j^{(i)}$ and $j \in [p^{(i)} + 1..s_i]$ appear before $\rho_{\sigma_{(k_i)+1}}$ in $\hat{w}$.

If a word w verifies Property (P_i) for all i then we say that w verifies Property (P). We call t_i the time just before σ_{k_i} enters stack H.

▶ **Theorem 9.** *If σ is 2-stack sortable then there is a sorting word of σ satisfying Property (P). In particular, in the sorting process encoded by this word, the elements in the stacks at time t_i are exactly those of $\sigma^{(i)}$.*

Theorem 9 is proved recursively using the following lemmas:

▶ **Lemma 10** (easy)**.** *If the sorting word encoding a sorting process of σ verifies Property (P_i), then the elements in the stacks at time t_i are exactly those of $\sigma^{(i)}$.*

▶ **Lemma 11** (from Lemma 3)**.** *If $\sigma = \ominus[B_1, \ldots B_k]$ then in any poppable stack configuration reachable for σ, for all $i < j$, the elements of B_i are, in the stacks, below the elements of B_j.*

▶ **Lemma 12** (from Lemma 7 and Lemma 11)**.** *Let w be a sorting word for a permutation σ, r be the number of RTL-minima of σ and $\ell \in [1..r]$. If w verifies (P_i) for $i \in [1..\ell{-}1]$ then there exists a sorting word w' for σ that verifies (P_i) for $i \in [1..\ell]$.*

Theorem 9 ensures that if a permutation σ is sortable then there is a sorting in which at each time step t_i, the elements in the stacks are those of $\sigma^{(i)}$. Thus if σ is sortable, then for all i, $\sigma^{(i)}$ admits a pushall stack configuration. This necessary condition is not sufficient: the pushall stack configuration for $\sigma^{(i)}$ has to be *accessible* from the one of $\sigma^{(i-1)}$.

2.4 Stack configurations and accessibility

The stack configurations for a sorting process encode the elements that are currently in the stacks. But some elements are still waiting in the input and some elements have been output. To fully characterize a configuration, we define an *extended* stack configuration of a permutation σ of size n to be a pair (c, i) where $i \in \{1, \dots, n+1\}$ and c is a poppable stack configuration made of all elements within $\sigma_1, \sigma_2, \dots, \sigma_{i-1}$ that are greater than a value p. The elements $\sigma_i, \dots, \sigma_n$ are still in the input and the elements $\sigma_j < p, j < i$ have already been output. Note that we don't ask the configuration to be reachable.

▶ **Definition 13.** Let (c, i) be an extended stack configuration of a permutation σ. Then an extended stack configuration (c', j) of σ is *accessible* from (c, i) if the stack configuration (c', j) can be reached starting from (c, i) and performing operations ρ, λ and μ s. t. the elements of $c \cup \{\sigma_i \dots \sigma_n\}$ that are output by the operations μ performed are output in increasing order.

For example, for $\sigma = 2\,3\,1\,6\,5\,8\,4\,7$, the sequence of operations $\mu_2\mu_3\rho_6\rho_5\rho_8\lambda_8$ proves that $(\boxed{8}\,\boxed{\begin{smallmatrix}5\\6\end{smallmatrix}}, 7)$ is accessible from $(\boxed{\begin{smallmatrix}2\\3\end{smallmatrix}}\,\boxed{}, 4)$. But $(\boxed{\begin{smallmatrix}2\\3\end{smallmatrix}}\,\boxed{6}, 5)$ is not accessible from $(\boxed{\begin{smallmatrix}1\\2\end{smallmatrix}}\,\boxed{3}, 4)$.

In the following, given two total pushall stack configurations c and c' corresponding to $\sigma^{(i)}$ and $\sigma^{(i+1)}$, we study conditions for c' to be accessible from c,

▶ **Lemma 14.** *Let (c, k_i), resp. (c', k_{i+1}), be a pushall stack configuration of $\sigma^{(i)}$, resp. $\sigma^{(i+1)}$. Let $\pi = \sigma_{|B^{(i)}_{p^{(i)}} \bigcup B^{(i+1)}_{q^{(i+1)}}}$. Then (c', k_{i+1}) is accessible from (c, k_i) for σ if and only if:*

1. *$(c'_{|\pi}, |\pi| + 1)$ is accessible from $(c_{|\pi}, \#(D^{(i)} \bigcup B^{(i)}_{p^{(i)}}) + 1)$ for π (see Figure 2).*
2. *$\forall j < \min(p^{(i)}, q^{(i+1)})$, $c_{|B^{(i)}_j} = c'_{|B^{(i)}_j}$.*
3. *$\forall j > q^{(i+1)}, c'_{|B^{(i+1)}_j}$ is a pushall configuration of $\sigma_{|B^{(i+1)}_j}$.*

Informally, it is possible to efficiently decide whether a configuration at time t_i can evolve into a given configuration at time t_{i+1}. Moreover, during this transition, only a few operations are undetermined: the largest elements don't move, the smallest ones are output in increasing order, and the remaining ones form a $\ominus$-indecomposable permutation. This will allow us to exhibit a polynomial algorithm checking accessibility. The proof of Lemma 14 relies on Lemma 7, Lemma 11 and the following lemma:

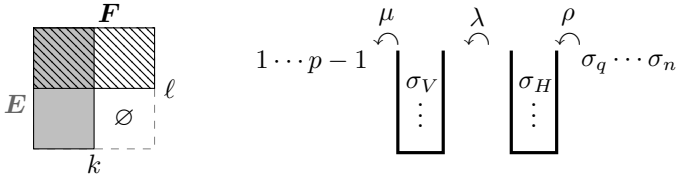
▶ **Lemma 15.** *Let $\sigma_\ell \in A^{(i)}$. During a sorting process of σ, the elements σ_m s. t. $\sigma_m > \sigma_\ell$ and $m < \ell$ do not move between t_i and t_{i+1} (indeed σ_ℓ prevent those elements from moving).*

Thanks to Lemma 14, if c and c' are two total pushall stack configurations corresponding to stack configurations of $\sigma^{(i)}$ and $\sigma^{(i+1)}$, to decide whether c' is accessible from c it is enough to check three conditions. The last two ones are easy to check, and the first one can be checked using the following lemma:

▶ **Lemma 16.** *Let σ be a permutation of size n and (c, i), (c', j) two extended stack configurations of σ with $i < j$. Let E (resp. F) be the set of elements of c (resp. c').*

- *If there exists $k, \ell \in \{1 \dots n\}$ such that $E = \{\sigma_m \mid m \leq k\}$ and $F = \{\sigma_m \mid \sigma_m \geq \ell\}$,*
- *if moreover $E \cup F = \sigma$,*

then we can decide in linear time whether (c', j) is accessible from (c, i) using Algorithm 1.



Algorithm 1: isAccessible$\big((c,i),(c',j),\sigma\big)$

Data: σ a permutation and $(c,i),(c',j)$ two stack configurations of σ satisfying conditions of Lemma 16
Result: `true` or `false` depending on whether the configuration c' is accessible from c
Put configuration c in the stacks H and V
$p \leftarrow$ the smallest element of $c \cup \{\sigma_i \dots \sigma_n\}$ (next element to be output)
$q \leftarrow i$ (next index of σ that must enter the stacks)
We denote by $V(c')$ the set of elements of V in configuration c' and by σ_V the top of V in the current configuration (the same goes for H).
while $q < j$ *or* $p < \ell$ *or* $\sigma_H \in V(c')$ **do**
  if $\sigma_V = p$ **then** Perform μ; $p \leftarrow p+1$
  else if $\sigma_H < \ell$ **then** Perform λ
    else if $H = \varnothing$ *or* $\sigma_H \in H(c')$ **then** Perform ρ; $q \leftarrow q+1$
      else if $\sigma_q \in H(c')$ *or* $\sigma_H > \sigma_q$ **then** Perform λ
        else Perform ρ; $q \leftarrow q+1$
Return $(H,V) == c'$

The proof of Lemma 16 relies on Lemmas 4 and 7. The idea is that Algorithm 1 performs only operations that we have to do to obtain (c',j) starting from (c,i). Thus (c',j) is accessible from (c,i) if and only if the configuration obtained at the end is c'.

3 An iterative algorithm

3.1 A first naive algorithm

From Theorem 9, a permutation σ is 2-stack sortable if and only if it admits a sorting process satisfying Property (P). The main idea is to compute the set of sorting processes of σ satisfying (P) and decide whether σ is 2-stack sortable by testing the emptiness of this set.

Verifying (P) means verifying (P_j) for all j from 1 to r, r being the number of right-to-left minima (whose indices are denoted k_j). The algorithm proceeds in r steps: for all i from 1 to r we iteratively compute the sorting processes of $\sigma_{\leq k_i}$ verifying (P_ℓ) for all ℓ from 1 to i (with $\sigma_{\leq k_i} = \sigma_1 \dots \sigma_{k_i}$). As $\sigma_{\leq k_r} = \sigma$, the last step gives sorting processes of σ satisfying (P).

By "compute the sorting processes of $\sigma_{\leq k_i}$" we mean "compute the stack configuration just before σ_{k_i} enters the stacks in such a sorting process":

▶ **Definition 17.** We call P_i-stack configuration of σ a stack configuration $c_\sigma(w)$ for which there exists u such that the first letter of u is $\rho_{\sigma_{k_i}}$ and wu is a sorting word of $\sigma_{\leq k_i}$ verifying (P) for $\sigma_{\leq k_i}$ (that is, verifying (P_ℓ) for all ℓ from 1 to i).

The algorithm is based on the following two lemmas:

▶ **Lemma 18** (Consequence of Theorem 9)**.** *For any i from 1 to r, $\sigma_{\leq k_i}$ is 2-stack sortable if and only if the set of P_i-stack configurations of σ is nonempty. In particular, σ is 2-stack sortable if and only if the set of P_r-stack configurations of σ is nonempty.*

▶ **Lemma 19** (Consequence of Lemma 10)**.** *Any P_i-stack configuration of σ is a pushall stack configuration of $\sigma^{(i)}$, accessible from some P_{i-1}-stack configurations of σ.*

The algorithm proceeds in r steps such that after step i we know every P_i-stack configuration of σ and we want to compute at step $i+1$ the P_{i+1}-stack configurations of σ. As P_{i+1}-stack configurations are pushall stack configurations of $\sigma^{(i+1)}$, a possible algorithm is to

take every pair of configurations (c, c') with c being a P_i-stack configuration of σ (computed at step i) and c' be any pushall stack configuration of $\sigma^{(i+1)}$ (given by Algorithm 5 of [9], see Theorem 6). Then we can use Algorithm 1 to decide whether c' is accessible from c for σ. This leads to an algorithm deciding whether a permutation σ is 2-stack sortable, but this algorithm is not polynomial. Indeed, the number of P_i-stack configurations of σ is possibly exponential. However, this set can be described by a polynomial representation as a graph.

3.2 Towards the sorting graph

We now explain how to adapt the previous idea to obtain a polynomial algorithm. Instead of computing all P_i-stack configurations of σ (which are pushall stack configurations of $\sigma^{(i)}$), we compute the restriction of such configurations to blocks $B_j^{(i)}$ of the $\ominus$-decomposition of $\sigma^{(i)}$. By Lemma 11, those configurations are stacked one upon the others to give a P_i-stack configuration. The stack configurations of any block $B_j^{(i)}$ are labeled with an integer which is assigned when the configuration is computed. Those pairs (configurations, integer) will be the vertices of the graph $\mathcal{G}^{(i)}$ which we call a *sorting graph*, the edges of which representing the configurations that can be stacked one upon the other. Vertices of the graph $\mathcal{G}^{(i)}$ are partitioned into levels corresponding to blocks $B_j^{(i)}$. The integer labels allows us to ensure the polynomiality of the representation. Indeed, a given label can only appear once per level of the graph $\mathcal{G}^{(i)}$. As those labels are assigned to configurations when they are created, each label corresponding to a pushall stack configuration, from Theorem 4.4 of [9] there are at most $9|\sigma|$ distinct labels thus at most $9|\sigma|$ vertices per level of the graph $\mathcal{G}^{(i)}$. This is formalized in Lemma 22. The label can be seen as the memory of the configuration that encodes its history since it has been created: two configurations having the same label come from the same initial pushall configuration.

More precisely, the sorting graph $\mathcal{G}^{(i)}$ for a permutation σ and an index i verifies:

- The vertices of $\mathcal{G}^{(i)}$ are partitioned into s_i subsets $V_j^{(i)}$ with $j \in [1 \dots s_i]$ called levels.
- For any $j \in [1 \dots s_i]$, the number of vertices in level $V_j^{(i)}$ is less than $9|\sigma|$.
- Each vertex $v \in \mathcal{G}^{(i)}$ is a pair (c, ℓ) with c a stack configuration and ℓ an index called *configuration index*.
- All configuration indices are distinct inside a graph level $V_j^{(i)}$.
- $(c, \ell) \in V_j^{(i)} \Rightarrow c$ is a pushall stack configuration of $B_j^{(i)}$ accessible for σ.
- There are edges only between vertices of adjacent levels $V_j^{(i)}$, $V_{j+1}^{(i)}$ (this implies Lemma 23).
- The paths between vertices of $V_1^{(i)}$ and $V_{s_i}^{(i)}$ correspond to the stack configurations of $\sigma^{(i)}$. Precisely, and that is why the algorithm is correct, *such paths are in bijection with the P_i-stack configurations of σ* by stacking one upon the other the configurations of the vertices of a path.
- For any vertex v of $\mathcal{G}^{(i)}$, there is a path between vertices of $V_1^{(i)}$ and $V_{s_i}^{(i)}$ going through v.

Take for example the permutation $\sigma = 4321$. There is only one right-to-left minimum, which is 1.

The sorting graph $\mathcal{G}^{(1)}$ for $\sigma = 4321$ encodes the P_1-stack configurations of σ, that are in particular pushall stack configurations of $\sigma^{(1)} = 432$. There are 8 different such configurations, which are:

$$\left|\begin{smallmatrix}2\\3\\4\end{smallmatrix}\right| \quad |2|\left|\begin{smallmatrix}3\\4\end{smallmatrix}\right| \quad |3|\left|\begin{smallmatrix}2\\4\end{smallmatrix}\right| \quad |4|\left|\begin{smallmatrix}2\\3\end{smallmatrix}\right| \quad \left|\begin{smallmatrix}3\\4\end{smallmatrix}\right||2| \quad \left|\begin{smallmatrix}2\\4\end{smallmatrix}\right||3| \quad \left|\begin{smallmatrix}2\\3\end{smallmatrix}\right||4| \quad \left|\begin{smallmatrix}2\\3\\4\end{smallmatrix}\right|$$

As the $\ominus$-decomposition of $\sigma^{(1)}$ is $\sigma^{(1)} = \ominus[4,3,2]$, the sorting graph $\mathcal{G}^{(1)}$ has 3 levels (see Figure 3).

Then the 8 P_1-stack configurations of σ are found taking each of the 8 different paths going from any configuration of B_1 to any configuration of B_3. For example, the thick path of Figure 3 gives the stack configuration $\begin{vmatrix}2\\4\end{vmatrix}\,\boxed{3}$ by stacking the selected configuration of B_3 above the configuration of B_2 and so on.

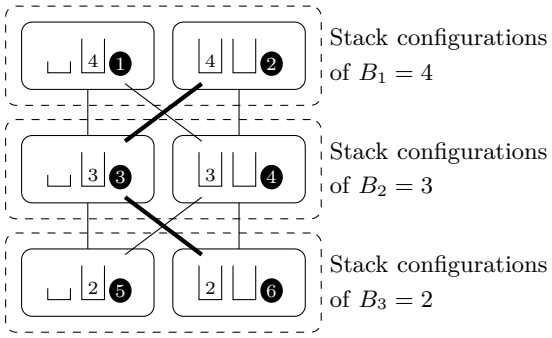


Figure 3 Sorting graph $\mathcal{G}^{(1)}$ of $\sigma = 4321$.

Our algorithm computes iteratively the graph $\mathcal{G}^{(i)}$ from $\mathcal{G}^{(i-1)}$ for i from 2 to r. The way $\mathcal{G}^{(i)}$ is computed from $\mathcal{G}^{(i-1)}$ depends on the relative values of $p^{(i)}$ and $q^{(i+1)}$. By definition of $\mathcal{G}^{(i)}$, if at any step $\mathcal{G}^{(i)}$ is empty, it means that $\sigma_{\leq k_i}$ is not sortable (from Theorem 9), so σ is not sortable either, and the algorithm returns `false`. This is summarized in Algorithm 2.

Algorithm 2: *isSortable*

Data: σ a permutation
Result: `true` or `false` depending on whether σ is 2-stack sortable
$\mathcal{G} \leftarrow ComputeG1$
for i *from* 2 *to* r **do**
  if $p^{(i)} = q^{(i+1)}$ **then** $\mathcal{G} \leftarrow iteratepEqualsq(\mathcal{G})$ or **return** `false`
  else if $p^{(i)} < q^{(i+1)}$ **then** $\mathcal{G} \leftarrow iteratepLessThanq(\mathcal{G})$ or **return** `false`
    else $\mathcal{G} \leftarrow iteratepGreaterThanq(\mathcal{G})$ or **return** `false`
return `true`

The rest of Section 3 describes the sub-procedures used in our main algorithm $isSortable(\sigma)$.

3.3 First step: $\mathcal{G}^{(1)}$

In this subsection, we show how to compute the P_1-stack configurations of σ, i.e. the stack configurations corresponding to time t_1 for sorting words of $\sigma_{\leq k_1}$ that satisfy (P) for $\sigma_{\leq k_1}$.

From Lemma 19, such a stack configuration is a pushall stack configuration of $\sigma^{(1)}$. Conversely, since $\sigma_{k_1} = 1$, $\sigma^{(1)} = \sigma_{<k_1}$ and each sorting word of $\sigma_{\leq k_1}$ satisfies (P_1) for $\sigma_{\leq k_1}$. Thus the set of P_1-stack configurations of σ is the set of pushall stack configurations of $\sigma^{(1)}$.

By Proposition 4.7 of [9], these stack configurations are described by the set of stack configurations for each block of the $\ominus$-decomposition of $\sigma^{(1)}$. More precisely, with $\sigma^{(1)} = \ominus[B_1^{(1)}, \ldots, B_{s_1}^{(1)}]$, there is a bijection from $pushallConfigs(B_1^{(1)}) \times \cdots \times pushallConfigs(B_{s_1}^{(1)})$ onto $pushallConfigs(\sigma^{(1)})$ by stacking configurations one upon the other (as in Lemma 11). Thus, from Lemma 18, $\sigma_{\leq k_1}$ is not sortable if and only if a set $pushallConfigs(B_j^{(1)})$ is empty.

Moreover, it will be useful to label the configurations computed so that we attach a distinct integer to each stack configuration when computed.

At this point, we have encoded all configurations corresponding to words satisfying (P) up to the factor $\rho_1\lambda_1\mu_1$. The obtained graph is $\mathcal{G}^{(1)}$. This step is summarized in Algorithm 3.

3.4 From step i to step $i+1$

After step i we know the graph $\mathcal{G}^{(i)}$ encoding every P_i-stack configuration of σ and we want to compute the graph $\mathcal{G}^{(i+1)}$ encoding P_{i+1}-stack configurations of σ at step $i+1$. From

Algorithm 3: ComputeG1

Data: σ a permutation, m a global integer variable
Result: `false` if $\sigma_{\leq k_1}$ is not sortable, the sorting graph $\mathcal{G}^{(1)}$ otherwise.
$E = \varnothing$; Compute $\sigma^{(1)}$ and its $\ominus$-decomposition $\ominus[B_1^{(1)}, \dots, B_{s_1}^{(1)}]$
for j *from* 1 *to* $s_1^{(1)}$ **do**
  $V_j^{(1)} \leftarrow \varnothing$; $S = pushallConfigs(B_j^{(1)})$
  if $S = \varnothing$ **then return** `false`
  for $s \in S$ **do** $\{ V_j^{(1)} \leftarrow V_j^{(1)} \bigcup \{(s, m)\}$; $m \leftarrow m + 1 \}$
  if $j > 1$ **then** $E = E \bigcup \{(s, s'), s \in V_j^{(1)}, s' \in V_{j-1}^{(1)}\}$
return $\mathcal{G}^{(1)} = (\bigcup_{j \in [1..s_1^{(1)}]} V_j^{(1)},\ E\ )$

Lemma 19 it is enough to check the accessibility of pushall stack configuration of $\sigma^{(i+1)}$ from P_i-stack configurations of σ. We cannot check every pair of configurations (c, c') with c being a P_i-stack configuration and c' be a pushall stack configuration of $\sigma^{(i+1)}$, because the number of such pair of configurations is possibly exponential. Thus our algorithm focuses not on stack configurations of some $\sigma^{(\ell)}$ but on the restriction of such stack configurations to the blocks $B_j^{(\ell)}$, making use of Lemma 14. Using Lemma 19, Lemma 14 can be rephrased as:

▶ **Lemma 20.** *Let c' be a total stack configuration of $\sigma^{(i+1)}$, $p = p^{(i)}$ and $q = q^{(i+1)}$. Then c' is a P_{i+1}-stack configuration of σ if and only if:*

- *For any $j \geq q$, $c'_{|B_j^{(i+1)}}$ is a pushall stack configuration of $\sigma_{|B_j^{(i+1)}}$, and*
- *there exists a P_i-stack configuration c of σ such that:*
  - $c'_{|B^{(i)}_{\min(p,q)} \cup \dots \cup B_q^{(i)}}$ *is accessible from* $c_{|B^{(i+1)}_{\min(p,q)} \cup \dots \cup B_p^{(i+1)}}$ *for* $\sigma_{|B_p^{(i)} \bigcup B_q^{(i+1)}}$ *and*
  - $c'_{|B_1^{(i+1)} \cup \dots \cup B^{(i+1)}_{\min(p,q)-1}} = c_{|B_1^{(i)} \cup \dots \cup B^{(i)}_{\min(p,q)-1}}$

Recall that a P_i-stack configuration of σ is encoded by a path in the sorting graph $\mathcal{G}^{(i)}$, corresponding to the $\ominus$-decomposition of the permutation $\sigma^{(i)}$ into blocks $B_j^{(i)}$. The last point of Lemma 20 ensures that the first levels (1 to $min(p^{(i)}, q^{(i+1)}) - 1$) in $\mathcal{G}^{(i+1)}$ are the same as the ones in $\mathcal{G}^{(i)}$. The first point of Lemma 20 ensures that the last levels ($> q^{(i+1)}$) of $\mathcal{G}^{(i+1)}$ form a complete partitioned graph whose vertices are all pushall stack configurations of the corresponding blocks. So the only unknown levels for $\mathcal{G}^{(i+1)}$ are those between $\min(p^{(i)}, q^{(i+1)})$ and $q^{(i+1)}$ and we can compute them by testing accessibility.

There are distinct cases depending on the relative values of $p^{(i)}$ and $q^{(i+1)}$. To lighten the notations in the following, we sometimes write p (resp. q) instead of $p^{(i)}$ (resp. $q^{(i+1)}$).

3.4.1 Case $p^{(i)} = q^{(i+1)}$

If $p^{(i)} = q^{(i+1)}$ then $B^{(i+1)}_{q^{(i+1)}} \cap A^{(i)} = B^{(i)}_{p^{(i)}} \cap A^{(i)}$ (see Figure 2). We have the sorting graph $\mathcal{G}^{(i)}$ encoding all P_i-stack configurations of σ and we want to compute the sorting graph $\mathcal{G}^{(i+1)}$ encoding all P_{i+1}-stack configurations of σ assuming that $p^{(i)} = q^{(i+1)} = min(p^{(i)}, q^{(i+1)})$.

In this case, from Lemma 20, we only have to check accessibility of pushall configurations of $B_q^{(i+1)}$ from configurations of $B_p^{(i)}$ belonging to level p of $\mathcal{G}^{(i)}$. Indeed, from the properties of the sorting graph given p.621, for any vertex v of $\mathcal{G}^{(i)}$, there is a path between vertices of $V_1^{(i)}$ and $V_{s_i}^{(i)}$ going through v, and such a path corresponds to a P_i-stack configuration of σ. Thus for any configurations x of $B_p^{(i)}$ belonging to a vertex v of level p of $\mathcal{G}^{(i)}$, there is at least one P_i-stack configuration c of σ such that $c_{|B_p^{(i)}} = x$, and $c_{|B_1^{(i)} \cup \dots \cup B^{(i)}_{\min(p,q)-1}}$ is encoded by a path from v to level p of $\mathcal{G}^{(i)}$ (which goes through each level $< p$).

If there is no pushall configuration of $B_q^{(i+1)}$ accessible from some configurations of $B_p^{(i)}$ belonging to level p of $\mathcal{G}^{(i)}$, or if $\sigma^{(i+1)}$ has no pushall configuration, then σ has no P_{i+1}-stack configuration and $\sigma_{\leq k_{i+1}}$ is not sortable (from Lemma 18). This leads to algorithm 4.

Algorithm 4: *iteratepEqualsq*($\mathcal{G}^{(i)}$)

Data: σ a permutation and $\mathcal{G}^{(i)}$ the sorting graph at step i
Result: `false` if $\sigma_{\leq k_{i+1}}$ is not sortable, the sorting graph $\mathcal{G}^{(i+1)}$ otherwise.
$\mathcal{G}$ an empty sorting graph with s_{i+1} levels
$\mathcal{G}' \leftarrow ComputeG1(\sigma^{(i+1)})$ (pushall sorting graph of $\sigma^{(i+1)}$) or **return** `false`
Copy levels $q+1, \ldots, s_{i+1}$ of $\mathcal{G}'$ into the same levels of $\mathcal{G}$
for (c, ℓ) *in level* p *of* $\mathcal{G}^{(i)}$ **do**
  $\mathcal{H}$ the subgraph of $\mathcal{G}^{(i)}$ induced by (c, ℓ) in levels $< p$
  for (c', ℓ') *in level* q *of* $\mathcal{G}'$ **do**
    if $isAccessible(c, c', \sigma_{|B_p^{(i)} \bigcup B_q^{(i+1)}})$ **then**
      Add (c', ℓ') in level q of $\mathcal{G}$ (if not already done)
      Merge $\mathcal{H}$ in levels $\leq q$ of $\mathcal{G}$ with (c', ℓ') as origin
if *level* q *of* $\mathcal{G}$ *is empty* **then return** `false`
for (c', ℓ') *in level* q *of* $\mathcal{G}$ **do** Add all edges from (c', ℓ') to each vertex of level $q+1$ of $\mathcal{G}$;
return $\mathcal{G}$

3.4.2 Case $p^{(i)} < q^{(i+1)}$

If $p^{(i)} < q^{(i+1)}$ then $B_{q^{(i+1)}}^{(i+1)} \cap A^{(i)} \subsetneq B_{p^{(i)}}^{(i)} \cap A^{(i)}$ (see Figure 2). By Lemma 20, we have to select among pushall stack configurations of blocks $p, p+1, \ldots, q$ of $\sigma^{(i+1)}$ those accessible from a configuration of $B_p^{(i)}$ that appears at level p in $\mathcal{G}^{(i)}$. We can restrict the accessibility test from configurations of $B_p^{(i)}$ appearing in graph $\mathcal{G}^{(i)}$ to pushall stack configurations of $B_q^{(i+1)}$. Indeed, Lemma 15 ensures that the elements of blocks $B_j^{(i+1)}$ for j from p to $q-1$ are in the same stack at time t_i and at time t_{i+1}. Thus configurations of $B_j^{(i+1)}$ for j from p to $q-1$ are restrictions of configurations of $B_p^{(i)}$. We keep the same label in the vertex to encode that those configurations of $B_p^{(i+1)}, B_{p+1}^{(i+1)}, \ldots, B_{q-1}^{(i+1)}$ come from the same configuration of $B_p^{(i)}$ and we build edges between vertices of $B_{j+1}^{(i+1)}$ and $B_j^{(i+1)}$ that come from the same configuration of $B_p^{(i)}$. It is because of this case $p = q$ that we have to label configurations in our sorting graph. Indeed, two different stack configurations c_1 and c_2 of $B_p^{(i)}$ may have the same restriction to some block $B_j^{(i+1)}$ but not be compatible with the same configurations of the other blocks, thus we want the corresponding vertices of level j of $\mathcal{G}^{(i+1)}$ to be distinct, that's why we use labels. More precisely, we have algorithm 5.

Algorithm 5: *iteratepLessThanq*($\mathcal{G}^{(i)}$)

Same as Algorithm 4, but replace this line:
  Merge $\mathcal{H}$ in levels $\leq q$ of $\mathcal{G}$ with (c', ℓ') as origin
by those four lines:
  for j *from* $q-1$ *downto* p **do**
    Add $(c_{|B_j^{(i+1)}}, \ell)$ in level j of $\mathcal{G}$
    Add an edge between $(c_{|B_j^{(i+1)}}, \ell)$ and $(c_{|B_{j+1}^{(i+1)}}, \ell)$ in $\mathcal{G}$.
  Merge $\mathcal{H}$ in levels $\leq p$ of $\mathcal{G}$ with $(c_{|B_p^{(i+1)}}, \ell)$ as origin

Note that in Algorithm 5, before calling $isAccessible(c, c', \sigma_{|B_p^{(i)} \bigcup B_q^{(i+1)}})$, we extend configuration c' to $D^{(i)} \bigcup B_q^{(i+1)}$ by assigning the same stack than in c to points of $D^{(i)} \setminus B_q^{(i+1)}$. This is justified by Lemma 15.

3.4.3 Case $p^{(i)} > q^{(i+1)}$

If $p^{(i)} > q^{(i+1)}$ then $B^{(i)}_{p^{(i)}} \cap A^{(i)} \subsetneq B^{(i+1)}_{q^{(i+1)}} \cap A^{(i)}$ (see Figure 2). This case is very similar to the preceding one except that $B_p^{(i)}$ is not cut into pieces but glued with preceding blocks. As a consequence, when testing accessibility of a configuration of $B_q^{(i+1)}$, we should consider every corresponding configuration in $\mathcal{G}^{(i)}$, that is, every configuration obtained by stacking configurations at level $q, q+1, \ldots, p$ in $\mathcal{G}^{(i)}$. Unfortunately, this may give an exponential number of configurations; but noticing that by Lemma 15 the elements of blocks $B_q^{(i)}, B_{q+1}^{(i)} \ldots B_{p-1}^{(i)}$ are exactly in the same stack at time t_i and at time t_{i+1}, it is sufficient to check the accessibility of a pushall configuration c' of $B_q^{(i+1)}$ from a configuration c of $B_p^{(i)}$ and verify afterwards whether the configuration c has ancestors in $\mathcal{G}^{(i)}$ that match exactly the configuration c'. Thus in Algorithm 6, before calling $isAccessible(c, c', \sigma_{|B_p^{(i)} \bigcup B_q^{(i+1)}})$, we extend configuration c to $D^{(i)} \bigcup B_p^{(i)}$ by assigning the same stack than in c' to points of $D^{(i)} \setminus B_p^{(i)}$.

Algorithm 6: $iteratepLessThanq(\mathcal{G}^{(i)})$

Same as Algorithm 4, but replace this block:

Add (c', ℓ') in level q of $\mathcal{G}$ (if not already done)

Merge $\mathcal{H}$ in levels $\leq q$ of $\mathcal{G}$ with (c', ℓ') as origin

by those four lines (and drop the definition of $\mathcal{H}$, since it is redefined below):

if *there is a path* $(c, \ell) \leftrightarrow (c'_{|B^{(i)}_{p-1}}, \ell_1) \leftrightarrow \ldots \leftrightarrow (c'_{|B^{(i)}_q}, \ell_k)$ *in* $\mathcal{G}^{(i)}$ **then**

- Add (c', ℓ') in level q of $\mathcal{G}$ (if not already done)
- $\mathcal{H}$ the subgraph of $\mathcal{G}^{(i)}$ induced by $(c'_{|B^{(i)}_q}, \ell_k)$ in levels $< q$
- Merge $\mathcal{H}$ in levels $\leq q$ of $\mathcal{G}$ with (c', ℓ') as origin

Now that we have described all steps of our algorithm, let us study its complexity.

4 Complexity Analysis

In this section we state the complexity of *isSortable(σ)*, our main algorithm (Algorithm 2).

▶ **Theorem 21.** *Given a permutation σ, Algorithm 2* isSortable(σ) *decides whether σ is sortable with two stacks in series in polynomial time w.r.t. $|\sigma|$.*

The key idea to prove this theorem relies on bounding the size of each graph $\mathcal{G}^{(i)}$:

▶ **Lemma 22.** *For any $i \in [1..r]$, the maximal number of vertices in a level of $\mathcal{G}^{(i)}$ is $9n$ where n is the size of the input permutation.*

▶ **Lemma 23.** *For any $i \in [1..r]$, the number of vertices of $\mathcal{G}^{(i)}$ is $\mathcal{O}(n^2)$ and the number of edges of $\mathcal{G}^{(i)}$ is $\mathcal{O}(n^3)$, where n is the size of the input permutation.*

References

1 Michael Albert, Mike D. Atkinson, and Steve Linton. Permutations generated by stacks and deques. *Annals of Combinatorics*, 14:3–16, 2010.
2 Mike D. Atkinson, M. M. Murphy, and N. Ruskuc. Sorting with two ordered stacks in series. *Theor. Comput. Sci.*, 289:205–223, October 2002.
3 Miklós Bóna. A survey of stack-sorting disciplines. *Electron. J. Combin.*, 9(2):16, 2003.
4 S. Kitaev. *Patterns in Permutations and Words*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2011.
5 D. E. Knuth. *The Art of Computer Programming, Volume I*. Addison-Wesley, 1968.
6 D. E. Knuth. *The Art of Computer Programming, Volume III*. Addison-Wesley, 1973.
7 Maximillian M. Murphy. *Restricted permutations, anti chains, atomic classes and stack sorting*. PhD thesis, University of St Andrews, 2002.
8 A. Pierrot. *Combinatoire et algorithmique dans les classes de permutations*. PhD thesis, Université Paris Diderot - Paris 7, 2013. (in English).
9 A. Pierrot and D. Rossin. 2-stack pushall sortable permutations, 2013. arxiv:1303.4376.
10 Robert E. Tarjan. Sorting using networks of queues and stacks. *J. ACM*, 19(2), 1972.
11 Julian West. Sorting twice through a stack. *Theor. Comput. Sci.*, 117(1&2):303–313, 1993.

Communication Lower Bounds for Distributed-Memory Computations*

Michele Scquizzato[†1] **and Francesco Silvestri**[2]

1 **Department of Computer Science, University of Pittsburgh, Pittsburgh, US**
scquizza@pitt.edu
2 **Department of Information Engineering, University of Padova, Padova, Italy**
silvest1@dei.unipd.it

Abstract

In this paper we propose a new approach to the study of the communication requirements of distributed computations, which advocates for the removal of the restrictive assumptions under which earlier results were derived. We illustrate our approach by giving tight lower bounds on the communication complexity required to solve several computational problems in a distributed-memory parallel machine, namely standard matrix multiplication, stencil computations, comparison sorting, and the Fast Fourier Transform. Our bounds rely only on a mild assumption on work distribution, and significantly strengthen previous results which require either the computation to be balanced among the processors, or specific initial distributions of the input data, or an upper bound on the size of processors' local memories.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Communication, lower bounds, distributed memory, parallel algorithms, BSP

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.627

1 Introduction

Communication is a major factor determining the performance of algorithms on current computing systems, as the time and energy needed to transfer data between processing and storage elements is often significantly higher than that of performing arithmetic operations. The gap between computation and communication costs, which is ultimately due to basic physical principles, is expected to become wider and wider as architectural advances allow to build systems of increasing size and complexity. Hence, the cost of data movement will play an even greater role in future years.

As in all endeavors where performance is systematically pursued, it is important to evaluate the distance from optimality of a proposed algorithmic solution, by establishing appropriate lower bounds. Given the well-known difficulty of establishing lower bounds, results are often obtained under restrictive assumptions that may severely limit their applicability. It is therefore important to progressively reduce or fully eliminate such restrictions.

In this spirit, we consider lower bounds on the amount of communication that is required to solve some classical computational problems on a distributed-memory parallel system.

* This work was supported, in part, by the University of Padova Project *CPDA121378* and by MIUR of Italy under project *AMANDA*.
† Supported, in part, by a fellowship of "Fondazione Ing. Aldo Gini", University of Padova, Italy. Most of this work was done while this author was a Ph.D. student at the University of Padova.



© Michele Scquizzato and Francesco Silvestri;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 627–638



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



Specifically, we revisit the assumptions and constraints under which preceding results were derived, and prove new lower bounds which use much weaker hypotheses and thus have wider applicability. Even when the functional form of the bounds remains the same, our results do yield new insights to algorithm developers since they might reveal if some settings are needed, or not, in order to obtain better performance.

We model the machine using the standard *Bulk Synchronous Parallel* (BSP) model of computation [32], which consists of a collection of p processors, each equipped with an unbounded private memory and communicating with each other through a communication network. The distribution of inputs and outputs effectively forms a part of the problem specification, thus restricting the applicability of upper and lower bounds. Much of previous work on BSP algorithms considers a version of the BSP model equipped with an additional *external memory*, which serves as the source of the input and the destination for the output (see, e.g., [29]). This modification significantly alters the spirit of the BSP of serving as a model for distributed-memory machines, making it very similar to shared-memory models like the LPRAM [1]. In fact, in a distributed-memory machine, the inputs might already be distributed in some manner prior to the invocation of the algorithm, and the outputs are usually left distributed in the processors' local memories at the end of the execution, especially if the computation is a subroutine of a larger computation. Thus, lower bounds that use this assumption, which essentially exploit this "hack" to guarantee that acquiring the n input elements contributes to the communication cost of algorithms (as some processor must read at least $\lceil n/p \rceil$ input values), are not directly applicable to distributed-memory architectures.

Other authors, within the original BSP model, assume specific distributions of the input data. As we shall see later, it is usually assumed that the input is initially *evenly* distributed among the p processors, that is, each processor is assigned either $\lceil n/p \rceil$ or $\lfloor n/p \rfloor$ pieces of the input. However, this apparently reasonable hypothesis is somewhat restrictive, and actually not part of the logic of the BSP model. In fact, the physical distribution of input data across the processors may depend on several factors, ranging from how the input data set gets acquired, to how the output of the preceding computation is distributed in the case of algorithms being cascaded (that is, when the output of one is the input for the next), to file system policies. Moreover, a uniform partition of the inputs postulates, but does not prove, that unbalanced distributions may cause severe communication bottlenecks.

One possibility to circumvent both the issues discussed above is to require, in place of the even distribution of the inputs and of the presence of an external memory, that algorithms exhibit some level of *load balancing* of the computation. Typically, if $\mathcal{W}$ denotes the total work required by any algorithm to solve the given problem, it is required that each processor performs $O(\mathcal{W}/p)$ elementary computations. However, this way it is implicitly assumed, but (as above) not proved, that optimal solutions balance computation. In fact, in general there is a tradeoff between computation costs and communication costs. Some papers (see, e.g., [22, 35]) quantify such tradeoffs by establishing lower bounds on the communication cost of any algorithm as a function of its computation time. Nevertheless, results of this kind usually indicate that the higher lower bounds on communication correspond only to perfectly (to within constant factors) work-balanced computations, and such bounds are tight since achieved by balanced algorithms. This leaves open the possibility that a substantial saving on communication costs could actually be achieved at a price of a small unbalance of the computation loads.

Another common assumption is putting an upper bound on the *size* of processors' local memories. However, current technological advances allow to build cheap memory and storage

devices that, for many applications, allow a single machine to store the whole input data set and the intermediate data. Moreover, results derived under this assumption are less general than results that put no limits on the amount of storage available to processors; indeed, lower bounds are relatively easier to establish, as the model essentially becomes a parallel version of the standard external memory model for sequential computations, for which much more results and techniques are known (see, e.g., [16, 2]).

In contrast, lower bounds presented in this paper do not hinge on any of the above assumptions. We develop new lower bounds for a number of key computational problems, namely standard matrix multiplication, stencil computations, comparison sorting, and the Fast Fourier Transform, using the weak assumption that no processor performs more than a constant fraction of the total required work. This requires more involved arguments, and substantially strengthens previous work on communication lower bounds for distributed-memory computations.

The model. The *Bulk Synchronous Parallel* (BSP) model of computation was introduced by Valiant [32] as a bridging model for general-purpose parallel computing. The architectural component of the model consists of p processing elements $P_0, P_1, \ldots, P_{p-1}$, each equipped with an unbounded local memory, interconnected by a communication medium. The execution of a BSP algorithm consists of a sequence of *supersteps*, where each processor can perform operations on data in its local memory, send/receive messages (each occupying a constant number of words) and, at the end, execute a global synchronization. The running time of the i-th superstep is expressed in terms of two parameters, g and ℓ, as $T_i = w_i + h_i g + \ell$, where w_i is the maximum number of local operations performed by any processor, and h_i is the maximum number of messages sent or received by any processor. The *running time* $T_{\mathcal{A}}$ of a BSP algorithm $\mathcal{A}$ is the sum of the times of its supersteps and can be expressed as $W_{\mathcal{A}} + H_{\mathcal{A}} g + S_{\mathcal{A}} \ell$, where $S_{\mathcal{A}}$ is the number of supersteps, $W_{\mathcal{A}} = \sum_{i=1}^{S_{\mathcal{A}}} w_i$ is the *local computation complexity*, and $H_{\mathcal{A}} = \sum_{i=1}^{S_{\mathcal{A}}} h_i$ is the *communication complexity*.

Previous work. The complexity of communication on various models of computation has received considerable attention. Lower bounds are often established through adaptations of the techniques of Hong and Kung [16] for hierarchical memory, or by critical path arguments, such as those in [1]. For applications of these and other techniques see [22, 2, 25, 15, 8, 6, 17, 24, 4, 9, 5] as well as [26] and references therein. In the following, we discuss previous work on lower bounds for the communication complexity of the problems studied in this paper.

A standard computational problem is the multiplication of two $n \times n$ matrices. For the classical $\Theta\left(n^3\right)$ algorithm, an $\Omega\left(n^2/p^{2/3}\right)$ lower bound has been previously derived for the BSP [31] and the LPRAM [1]. However, both results hinge on the hypothesis that the input initially resides outside the processors' local memories and thus must be read, contributing to the communication complexity of the algorithms. As such, these results are an immediate consequence of a result of [16] (then restated in [17]) which, loosely speaking, bounds from above the amount of computation that can be performed with a given quantity of data. When input is assumed to be initially evenly distributed across the p processors' local memories, the same lower bound is claimed in [11]. Recently, Ballard et al. [3] obtained a result of the same form by assuming perfectly balanced (to within constant factors) computations, and disallowing any initial replication of inputs. The very same bound was found also by Irony et al. [17], who restrict their attention to computations that take place on machines where processors' local memory size is assumed to be $M = O\left(n^2/p^{2/3}\right)$ (see also [4]). Finally, Solomonik and Demmel [28] investigate tradeoffs between input replication and communication complexity.

A class of computations ubiquitous in scientific computing is that of stencil computations, where each computing node in a multi-dimensional grid is updated with weighted values contributed by neighboring nodes. These computations include the *diamond* DAG in the two-dimensional case and the *cube* DAG in three dimensions. For the former, Papadimitriou and Ullman [22] present a communication-time tradeoff which yields a tight $\Omega(n)$ lower bound on the communication complexity only for the case of balanced computations. Aggarwal et al. [1] extend this result to all algorithms whose computational complexity is within a constant factor of the number of nodes of the DAG. To the best of our knowledge, this is the sole example of a tight lower bound that holds under the same hypothesis used in this paper. By generalizing the technique in [22], Tiskin [31] establishes a tight bound for the cube DAG, and claims its extension to higher dimensions. However, this results only hold when the computational load is balanced among the p processors.

Another key problem is sorting. Many papers assume that the n inputs initially reside outside processors' local memories, thus obtaining an $\Omega(n/p)$ lower bound which turns out to be tight when it is additionally assumed that problem instances have sufficient *slackness*, that is, $n >> p$ (e.g., $p^2 \leq n$ is a common assumption). Under some technical assumptions, a bound of the form $\Omega(n \log n/(p \log(n/p)))$, which is tight for all values of $p \leq n$, was first given within the LPRAM model [1]. This bound, however, includes the cost to read the input from the shared memory. A similar lower bound was derived later by Goodrich [15] within the BSP model, but the result holds only for the subclass of algorithms performing supersteps of degree $h = \Theta(n/p)$, and when the inputs are evenly distributed among the processors.

Previous work on the communication required to compute an FFT DAG of size n is similar to previous work for sorting. By exploiting the property that, as shown in [34], the cascade of three FFT networks has the topology of a full sorting network, the aforementioned lower bounds for sorting also hold for the FFT DAG. In a recent paper [9], we obtain the same result assuming that the maximum number of outputs held by any processor at the end of the algorithm is at most $n/2$, and without assumptions on the distribution of the input and of the computational loads; while these hypotheses are not equivalent to the one we are using in this paper, the result in [9] is the closest to the one that we will develop in Section 5.

Our contribution. In this paper we present lower bounds on the communication complexity required by key computational problems such as standard matrix multiplication, stencil computations, comparison sorting, and the Fast Fourier Transform, when solved by parallel algorithms on the BSP model. These results, which are all tight for the whole range of model parameters, rely on the hypothesis that no processor performs more than a *constant* fraction of the total required work. More formally, let $\mathcal{W}$ be the total work required by any algorithm to solve the given problem (if the problem is represented by a directed acyclic graph, then $\mathcal{W}$ is the number of nodes of the DAG, otherwise $\mathcal{W}$ is a lower bound on the computation time required by any sequential algorithm), and let W be the maximum amount of work performed by any BSP processor; then, W is assumed to satisfy the bound $W \leq \epsilon\mathcal{W}$, for some constant $\epsilon \in (0, 1)$. The rationale behind this approach is that communication is the major bottleneck of a distributed-memory computation unless the latter is sequential or "nearly sequential", in which case the main contribution to the running time T of an algorithm comes from computation. Since it is directly linked to the running time metric, and it does not allow for any other restrictive assumptions suggested by orthogonal constraints, we believe that this is the right approach to perform a systematic analysis of the communication requirements of distributed-memory computations.

We emphasize that, in contrast to previous work, our lower bounds do not count the communication required to acquire the input, allow for any initial distribution of the input among the processors' local memories, assume no upper bound on the sizes of the latter, and do not require computations to be balanced. On the other hand, some of our results make use of additional technical assumptions, such as the non-recomputation of intermediate results in the course of the computation, or some restrictions on the replication of input data. Such restrictions, however, were already in place in almost all of the corresponding state-of-the-art lower bounds.

A full version of the paper can be found in [27].

2 Matrix Multiplication

In this section we consider the problem of multiplying two $n \times n$ matrices, A and B, using only semiring operations, that is, addition and multiplication. Hence, each element $c_{i,j}$ of the output matrix C is an explicit sum of products $a_{i,k} \cdot b_{k,j}$, which are called *multiplicative terms*. This rules out, e.g., Strassen's algorithm and the Boolean matrix multiplication algorithm of Tiskin [30]. As shown in [19], any algorithm using only semiring operations must compute at least n^3 distinct multiplicative terms.

In this section we establish a lower bound on the communication complexity of any parallel algorithm for matrix multiplication on a BSP with p processors. This result is derived assuming that no processor performs more than a constant fraction of the n^3 total work required by any algorithm, measured as the number of scalar multiplications, and that each input element is initially stored in the local memory of exactly one processor. The bound has the form of $\Omega\left(W^{2/3}\right)$, where W is the maximum number of multiplicative terms evaluated by a processor, and is tight for all values of p between two and n^2. The argument through which we establish such a result is a repeated application of a "bandwidth" argument which, loosely speaking, is as follows. Consider a processor which performs the maximum amount of work. If this processor initially holds "few" input values, then, since it computes at least n^3/p multiplicative terms, it must receive "many" inputs from the submachine including the other processors; otherwise, if it initially holds "many" inputs, then it has to send many of them to the other processors, because it cannot perform too much work on its own, and thus the other processors have to perform at least a constant fraction of the total work. The lower bound applies to any distribution of input and output matrices, and only requires that the input matrices are not initially replicated.

Towards this end, we first establish a lower bound of $\Omega\left(n^2\right)$ under the same hypotheses outlined above for two processors. This result is derived using a bandwidth argument that bounds from below the amount of data that must travel across the communication network of a two-processor machine. A bound of the same form can be found in [17, Section 6], which holds only when the elements of the input matrices A and B are evenly, or almost evenly, distributed among the two processors. Our result, which instead allows any initial distribution of the input matrices (without replication), establishes the same bound by using a mild hypothesis on the maximum computation load faced by the processors. Due to space constraints, the proof of the result is deferred to the full version of the paper.

▶ **Lemma 1.** *Let $\mathcal{A}$ be any algorithm for computing the matrix product $C = AB$, using only semiring operations, on a BSP with two processors. If each processor computes at most ϵn^3 multiplicative terms, where ϵ is an arbitrary constant in $[1/2, 1)$, and the input matrices are not initially replicated, then the communication complexity of the algorithm is*

$$H_{\mathcal{A}}(n, p) = \Omega\left(n^2\right).$$

Now we can prove the main result of this section. The following theorem establishes an $\Omega\left(W^{2/3}\right)$ lower bound to the communication complexity of any standard algorithm, where W denotes the maximum number of multiplicative terms evaluated by a processor. By the result of [19] and by the pigeonhole principle, there exists a processor that computes at least n^3/p multiplicative terms, from which the standard $\Omega\left(n^2/p^{2/3}\right)$ lower bound follows.

▶ **Theorem 2.** *Let $\mathcal{A}$ be any algorithm for computing the matrix product $C = AB$, using only semiring operations, on a BSP with p processors, where $1 < p \leq n^2$, and let W be the maximum number of multiplicative terms evaluated by a processor. If $W \leq \max\{n^3/p, n^3/11^3\}$, and the input matrices are not initially replicated, then the communication complexity of the algorithm is*

$$H_{\mathcal{A}}(n,p) = \Omega\left(W^{2/3}\right).$$

Proof. Without loss of generality, we assume that any multiplicative term computed by the processors is actually used towards the computation of some entry of the output matrix C (that is, processors do not perform "useless" computations). Consider one of the processors that compute W multiplicative terms, and without loss of generality let P_0 denote such a processor. Let I be the number of input elements initially held by this processor in its local memory.

Consider first the case $I \leq W^{2/3}/5$. By [16, Lemma 6.1], a processor that computes W multiplicative terms either accesses, during the whole execution of algorithm $\mathcal{A}$, at least $(W/2)^{2/3}$ input elements, or computes multiplicative terms relative to at least $(W/2)^{2/3}$ elements of the output matrix. In the first case, since P_0 initially holds $I \leq W^{2/3}/5$ input elements, it must receive at least $(W/2)^{2/3} - I = \Omega\left(W^{2/3}\right)$ data words from other processors, and the theorem follows. On the other hand, suppose P_0 computes multiplicative terms relative to $(W/2)^{2/3}$ entries of the output matrix, and partition such entries into three groups: G_1, the set of entries whose multiplicative terms have all been computed by the processor; G_2, the set of entries produced by the processor but for which some multiplicative term or partial sum has been communicated by some other processor; G_3, the set of entries not produced by the processor. Clearly, at least one of these three groups must have size at least $(W/2)^{2/3}/3$. If $|G_1| \geq (W/2)^{2/3}/3$, then P_0 must have computed at least $n(W/2)^{2/3}/3$ multiplicative terms, and since any entry of the input matrices occurs in only n of such terms, the processor must have received $(W/2)^{2/3}/3 - I = \Omega\left(W^{2/3}\right)$ elements from other processors. A similar argument applies to both G_2 and G_3.

Now suppose $I > W^{2/3}/5$ and $p \geq 11^3$. Note that in this case, since $p \geq 11^3$, it holds that $W \leq n^3/11^3$. Assume, without loss of generality, that P_0 initially holds at least $I/2$ elements of matrix A. Since any entry of the input matrices occurs in n multiplicative terms, there are at least $In/2$ multiplicative terms that depend on the entries of A initially held by the processor. Since W multiplicative terms are computed by the processor, the remaining $In/2 - W \geq W^{2/3}n/10 - W^{2/3}n/11 = W^{2/3}n/110$ ones are computed by other processors. Since, by hypothesis, each entry of A is initially non-replicated and a processor can compute at most n multiplicative terms using a single entry of A, we have that $(In/2 - W)/n \geq W^{2/3}/110$ messages are required for sending the appropriate entries of A to the processors that will compute the remaining entries. Hence, $H_{\mathcal{A}}(n,p) \geq W^{2/3}/110$.

Finally, when $I > W^{2/3}/5$ and $p < 11^3$, the sought lower bound follows by Lemma 1. Indeed, the p processors can be virtually partitioned into two subsets, each consisting of exactly $p/2$ processors; in particular, processor P_0^* will be identified with the submachine including the first half of the p processors, and P_1^* with the submachine including the second half. Since $p < 11^3$, by hypothesis each BSP processor computes at most n^3/p multiplicative

terms, and thus both P_0^* and P_1^* compute at most $(n^3/p)(p/2) = n^3/2$ multiplicative terms overall. Hence we can apply Lemma 1 to processors P_0^* and P_1^*, obtaining the desired result. ◀

The proposed bound is tight and is matched by the algorithm that decomposes the problem into $n^3/W \leq p$ subproblems of size $W^{1/3} \times W^{1/3}$, and then solves each subproblem sequentially in each round. Since $W \geq n^3/p$, the minimum communication complexity is $\Omega\left(n^2/p^{2/3}\right)$, which is achieved by the standard 3D algorithm (see, e.g., [17]).

3 Stencil Computations

A *stencil* defines the computation of an element in a $(d-1)$-dimensional spatial grid at time t as a function of neighboring grid elements at time $t-1, \ldots, t-\tau$, for some value $\tau \geq 1$ and constant $d > 1$ (see, e.g., [14]). We provide an $\Omega\left(n^{d-1}/p^{(d-2)/(d-1)}\right)$ lower bound to the communication complexity of any algorithm evaluating n time steps of a $(d-1)$-dimensional stencil. For simplicity we assume $\tau = 1$, however our bounds still apply in the general case. The bound follows by investigating the (n, d)-*stencil problem*, which consists in evaluating all nodes of a d-dimensional array DAG of size n. A *d-dimensional array DAG* has n^d nodes $\langle i_0, \ldots, i_{d-1}\rangle$, for each $0 \leq i_0, \ldots, i_{d-1} < n$, and there is an arc from $\langle i_0, \ldots, i_k, \ldots, i_{d-1}\rangle$ to $\langle i_0, \ldots, i_k + 1, \ldots, i_{d-1}\rangle$, for each $0 \leq k < d$ and $0 \leq i_0, \ldots, i_{d-1} < n-1$. Observe that $\langle 0, \ldots, 0\rangle$ and $\langle n-1, \ldots, n-1\rangle$ are the single input and output nodes, respectively. A lower bound to the (n, d)-stencil problem applies to the computation of n steps of a stencil: indeed, the DAG given by the $(d-1)$-dimensional grid plus the time dimension spans a d-dimensional spacetime containing an $(n/2, d)$-array as a subgraph.

Our result hinges on the restriction on the nature of the computation whereby each vertex of the DAG is computed exactly once. In this setting, the crucial property is that for each arc (u, v) such that u is computed by processor P and v is computed by processor P', $P \neq P'$, there corresponds a message from P to P' (which may also cross other processors). Such arcs are referred to as *communication arcs*.

We now introduce some preliminary definitions, which will be used throughout the section. We envision an (n, d)-stencil DAG as partitioned into $p^{d/(d-1)}$ smaller d-dimensional arrays, called *blocks*, of size $n/p^{1/(d-1)}$, and denote each block with $B_{i_0,\ldots,i_{d-1}}$ for $0 \leq i_0, \ldots, i_{d-1} < p^{1/(d-1)}$. Block $B_{i_0,\ldots,i_{d-1}}$ contains nodes $\langle i'_0, \ldots, i'_{d-1}\rangle$, for each $i_k n/p^{1/(d-1)} \leq i'_k < (i_k + 1)n/p^{1/(d-1)}$. A block has $n^d/p^{d/(d-1)}$ nodes, and is said *ℓ-owned* if more than half of its nodes are evaluated by processor P_ℓ, with $0 \leq \ell < p$. A block is *owned* if there exists some ℓ, with $0 \leq \ell < p$, such that it is ℓ-owned; it is *shared* otherwise. Two blocks $B_{i_0,\ldots,i_{d-1}}$ and $B_{i'_0,\ldots,i'_{d-1}}$ are said to be *adjacent* if their coordinates differ in just one position k and $|i_k - i'_k| = 1$ (i.e., they share a face). For the sake of simplicity, we assume that n and p are powers of 2^{d-1} and thus the previous values (e.g., $n/p^{1/(d-1)}$) are integral: since d is a constant, this assumption is verified by suitably increasing n and decreasing p by a constant factor which does not asymptotically affect our lower bounds.

In order to establish our main lower bound, we need two preliminary lemmas (whose proofs are deferred to the full version). The first one gives a slack lower bound based on the d-dimensional version of the Loomis-Whitney geometric inequality [21], and resembles the result of Theorem 2 for matrix multiplication when $d = 3$.

▶ **Lemma 3.** *Let $\mathcal{A}_d$ be any algorithm solving the (n, d)-stencil problem, without recomputation, on a BSP with p processors, where $1 < p \leq n^{d-1}$, and denote with W the maximum number of nodes evaluated by a processor. If $W \leq \epsilon n^d$, for an arbitrary constant $\epsilon \in (0, 1)$,*

then the communication complexity of the algorithm is

$$H_{\mathcal{A}_d}(n,p) = \Omega\left(W^{(d-1)/d}\right).$$

Now we need a second lemma that bounds from below the number of messages exchanged by a processor P_ℓ while evaluating nodes in an ℓ-owned block and in an adjacent block which is not ℓ-owned.

▶ **Lemma 4.** *Consider an ℓ-owned block B adjacent to a shared or ℓ'-owned block B', with $\ell \neq \ell'$. Then, the number of messages exchanged by processor P_ℓ for evaluating, without recomputation, nodes in B and B' is $\Omega\left(\frac{n^{d-1}}{p}\right)$.*

The next theorem gives the claimed $\Omega\left(n^{d-1}/(p^{(d-2)/(d-1)})\right)$ lower bound, and its proof is inspired by the argument in [31] for the cube DAG (which however assumes balanced work). The lower bound is matched by the balanced algorithm given in [31], which decomposes the (n,d)-stencil into $p^{d/(d-1)}$ subDAGs of dimension d and size $n/p^{1/(d-1)}$. The proof of the theorem is deferred to the full version.

▶ **Theorem 5.** *Let $\mathcal{A}_d$ be any algorithm for solving the (n,d)-stencil problem, without recomputation, on a BSP with p processors, where $1 < p \leq n^{d-1}$, and let W be the maximum number of nodes evaluated by a processor. If $W \leq \epsilon n^d$, for an arbitrary constant $\epsilon \in (0,1)$, then the communication complexity of the algorithm is*

$$H_{\mathcal{A}_d}(n,p) = \Omega\left(\frac{n^{d-1}}{p^{(d-2)/(d-1)}}\right).$$

4 Sorting

In this section we give a lower bound to the communication complexity of comparison-based sorting algorithms. Comparison sorting is defined as the problem in which a given set X of n input keys from an ordered set has to be sorted, such that the only operations allowed on members of X are pairwise comparisons. Our bound only requires that no processor does more than a constant fraction ϵ of the $\Theta(n \log n)$ comparisons required by any comparison sorting algorithm, for any $\epsilon \in (0,1)$, and does not impose any protocol on the distribution of the inputs and the outputs on the processors, nor upper bounds to the size of their local memories, or specific communication patterns. As for previous work, we still need the technical assumptions that the inputs are not initially replicated, and that the processors store only a constant number of copies of any input key at any moment during the execution of the algorithm.

The main result follows from the application of two lemmas, each of which provides a different and independent lower bound to the communication complexity of sorting. Both rely on non-trivial counting arguments, adapted from [2, 1], that hinge on the fact that any comparison sorting algorithm must be able to distinguish between all the $n!$ permutations of the n inputs. The first lemma provides a lower bound as a function of the maximum number S of input keys initially held by a processor. The second gives a lower bound as a function of the number Π of permutations that can be distinguished before any communications take place. We begin by stating the first lemma.

▶ **Lemma 6.** *Let $\mathcal{A}$ be any algorithm sorting n keys on a BSP with p processors, with $1 < p \leq n$, and let S denote the maximum number of input keys initially held by a processor. If each processor performs at most $\epsilon(n \log n)$ comparisons, with ϵ being an arbitrary constant*

in $(0,1)$*, and the input is not initially replicated, then the communication complexity of the algorithm is*

$$H_{\mathcal{A}}(n,p) = \Omega(S).$$

We now provide a second lemma, which bounds from below the communication complexity of sorting in BSP as a function of the number Π of permutations that can be distinguished before any communications take place, that is, when processors can only compare their local inputs.

▶ **Lemma 7.** *Let* $\mathcal{A}$ *be any algorithm sorting* n *keys on a BSP with* p *processors, where* $1 < p \leq n$*, and let* Π *be the number of distinct permutations that can be distinguished by* $\mathcal{A}$ before *the second superstep, that is, by comparing the inputs that (possibly) reside initially in the processors' local memories. If* $\mathcal{A}$ *stores only a constant number of copies of any key at any time instant, then the communication complexity of the algorithm is*

$$H_{\mathcal{A}}(n,p) = \Omega\left(\frac{n\log n - \log\Pi}{p\log(n/p)}\right).$$

Now we are ready to prove the main result of this section, an $\Omega((n\log n)/(p\log(n/p)))$ lower bound to the communication complexity of any comparison sorting algorithm. The result follows by combining the bounds given by the previous two lemmas. Both bounds are not tight when considered independently, the first (Lemma 6) because it is weak when at the beginning the input keys tend to be distributed evenly among the processors, the second (Lemma 7) because it is weak when the input keys tend to be concentrated on one or few processors. However, the simultaneous application of both provides the sought (tight) lower bound. Once again, the proof of the following theorem is deferred to the full version.

▶ **Theorem 8.** *Let* $\mathcal{A}$ *be any algorithm for sorting* n *keys on a BSP with* p *processors, with* $1 < p \leq n$*. If each processor performs at most* $\epsilon(n\log n)$ *comparisons, with* ϵ *being an arbitrary constant in* $(0,1)$*, the inputs are not initially replicated, and the* p *processors store only a constant number of copies of any key at any time instant, then the communication complexity of the algorithm is*

$$H_{\mathcal{A}}(n,p) = \Omega\left(\frac{n\log n}{p\log(n/p)}\right).$$

5 Fast Fourier Transform

In this section we consider the problem of computing the Discrete Fourier Transform of n values using the n-input FFT DAG. In the FFT DAG, a vertex is a pair $\langle w,l\rangle$, with $0 \leq w < n$ and $0 \leq l \leq \log n$, and there exists an arc between two vertices $\langle w,l\rangle$ and $\langle w',l'\rangle$ if $l' = l+1$, and either w and w' are identical or their binary representations differ exactly in the l'-th bit. We show that, when no processor computes more than a constant fraction of the total number of vertices of the DAG, the communication complexity is $\Omega(n\log n/(p\log(n/p)))$. Our bound does not assume any particular I/O protocol, and only requires that every input resides in the local memory of exactly one processor before the computation begins; as for preceding results, our bound also hinges on the restriction on the nature of the computation whereby each vertex of the FFT DAG is computed exactly once. The bound is tight for any $p \leq n$, and is achieved by the well-known recursive decomposition of the DAG into two sets of smaller $\sqrt{n}$-input FFT DAGs with each set containing $\sqrt{n}$ of such subDAGs (see, e.g., [7]).

We will first establish a lemma which, under the same hypothesis of the main result, provides a lower bound to the communication complexity as a function of the maximum work performed by any processor. The proof of the lemma (which, for space limitations, is deferred to the full version) is based on a bandwidth argument, which exploits the fact that an FFT DAG can perform all cyclic shifts (see, e.g., [20]), and on the following technical result which is implicit in the work of Hong and Kung (a simplified proof is due to Aggarwal and Vitter [2]).

▶ **Lemma 9** ([16]). *Consider the computation of the n-input FFT DAG. During the computation, if a processor accesses at most S nodes of the DAG, then it can evaluate at most $2S \log S$ nodes, for any $S \geq 2$.*

▶ **Lemma 10.** *Let $\mathcal{A}$ be any algorithm computing, without recomputation, an n-input FFT DAG on a BSP with p processors, with $1 < p \leq n$, and let W be the maximum number of nodes of the FFT DAG computed by a processor. If $W \leq \epsilon(n \log n)$, for an arbitrary constant in $(0, 1)$, and the inputs are not initially replicated, then the communication complexity of the algorithm is*

$$H_{\mathcal{A}}(n, p) = \Omega\left(\frac{W}{\log W}\right).$$

The main result of this section follows by a simple application of the preceding lemma and of a result implicit in the proof of the lower bound due to Bilardi et al. [9, Corollary 1]. The proof is deferred to the full version.

▶ **Theorem 11.** *Let $\mathcal{A}$ be any algorithm computing, without recomputation, an n-input FFT DAG on a BSP with p processors, where $1 < p \leq n$. If each processor computes at most $\epsilon(n \log n)$ nodes, for an arbitrary constant in $(0, 1)$, of the FFT DAG and the inputs are not initially replicated, then the communication complexity of the algorithm is*

$$H_{\mathcal{A}}(n, p) = \Omega\left(\frac{n \log n}{p \log(n/p)}\right).$$

6 Conclusions

We have presented new lower bounds on the amount of communication required to solve some key computational problems in distributed-memory parallel architectures. All our bounds have the same functional form of previous results that appear in the literature; however, the latter are built by making a critical use of some assumptions that rule out a large part of possible algorithms. The novelty and the significance of our results stem from the assumptions under which our lower bounds are developed, which are much weaker than those used in previous work.

Our bounds are derived within the BSP model of computation, but can be easily extended to other models for distributed computations based on or similar to the BSP, such as LogP [13] and MapReduce [18, 23]. Moreover, we believe that our results can be also ported to models for multicore computing (see, e.g., [10, 33, 12]), since our proofs are based on some techniques that have already been exploited in this scenario.

There is still much to do towards the establishment of a definitive theory of communication-efficient algorithms. In fact, we were not able to remove all the restrictions there were in place in previous work: in some cases our lower bounds still make use of some technical assumptions, such as the non-recomputation of intermediate results, or restrictions on the replication of

input data. Although it seems that such restrictions can be relaxed to encompass a small amount of recomputation or input replication, it is an open question to assess whether these assumptions are inherent to our proof techniques or can be removed. In particular, it is not clear, in general, when recomputation has the power to reduce communications, since many lower bound techniques do not apply in this more general scenario (see, e.g., [5]). Providing tight lower bounds that hold also when recomputation is allowed is a fascinating and challenging avenue for future research.

Acknowledgements. The authors would like to thank Gianfranco Bilardi and Andrea Pietracaprina for useful comments.

References

1 Alok Aggarwal, Ashok K. Chandra, and Marc Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71:3–28, 1990.

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

3 Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. In *Proc. 24th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 77–79, 2012.

4 Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. on Matrix Analysis and Applications*, 32(3):866–901, 2011.

5 Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Graph expansion and communication costs of fast matrix multiplication. *J. of the ACM*, 59(6):32:1–32:23, 2012.

6 Gianfranco Bilardi, Andrea Pietracaprina, and Paolo D'Alberto. On the space and access complexity of computation DAGs. In *Proc. 26th International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 47–58, 2000.

7 Gianfranco Bilardi, Andrea Pietracaprina, Geppino Pucci, and Francesco Silvestri. Network-oblivious algorithms. In *Proc. 21st IEEE International Parallel and Distributed Processing Symposium*, pages 1–10, 2007.

8 Gianfranco Bilardi and Franco Preparata. Processor-time tradeoffs under bounded-speed message propagation: Part II, lower bounds. *Theory of Computing Systems*, 32(5):531–559, 1999.

9 Gianfranco Bilardi, Michele Scquizzato, and Francesco Silvestri. A lower bound technique for communication on BSP with application to the FFT. In *Proc. 18th International Conference on Parallel Processing*, pages 676–687, 2012.

10 Guy E. Blelloch, Rezaul A. Chowdhury, Phillip B. Gibbons, Vijaya Ramachandran, Shimin Chen, and Michael Kozuch. Provably good multicore cache performance for divide-and-conquer algorithms. In *Proc. 19th ACM-SIAM Symposium on Discrete Algorithms*, pages 501–510, 2008.

11 Thomas Cheatham, Amr F. Fahmy, Dan C. Stefanescu, and Leslie G. Valiant. Bulk synchronous parallel computing – a paradigm for transportable software. In *Proc. 28th Hawaii International Conference on System Sciences*, pages 268–275, 1995.

12 Rezaul Alam Chowdhury, Vijaya Ramachandran, Francesco Silvestri, and Brandon Blakeley. Oblivious algorithms for multicores and networks of processors. *Journal of Parallel and Distributed Computing*, 73(7):911–925, 2013.

13 David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Eunice E. Santos, Klaus E. Schauser, Ramesh Subramonian, and Thorsten von Eicken. LogP: A practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, 1996.

14 Matteo Frigo and Volker Strumpen. Cache oblivious stencil computations. In *Proc. 19th International Conference on Supercomputing*, pages 361–366, 2005.

15 Michael T. Goodrich. Communication-efficient parallel sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999.

16 Jia-Wei Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proc. 13th ACM Symposium on Theory of Computing*, pages 326–333, 1981.

17 Dror Irony, Sivan Toledo, and Alexandre Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.

18 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 938–948, 2010.

19 Leslie Robert Kerr. *The Effect of Algebraic Structure on the Computational Complexity of Matrix Multiplication*. PhD thesis, Cornell University, 1970.

20 Frank T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers Inc., 1992.

21 L.H. Loomis and H. Whitney. An inequality related to the isoperimetric inequality. *Bulletin of The American Mathematical Society*, 55:961–962, 1949.

22 Christos H. Papadimitriou and Jeffrey D. Ullman. A communication-time tradeoff. *SIAM Journal on Computing*, 16(4):639–646, 1987.

23 Andrea Pietracaprina, Geppino Pucci, Matteo Riondato, Francesco Silvestri, and Eli Upfal. Space-round tradeoffs for MapReduce computations. In *Proc. 26th International Conference on Supercomputing*, pages 235–244, 2012.

24 Desh Ranjan, John Savage, and Mohammad Zubair. Strong I/O lower bounds for binomial and FFT computation graphs. In *Proc. 17th Annual International Conference on Computing and Combinatorics*, pages 134–145, 2011.

25 John E. Savage. Extending the Hong-Kung model to memory hierarchies. In *Proc. First Annual International Conference on Computing and Combinatorics*, pages 270–281, 1995.

26 John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley Longman Publishing Co., Inc., 1998.

27 Michele Scquizzato and Francesco Silvestri. Communication lower bounds for distributed-memory computations. *CoRR*, abs/1307.1805, 2013.

28 Edgar Solomonik and James Demmel. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In *Proc. 17th International Conference on Parallel Processing*, pages 90–109, 2011.

29 Alexander Tiskin. BSP (bulk synchronous parallelism). In *Encyclopedia of Parallel Computing*, pages 192–199. Springer, 2011.

30 Alexandre Tiskin. Bulk-synchronous parallel multiplication of Boolean matrices. In *Proc. 25th Int'l Colloquium on Automata, Languages and Programming*, pages 494–506, 1998.

31 Alexandre Tiskin. *The Design and Analysis of Bulk-Synchronous Parallel Algorithms*. PhD thesis, University of Oxford, 1998.

32 Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

33 Leslie G. Valiant. A bridging model for multi-core computing. *Journal of Computer and System Sciences*, 77(1):154–166, 2011.

34 Chuan-Lin Wu and Tse-Yun Feng. The universality of the shuffle-exchange network. *IEEE Transactions on Computers*, 30:324–332, 1981.

35 I-Chen Wu and H. T. Kung. Communication complexity for parallel divide-and-conquer. In *Proc. 32nd annual Symposium on Foundations of Computer Science*, pages 151–162, 1991.

Stochastic Scheduling on Unrelated Machines *

Martin Skutella[1], Maxim Sviridenko[2], and Marc Uetz[3]

1 **TU Berlin, Institut für Mathematik, Berlin, Germany**
`martin.skutella@tu-berlin.de`
2 **University of Warwick, Department of Computer Science, Coventry, United Kingdom**
`sviri@dcs.warwick.ac.uk`
3 **University of Twente, Department of Applied Mathematics, Enschede, The Netherlands**
`m.uetz@utwente.nl`

Abstract

Two important characteristics encountered in many real-world scheduling problems are heterogeneous processors and a certain degree of uncertainty about the sizes of jobs. In this paper we address both, and study for the first time a scheduling problem that combines the classical unrelated machine scheduling model with stochastic processing times of jobs. Here, the processing time of job j on machine i is governed by random variable P_{ij}, and its realization becomes known only upon job completion. With w_j being the given weight of job j, we study the objective to minimize the expected total weighted completion time $\mathbb{E}\left[\sum_j w_j C_j\right]$, where C_j is the completion time of job j. By means of a novel time-indexed linear programming relaxation, we compute in polynomial time a scheduling policy with performance guarantee $(3+\Delta)/2+\varepsilon$. Here, $\varepsilon > 0$ is arbitrarily small, and Δ is an upper bound on the squared coefficient of variation of the processing times. When jobs also have individual release dates r_{ij}, our bound is $(2+\Delta)+\varepsilon$. We also show that the dependence of the performance guarantees on Δ is tight. Via $\Delta = 0$, currently best known bounds for deterministic scheduling on unrelated machines are contained as special case.

1998 ACM Subject Classification F.2.2 Sequencing and Scheduling, G.2.1 Combinatorial Algorithms, G.1.6 Optimization

Keywords and phrases Stochastic Scheduling, Unrelated Machines, Approximation Algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.639

1 Introduction

Deterministic scheduling. The problem to minimize the total weighted completion time on unrelated parallel machines, denoted $\mathrm{R}\,|\,(r_{ij})\,|\,\sum w_j C_j$ in the three-field notation of Graham et al. [8], is one of the most important classical problems in the theory of deterministic scheduling. Each job j has a weight w_j, possibly an individual release date r_{ij} before which job j must not be scheduled on machine i, and the processing time of job j on machine i is p_{ij}. Each job has to be processed non preemptively on any one of the machines, and each machine can process at most one job at a time. The objective is to find a schedule minimizing

* The first author was supported by the DFG Research Center Matheon "Mathematics for key technologies" in Berlin and by the DFG Focus Program 1307 within the project "Algorithm Engineering for Real-time Scheduling and Routing". The second author was supported by EPSRC grants EP/J021814/1, EP/D063191/1, FP7 Marie Curie Career Integration Grant and Royal Society Wolfson Research Merit Award. The third author was supported by the Centre for Telematics and Information Technology (CTIT) of the University of Twente.



© Martin Skutella, Maxim Sviridenko, and Marc Uetz;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 639–650



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



the total weighted completion time $\sum_j w_j C_j$, where C_j denotes the completion time of job j in the schedule. The special case with identical parallel machines is already known to be strongly NP-hard [12] but there do exist polynomial time approximation schemes [1, 28]. The general setting of unrelated parallel machines turns out to be significantly harder and there is a complexity gap compared to identical parallel machines: Hoogeveen et al. [11] prove MaxSNP-hardness and hence there is no polynomial time approximation scheme. On the positive side, the currently best known approximation algorithms for unrelated parallel machines have performance guarantees 3/2 and 2, for the problem without and with release dates, respectively [4, 23, 25, 26]. Improving these bounds is considered to be among the most important open problems in scheduling [24] which is also an indication of the high significance of unrelated machine scheduling.

Stochastic scheduling. We consider for the first time the stochastic variant of unrelated machine scheduling. Here, the processing time of a job j on machine i is given by random variable P_{ij}. In stochastic scheduling, we are asked to compute a non-anticipatory scheduling policy. Roughly spoken, a scheduling policy makes scheduling decisions at certain decision times t, and these decisions are based on the observed past up to time t as well as the a priori knowledge of the input data of the problem. The policy, however, must not anticipate information about the future, such as the actual realizations of the processing times of jobs which have not yet been completed by time t. We refer to Möhring et al. [16] for the formal definition of stochastic scheduling policies, and here confine ourselves with an intuitive description that puts stochastic scheduling in the framework of stochastic dynamic optimization: Actions of a scheduling policy at a time t consists of a set of jobs, possibly empty, to be started on a set of idle machines, together with a tentative next decision time $t^* > t$. The next action of the policy is due at t^*, or the time of the next job completion, or the time when the next job is released, whatever occurs first. Depending on the action of the policy, the next decision time as well as the state of the schedule at the next decision time is realized according to the probability distributions of the jobs' processing times. A non-anticipatory policy may learn over time, but it has only access to distributional information about remaining processing times of unfinished jobs, conditioned on the state of the schedule at time t.[1] As all previous work in the area, we assume that the random variables P_{ij} are stochastically independent across jobs. For any given non-anticipatory scheduling policy, the possible outcome of the objective function $\sum_j w_j C_j$ is a random variable, and our goal is to minimize its expected value, which by linearity of expectation equals $\sum_j w_j \mathbb{E}[C_j]$.

Related work. Generalizing a well known result of Smith [29] for deterministic single machine scheduling, Rothkopf [19] proved in 1966 that the WSEPT rule[2] minimizes the expected total weighted completion time on a single machine. Apart from Weiss' results on the asymptotic optimality of WSEPT in stochastic scheduling on identical parallel machines [32, 33], the first constant factor approximation algorithms for stochastic scheduling on identical parallel machines have been obtained in 1999 by Möhring et al. [17]. Next to a linear programming (LP) based analysis of the WSEPT rule, they define list scheduling

[1] A concrete example may help: Imagine a job j which has processing time either small (ε) or large (M), both with probability 1/2. For a scheduling policy that starts this job at time t, it can make sense to define a tentative next decision time at $t^* = t + \varepsilon$, because then it learns with certainty what the actual processing time of job j is. Using such building blocks, one can even show that an optimal scheduling policy is generally not work conserving, i. e., machines are left deliberately idle [31].

[2] Weighted shortest expected processing time first: schedule jobs in order of non-increasing ratios $w_j/\mathbb{E}[P_j]$.

Table 1 Performance bounds for nonpreemptive stochastic machine scheduling problems. Parameter $\varepsilon > 0$ can be chosen arbitrarily small. Parameter Δ upper bounds the squared coefficient of variation $\mathbb{CV}^2[P_{ij}] = \mathbb{V}\mathrm{ar}[P_{ij}]/\mathbb{E}^2[P_{ij}]$ for all P_{ij}. The third column shows the results for $\mathbb{CV}[P_{ij}] \leq 1$; e. g., uniform, exponential, or Erlang distributions. As usual in stochastic scheduling, these bounds hold with respect to the expected performance of any non-anticipatory scheduling policy.

stochastic scheduling model	worst case performance guarantee arbitrary P_{ij}	$\mathbb{CV}[P_{ij}] \leq 1$	reference
$\mathrm{P}\,\vert\,\vert\,\mathbb{E}\big[\sum w_jC_j\big]$	$1+\frac{(m-1)(\Delta+1)}{2m}$	$2-1/m$	[17]
$\mathrm{P}\,\vert\,r_j\,\vert\,\mathbb{E}\big[\sum w_jC_j\big]$	$2+\Delta$	3	[22]
$\mathrm{R}\,\vert\,\vert\,\mathbb{E}\big[\sum w_jC_j\big]$	$1+\frac{\Delta+1}{2}+\varepsilon$	$2+\varepsilon$	this paper
$\mathrm{R}\,\vert\,r_{ij}\,\vert\,\mathbb{E}\big[\sum w_jC_j\big]$	$2+\Delta+\varepsilon$	$3+\varepsilon$	this paper

policies which are based on linear programming relaxations in completion time variables. The performance bounds are constant whenever the coefficients of variation of the jobs' processing times are bounded by a constant. As usual in stochastic scheduling, all bounds hold with respect to any non-anticipatory scheduling policy. By using an idea from Chekuri et al. [2], that approach was extended to stochastic scheduling problems with precedence constrains by Skutella and Uetz [27]. Subsequently, in line with earlier work by Chou et al. [3], Megow et al. [14] combined the stochastic scheduling model with online scheduling, and derived combinatorial, constant competitive algorithms that are not guided by linear programming relaxations. Yet all results, including the analysis in [14], are based on one and the same linear programming relaxation, namely that of [17]. With respect to the underlying relaxation, Schulz [22] goes one step further, and uses the mean busy time relaxation that was previously used also by Correa and Wagner [5], yet its validity in stochastic scheduling still relies on the validity of the completion time relaxation of [17]. Nevertheless, in comparison to [14], the clever use of an optimal solution to an equivalent time-indexed LP relaxation for deterministic scheduling yields improved and simpler results.

Two other research directions are related to our work, yet for different models and independent of the techniques of [17] as well as ours. One is approximation algorithms for preemptive stochastic scheduling by Megow and Vredeveld [15]. They use a single machine relaxation that is optimally solved by a Gittins index policy, and thereby achieve a competitive ratio of 2 for preemptive online stochastic scheduling on parallel identical machines. The other is work by Scharbrodt et al. [20] and Souza and Steger [30], who analyze the expected competitive ratio rather than the expected performance of a policy. In that model, one analyzes the ratio $\mathbb{E}[v(\Pi)/v(\text{Offline-Opt})]$, while we follow [14, 17, 22, 27] and focus on the ratio $\mathbb{E}[v(\Pi)]/\mathbb{E}[v(\Pi^{\mathrm{Opt}})]$ instead.

Note that all results discussed here are restricted to identical parallel machines. Table 1 gives an overview of currently best known performance bounds in nonpreemptive stochastic scheduling with minsum objective, next to the results obtained in this paper.

With respect to algorithmic ideas and techniques, the evolution of stochastic scheduling has largely benefited in the past from progress being made for the corresponding deterministic scheduling problems. For example, all LP-based approximation results for stochastic scheduling on identical parallel machines outlined above build upon a class of linear programming relaxations in completion time variables that dates back to Wolsey [34] and Queyranne [18] (for single machine scheduling), and was later generalized to identical parallel machines by

Schulz [21] and Hall et al. [10] who also presented LP-based approximation algorithms for deterministic scheduling problems.

Our contribution. We obtain the first approximation algorithms for stochastic scheduling on unrelated machines. Despite the fact that the unrelated machine scheduling model is significantly richer than identical machine scheduling, our bounds essentially match all previous performance bounds that have been obtained for the corresponding stochastic scheduling problems on identical parallel machines; see Table 1. We also give a tight lower bound, showing that the dependence of the performance bound on the squared coefficient of variation Δ is unavoidable for the class of policies that we use. For the first time we completely depart from the LP relaxation of Möhring et al. [17], and show how to put a novel, time-indexed linear programming relaxation to work in stochastic machine scheduling. We are optimistic that this novel approach will inspire further research and prove useful for other stochastic optimization problems in scheduling and related areas.

Time-indexed linear programming relaxations have played a pivotal role in the development of constant factor approximation algorithms for deterministic scheduling on unrelated parallel machines [23]. In spite of that, it remained unclear and a major open problem how to come up with a meaningful time-indexed LP relaxation for stochastic scheduling problems [13]. Here the main difficulty is that, in contrast to deterministic schedules that can be fully described by time-indexed 0-1-variables, scheduling *policies* feature a considerably richer structure including complex dependencies between the execution of different jobs which cannot be easily described by time-indexed variables.

In Section 3 we show how to overcome this difficulty and present the first time-indexed LP relaxation for stochastic scheduling on unrelated parallel machines. Here, the value of the time-indexed variable x_{ijt} represents the probability of job j being started on machine i at time t.[3] While writing down the machine capacity constraints[4] is rather easy for deterministic scheduling in this formulation, the situation is somewhat more complicated in the stochastic setting and we require a fair amount of information about the exact probability distributions of random variables P_{ij}.

Notice that, due to the stochastic nature of processing times, even a schedule produced by an optimal policy can be arbitrarily long such that infinitely many variables x_{ijt} may take positive values. Nonetheless, in the full version of the paper we show how to overcome this difficulty. Indeed, we can compute an LP-solution in polynomial time that approximates the optimal LP solution with arbitrary precision.

In Section 4 we discuss how to turn a feasible solution to the time-indexed LP relaxation into a simple scheduling policy. Our approach is inspired by the randomized rounding algorithm for deterministic scheduling on unrelated parallel machines in [23]. Each job j is randomly assigned to a machine i with probability $\sum_t x_{ijt}$; then, on each machine i, the WSEPT policy is used to schedule the jobs assigned to i. The analysis, however, is based on a somewhat more elaborate, random sequencing of jobs which is determined by a two-stage random process.

Since each job is immediately and irrevocably assigned to a machine, our scheduling policies fall into the special class of *fixed assignment policies*. Notice that these policies ignore the additional information that evolves over time in the form of the actual realizations of processing times. Not surprisingly, this ignorance comes at a price. In Section 6 we

[3] Even for simple scheduling policies like the WSEPT rule, determining this probability is non-trivial.
[4] The machine capacity constraints say that each machine can process at most one job at a time.

prove a lower bound of $\Delta/2$ on the performance guarantee of *any* fixed assignment policy. Moreover, we also show that the LP relaxation can have an optimality gap in the same order of magnitude. These negative results nicely complement our positive results; see Table 1.

In order to keep the presentation as simple as possible, we ignore release dates and restrict to the problem $\mathrm{R}\,|\,|\,\mathbb{E}\big[\sum w_jC_j\big]$ throughout most of the paper. Only in Section 5 we show how release dates can be taken care of in our approach.

Parallel to Stochastic Knapsack. There is an interesting parallel of the present work on stochastic scheduling with that on stochastic knapsack problems[5]. The first study of approximation algorithms for stochastic knapsack problems is due to Dean et al. [6], presenting constant factor approximation algorithms along with an analysis of the adaptivity gap[6]. Their results are based on a linear programming relaxation that is essentially the deterministic knapsack LP where item sizes and weights are replaced by expected values. In that sense, methodology-wise their linear program parallels that of [17] in stochastic scheduling on parallel machines. Recently, Gupta et al. [9] were able to obtain constant factor approximation algorithms for a much broader class of stochastic knapsack problems (and other problems, too). Key to these results is a more sophisticated, time-indexed linear programming relaxation, based on the same type of variables as we use here. It is interesting to note that in their paper as well as in ours, moving from "natural yet simple" LP relaxations to richer time-indexed LP relaxations is key to more general results.

2 Notation and preliminaries

We are given a set of jobs J of cardinality n with job weights $w_j \in \mathbb{Z}_{>0}$, $j \in J$, and a set of unrelated parallel machines M of cardinality m. Moreover, for every job $j \in J$ and every machine $i \in M$, we are given a random variable P_{ij}. Each job j needs to be executed on any one of the machines $i \in M$, and each machine can process at most one job at a time. If job j is processed on machine i, its processing time is P_{ij}. However, the actual realization of the processing time is only known upon j's completion and we are thus looking for a non-anticipatory scheduling policy which minimizes the expected total weighted completion time $\mathbb{E}\big[\sum_j w_jC_j\big]$, where C_j denotes the completion time of job j.

Later, in Section 5, we consider a slightly more general model where each job $j \in J$ also comes with a machine dependent release date $r_{ij} \in \mathbb{Z}_{\geq 0}$ before which job j must not be scheduled on machine i. One can think of applications where some job $j \in J$ might not be processed on a certain machine $i \in M$, i. e., $\mathbb{E}[P_{ij}] = \infty$. For the sake of simplicity of presentation, we assume in this paper that $\mathbb{E}[P_{ij}]$ is finite for all $i \in M$ and $j \in J$. But all presented results also hold for the more general case where $\mathbb{E}[P_{ij}] = \infty$ for certain pairs i, j.

Throughout this paper we assume that the random variables P_{ij}, $i \in M$, $j \in J$, take positive integral values only. The following lemma states that this assumption costs at most a factor $1 + \varepsilon$ in the objective function value.

▶ **Lemma 1.** *For any fixed $\varepsilon > 0$, while only loosing a factor $1 + \varepsilon$ in the objective function value, an arbitrary instance can be modified such that the random variables P_{ij}, $i \in M$, $j \in J$, take positive integral values only.*

[5] Note that a stochastic knapsack problem can be reinterpreted as a single machine stochastic scheduling problem where all jobs have due date 1, and with weighted earliness objective.

[6] In stochastic scheduling, this would correspond to the gap between the best static list scheduling policy and an optimal (adaptive) scheduling policy.

Proof. If $\mathbb{E}[P_{ij}] = 0$ and $r_{ij} = 0$ for some pair i, j, then we can ignore job j since it can be scheduled at no further cost on machine i at time 0. We can thus assume from now on that $\mathbb{E}[P_{ij}] > 0$ or $r_{ij} > 0$ for all pairs i, j. By scaling processing times and release dates appropriately, we can make sure that $\mathbb{E}[P_{ij}] \geq \frac{n}{\varepsilon}$ or $r_{ij} \geq \frac{n}{\varepsilon}$ for each pair i, j. As a result of this scaling step we know that, for any scheduling policy, $\mathbb{E}[C_j] \geq n/\varepsilon$ for each job $j \in J$. Rounding up all processing times to the nearest positive integer therefore increases the (expected) completion time of any job j by at most $n \leq \varepsilon\,\mathbb{E}[C_j]$. The overall increase in the objective function is thus bounded by a factor $1 + \varepsilon$. ◀

Given that all processing times are integral, we can obviously assume with no further loss of generality that jobs can only be started at integral points in time $t \in \mathbb{Z}_{\geq 0}$. In order to write down an LP relaxation in time-indexed variables, we require a fair amount of information about the exact probability distributions of random variables P_{ij}. More precisely, besides the expectations $\mathbb{E}[P_{ij}]$, we also need the values

$$q_{ijr} := \Pr[P_{ij} \geq r + 1] \qquad \text{for } i \in M,\ j \in J, \text{ and } r \in \mathbb{Z}_{\geq 0}.$$

This, of course, raises questions about the input size of the problem. Here we make the following assumption. In the input we are given, for each job $j \in J$ and each machine $i \in M$, the expected processing time $\mathbb{E}[P_{ij}]$. Moreover, we have access to an oracle which, for any triple i, j, r returns q_{ijr}. We emphasize that, in order for our approach to work, it suffices to get these values within some finite precision at the expense of an additional factor $1 + \varepsilon$ in the performance guarantee of our algorithms. More precisely, it suffices to get the values q_{ijr} rounded to multiples of ε/n, which, in particular, can be encoded polynomially in the input size. Notice that such an oracle can be simulated by a polynomial-time Monte Carlo algorithm that can sample from the distribution of the random variables P_{ij}. Having said that, in order to keep the presentation simple we neglect these aspects throughout the paper and assume that we have access to the exact values q_{ijr}.

In the analysis of our algorithm we need the following standard property of the moments of random variable P_{ij}.

▶ **Lemma 2.** *Let $j \in J$ and $i \in M$. Then,*

$$\sum_{r \in \mathbb{Z}_{\geq 0}} q_{ijr} = \mathbb{E}[P_{ij}] \qquad \text{and} \qquad \sum_{r \in \mathbb{Z}_{\geq 0}} (r + \tfrac{1}{2})\, q_{ijr} = \frac{1 + \mathbb{CV}[P_{ij}]^2}{2}\, \mathbb{E}[P_{ij}]^2 \ ,$$

where $\mathbb{CV}[P_{ij}]^2 := \big(\mathbb{E}[P_{ij}^2] - E[P_{ij}]^2\big)/\mathbb{E}[P_{ij}]^2$ is the squared coefficient of variation of P_{ij}.

The proof of the lemma is based on standard results for the nth moment of a random variable, see, e. g. [7, V.6, Lemma 1].

3 Time-indexed LP relaxation

In the following we derive an LP relaxation of the stochastic scheduling problem under consideration. For a given non-anticipatory scheduling policy Π, let x_{ijt} be the probability that Π starts job $j \in J$ on machine $i \in M$ at time $t \in \mathbb{Z}_{\geq 0}$. Notice that this random decision may depend on the actual processing times of other jobs started by Π before time t. On the other hand, due to the non-anticipatory nature of policy Π, the random variable P_{ij} is independent of Π's random decision to start job j on machine i at time t.

As the x_{ijt}'s are going to be the variables of our LP relaxation, we derive crucial properties that are going to be the constraints of the LP relaxation. If job $j \in J$ is started on

machine $i \in M$ at time $t \in \mathbb{Z}_{\geq 0}$, due to the non-anticipative nature of policy Π, j's expected completion time is $t + \mathbb{E}[P_{ij}]$. Thus, by linearity of expectation, the expected completion time of j is

$$\mathbb{E}[C_j] = \sum_{i \in M} \sum_{t \in \mathbb{Z}_{\geq 0}} x_{ijt} \left(t + \mathbb{E}[P_{ij}]\right) \ .$$

A more careful look at j's behavior reveals the following property. Conditioning on j being started on machine i at time t, the probability that j is still occupying machine i within the later time interval $[s, s+1]$, $s \in \mathbb{Z}_{\geq t}$, is equal to $q_{ij\,s-t}$ by definition. Unconditioning yields

$$\Pr\big[i \text{ processes } j \text{ in } [s, s+1]\big] = \sum_{t=0}^{s} x_{ijt}\, q_{ij\,s-t} \ . \tag{1}$$

As machine i can process at most one job at a time, also the *expected* number of jobs being processed by i in $[s, s+1]$ is bounded by 1. That is, by linearity of expectation,

$$\sum_{j \in J} \sum_{t=0}^{s} x_{ijt}\, q_{ij\,s-t} \leq 1 \ .$$

Finally, since policy Π has to process all jobs, we get for every job j $\sum_{i \in M} \sum_{t \in \mathbb{Z}_{\geq 0}} x_{ijt} = 1$. Thus, the probabilities x_{ijt} corresponding to policy Π form a feasible solution to the following LP relaxation, and the value of this LP solution x is equal to the expected value of the schedule produced by policy Π:

$$\begin{aligned}
\min \quad & \sum_{j \in J} w_j\, C_j^{\mathrm{LP}} & & \\
\text{s.t.} \quad & \sum_{i \in M} \sum_{t \in \mathbb{Z}_{\geq 0}} x_{ijt} = 1 & \text{for all } j \in J, & \quad (2)\\
& \sum_{j \in J} \sum_{t=0}^{s} x_{ijt}\, q_{ij\,s-t} \leq 1 & \text{for all } i \in M,\ s \in \mathbb{Z}_{\geq 0}, & \quad (3)\\
& C_j^{\mathrm{LP}} = \sum_{i \in M} \sum_{t \in \mathbb{Z}_{\geq 0}} x_{ijt} \left(t + \mathbb{E}[P_{ij}]\right) & \text{for all } j \in J, & \quad (4)\\
& x_{ijt} \geq 0 & \text{for all } j \in J,\ i \in M,\ t \in \mathbb{Z}_{\geq 0}. &
\end{aligned}$$

Notice that the LP variables C_j^{LP} are uniquely determined by the x-variables and could as well be omitted by replacing them in the objective function with the right hand side of (4).

Also notice that this linear program suffers from infinitely many variables and constraints. We claim that this can be dealt with at the expense of an additional factor $1 + \varepsilon$ in the performance guarantee of our algorithms. For a detailed discussion and formal proof, see the full version of the paper.

▶ **Theorem 3.** *The above infinite time-indexed LP relaxation can be solved in pseudo-polynomial time in the input size. Moreover, a $(1+\varepsilon)$-approximate LP solution can be found in time polynomial in the input size and $1/\varepsilon$.*

4 Turning an LP solution into a scheduling policy

For a feasible LP solution x, let $X_{ij} := \sum_{t \in \mathbb{Z}_{\geq 0}} x_{ijt}$ for $i \in M$, $j \in J$. LP constraints (2) imply that $\sum_{i \in M} X_{ij} = 1$ for every job $j \in J$.

Given the values X_{ij} corresponding to a feasible LP solution x, our scheduling policy Assign(X) assigns each job $j \in J$ independently at random to one machine $i \in M$ with probability X_{ij}. Then, on each machine $i \in M$, it sequences jobs assigned to i according to the WSEPT rule.

▶ **Theorem 4.** *The expected value of the schedule constructed by policy* Assign(X) *is at most* $\frac{3}{2} + \frac{\Delta}{2}$ *times the value of the underlying LP solution* x*.*

Notice that Theorem 4 and Theorem 3 imply the existence of a polynomial-time algorithm that, for any given instance of our stochastic scheduling problem and for any $\varepsilon > 0$, finds a scheduling policy with performance guarantee $\frac{3}{2} + \frac{\Delta}{2} + \varepsilon$. Remember that Δ upper bounds the squared coefficient of variation $\mathbb{CV}[P_{ij}]^2$ for all P_{ij}. It is not difficult to see that, instead of the random assignment of jobs to machines, we can use a deterministic assignment obtained via the method of conditional probabilities and still get the same performance guarantee.

The proof of Theorem 4 is based on a refined, somewhat more complicated policy, that takes the entire LP solution x into account and yields a worse schedule in expectation. It is therefore sufficient to prove the bound stated in Theorem 4 for this alternative policy which we refer to as Assign&Sequence(x).

Assign&Sequence(x)

1. For every job $j \in J$, choose a pair (i, t) independently at random with probability x_{ijt} and some $r \in \mathbb{Z}_{\geq 0}$ independently at random with probability $q_{ijr}/\mathbb{E}[P_{ij}]$; assign job j to machine i and set its *tentative start time* s to $s := t + r$ (we write "$j \to i, s$" for short).
2. On each machine $i \in M$, sequence all jobs assigned to i in order of increasing tentative start times; ties are broken randomly.

Notice that, as in the simpler policy Assign(X), job j is assigned to machine i with probability $\sum_{t \in \mathbb{Z}_{\geq 0}} x_{ijt} = X_{ij}$. Since Assign($X$) sequences the jobs on every machine in an optimal way, it is superior to policy Assign&Sequence(x). By construction of policy Assign&Sequence(x), the probability of assigning job $j \in J$ to machine $i \in M$ and setting its tentative start time to $s \in \mathbb{Z}_{\geq 0}$ is

$$\Pr[j \to i, s] = \sum_{t=0}^{s} x_{ijt} \, \frac{q_{ij\,s-t}}{\mathbb{E}[P_{ij}]} \quad . \tag{5}$$

We prove the following job-by-job performance guarantee for Assign&Sequence(x).

▶ **Theorem 5.** *For every job* $j \in J$*, the expected value of* j*'s completion time in the schedule constructed by policy* Assign&Sequence(x) *is at most* $(\frac{3}{2} + \frac{\Delta_j}{2})\, C_j^{\mathrm{LP}}$ *where* $\Delta_j := \max_{i \in M} \mathbb{CV}[P_{ij}]^2$*.*

By linearity of expectation, Theorem 5 immediately implies Theorem 4. In the proof of Theorem 5 we make use of the following lemma.

▶ **Lemma 6.** *Let* $j \in J$*,* $i \in M$*, and* $s \in \mathbb{Z}_{\geq 0}$*. If* $j \to i, s$*, then the expected total processing time of jobs that policy* Assign&Sequence(x) *schedules on machine* i *before job* j *is at most* $s + \frac{1}{2}$*.*

Proof. We first bound the expected total processing time of jobs $k \neq j$ with $k \to i, s'$ for some fixed $s' \in \mathbb{Z}_{\geq 0}$:

$$\sum_{k \neq j} \mathbb{E}[P_{ik}] \, \Pr[k \to i, s'] \overset{(5)}{=} \sum_{k \neq j} \sum_{t'=0}^{s'} x_{ikt'} \, q_{ik\,s'-t'} \leq 1 \qquad \text{by (3).}$$

Thus, the expected[7] total processing times of jobs processed before job j on machine i is at most

$$\sum_{k\neq j}\mathbb{E}[P_{ik}]\left(\sum_{s'=0}^{s-1}\Pr[k\to i,s']+\tfrac{1}{2}\Pr[k\to i,s]\right)\leq s+\tfrac{1}{2}\ .$$

This concludes the proof. ◀

Proof of Theorem 5. By Lemma 6 we get

$$\mathbb{E}[C_j\mid j\to i,s]\leq s+\tfrac{1}{2}+\mathbb{E}[P_{ij}] \tag{6}$$

for every job $j\in J$, machine $i\in M$, and tentative start time $s\in\mathbb{Z}_{\geq 0}$. Unconditioning the expectation yields

$$\mathbb{E}[C_j]=\sum_{i\in M}\sum_{s\in\mathbb{Z}_{\geq 0}}\mathbb{E}\big[C_j\mid j\to i,s\big]\Pr[j\to i,s]\ .$$

Applying inequality (6) and equation (5) we get

$$\mathbb{E}[C_j]\leq\sum_{i=1}^{m}\sum_{s\in\mathbb{Z}_{\geq 0}}\big(s+\tfrac{1}{2}+\mathbb{E}[P_{ij}]\big)\sum_{t=0}^{s}x_{ijt}\,\frac{q_{ij\,s-t}}{\mathbb{E}[P_{ij}]}\ .$$

Exchanging the order of summation of s and t, and setting $r:=s-t$ yields

$$\begin{aligned}\mathbb{E}[C_j]&\leq\sum_{i=1}^{m}\sum_{t\in\mathbb{Z}_{\geq 0}}x_{ijt}\left(t+\mathbb{E}[P_{ij}]+\sum_{r\in\mathbb{Z}_{\geq 0}}(r+\tfrac{1}{2})\,\frac{q_{ijr}}{\mathbb{E}[P_{ij}]}\right)\\&=\sum_{i=1}^{m}\sum_{t\in\mathbb{Z}_{\geq 0}}x_{ijt}\left(t+\left(\frac{3}{2}+\frac{\mathbb{CV}[P_{ij}]^2}{2}\right)\mathbb{E}[P_{ij}]\right)\\&\leq\left(\frac{3}{2}+\frac{\Delta_j}{2}\right)C_j^{\mathrm{LP}}\end{aligned}$$

by Lemma 2 and (4). This concludes the proof. ◀

We note that the same results can in fact be obtained by considering a weaker LP relaxation in variables y_{ijs}, corresponding to the probability that job j is being processed on machine i in time interval $[s,s+1]$.

5 Adding release dates

In this section we show how to adapt our analysis for a more general problem where each job $j\in J$ comes with a machine dependent deterministic release date $r_{ij}\in\mathbb{Z}_{\geq 0}$ before which job j must not be scheduled on machine i. To handle release dates we add one additional family of constraints to our time-indexed LP relaxation:

$$x_{ijt}=0 \qquad \text{for all } i\in M,\ j\in J,\ t<r_{ij}.$$

These constraints are obviously fulfilled by the probabilities x_{ijt} corresponding to an arbitrary scheduling policy Π as no job may be started before it is released. We consider the same LP based policy Assign&Sequence(x) for this more general problem.

[7] Notice that the expectation is taken with respect to both the random decisions of our policy Assign&Sequence(x) as well as the random processing times of jobs $k\neq j$.

▶ **Theorem 7.** *In the presence of release dates, for every job $j \in J$, the expected value of j's completion time in the schedule constructed by policy* Assign&Sequence(x) *is at most* $(2 + \Delta_j)\, C_j^{\mathrm{LP}}$ *where* $\Delta_j := \max_{i \in M} \mathbb{CV}[P_{ij}]^2$.

The proof of Theorem 7 is almost identical to the proof of Theorem 5, and contained in the full version of the paper. We conclude this section with the following result.

▶ **Corollary 8.** *In the presence of release dates, the expected value of the schedule constructed by policy* Assign&Sequence(x) *is at most* $2 + \Delta$ *times the value of the underlying LP solution x. Thus, for any given instance of the stochastic scheduling problem and for any $\varepsilon > 0$, a $(2 + \Delta + \varepsilon)$-approximate scheduling policy can be found in polynomial time.*

6 Tightness of Performance Bounds

In this section, we argue that our results cannot be easily improved, because both LP relaxation as well as our scheduling policies have an optimality gap of $\Theta(\Delta)$.

▶ **Theorem 9.** *Even for the special case of a single machine, the multiplicative gap between the expected value of an optimal policy and the value of an optimal LP solution can be as large as $\Delta/2$.*

This is remarkable and somewhat surprising since the corresponding time-indexed linear program for the deterministic single machine scheduling problem has the same optimal value as an optimal schedule.

▶ **Theorem 10.** *Even for the special case of identical parallel machines, the performance ratio of any fixed-assignment policy can be as large as $\frac{(1-\delta)\Delta}{2}$ for any $\delta > 0$, for large enough number of machines m.*

For the proof of these two theorems we refer to the full version of the paper.

7 Execution of Scheduling Policies

We have argued that the policy we propose can be computed in polynomial time, but so far did not discuss the computation time to actually execute the scheduling policy, or more generally, any stochastic scheduling policy. The major issue is how, and with which computational effort the scheduler learns about the next job completion when executing a set of jobs. Probabilistically, this event is described by the minimum of a set of random variables, of which we just sample while executing the policy. In general, and already if there is just one single job to be processed, there might of course be nonzero probability for a job to be exponentially longer than expected. But due to Markov's inequality, the probability for exceeding the expected processing time by an exponential factor is exponentially small, too. Therefore, with high probability the sampled processing times of jobs can be encoded polynomially in the input size of the problem. Apart from this minor issue inherent in all stochastic scheduling problems, we note that the policy Assign(X) is in particular elementary [16], meaning that jobs are only started upon release times or completion times of other jobs. Hence, there is only a linear number of decision times.

8 Concluding remarks

One of the main technical contributions of this paper is to introduce the important concept of time-indexed linear programming relaxations to the area of stochastic scheduling, which yields,

for the first time, performance bounds for stochastic unrelated machine scheduling that even match currently best known results for deterministic unrelated machine scheduling. Obtaining performance bounds independent of the coefficient of variation of the underlying processing times remains an interesting challenge, even for the special case of parallel machines.

Acknowledgements. The authors would like to thank Nicole Megow for helpful discussions on the topic of this paper, and Andreas S. Schulz and Benjamin Labonté for valuable comments on an earlier version of this paper. Most of the results presented in this paper were obtained during the Dagstuhl Seminar 13111 "Scheduling". The authors would like to thank the organizers and Schloss Dagstuhl for providing an enjoyable and stimulating atmosphere.

References

1 F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–43, New York City, NY, 1999.

2 C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31:146–166, 2001.

3 C.F. Chou, H. Liu, M. Queyranne, and D. Simchi-Levi. On the asymptotic optimality of a simple on-line algorithm for the stochastic single machine weighted completion time problem and its extensions. *Operations Research*, 54:464– 474, 2006.

4 F. A. Chudak. A min-sum 3/2-approximation algorithm for scheduling unrelated parallel machines. *Journal of Scheduling*, 2:73–77, 1999.

5 J. Correa and M. Wagner. LP-based online scheduling: From single to parallel machines. *Mathematical Programming*, 119:109–136, 2008.

6 B. Dean, M. X. Goemans, and J. Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Mathematics of Operations Research*, 33:945–964, 2008.

7 W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. Wiley Series in Probability and Mathematical Statistics, 2nd edition, 1971.

8 R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

9 A. Gupta, R. Krishnaswamy, M. Molinaro, and R. Ravi. Approximation algorithms for correlated knapsacks and non-martingale bandits. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, pages 827–836. IEEE Computer Society, 2011.

10 L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.

11 H. Hoogeveen, P. Schuurman, and G. J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. *INFORMS Journal on Computing*, 13:157–168, 2001.

12 J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

13 N. Megow. Approximation in stochastic scheduling. Invited talk at the Dagstuhl Seminar 13111 "Scheduling", Schloss Dagstuhl, Wadern, 2013.

14 N. Megow, M. Uetz, and T. Vredeveld. Models and algorithms for stochastic online scheduling. *Mathematics of Operations Research*, 31(3):513–525, 2006.

15 N. Megow and T. Vredeveld. Approximation in preemptive stochastic online scheduling. In Y. Azar and T. Erlebach, editors, *Algorithms - ESA 2006*, volume 4168 of *Lecture Notes in Computer Science*, pages 516–527. Springer-Verlag, 2006.

16 R. H. Möhring, F. J. Radermacher, and G. Weiss. Stochastic scheduling problems I: General strategies. *ZOR - Zeitschrift für Operations Research*, 28:193–260, 1984.

17 R. H. Möhring, A. S. Schulz, and M. Uetz. Approximation in stochastic scheduling: The power of LP-based priority policies. *Journal of the ACM*, 46:924–942, 1999.

18 M. Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58:263–285, 1993.

19 M. H. Rothkopf. Scheduling with random service times. *Management Science*, 12:703–713, 1966.

20 M. Scharbrodt, T. Schickinger, and A. Steger. A new average case analysis for completion time scheduling. *Journal of the ACM*, 53:121–146, 2006.

21 A. S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 301–315. Springer, 1996.

22 A. S. Schulz. Stochastic online scheduling revisited. In B. Yang, D.-Z. Du, and C. Wang, editors, *Combinatorial Optimization and Applications*, volume 5165 of *Lecture Notes in Computer Science*, pages 448–457. Springer, 2008.

23 A. S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15:450–469, 2002.

24 P. Schuurman and G. J. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2:203–213, 1999.

25 J. Sethuraman and M. S. Squillante. Optimal scheduling of multiclass parallel machines. In *Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 963–964, 1999.

26 M. Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of the ACM*, 48:206–242, 2001.

27 M. Skutella and M. Uetz. Stochastic machine scheduling with precedence constraints. *SIAM Journal on Computing*, 34:788–802, 2005.

28 M. Skutella and G. J. Woeginger. A PTAS for minimizing the total weighted completion time on identical parallel machines. *Mathematics of Operations Research*, 25:63–75, 2000.

29 W. E. Smith. Various optimizers for single-stage production. *Naval Research and Logistics Quarterly*, 3:59–66, 1956.

30 A. Souza and A. Steger. The expected competitive ratio for weighted completion time scheduling. *Theory of Computing Systems*, 39:121–136, 2006.

31 M. Uetz. When greediness fails: Examples from stochastic scheduling. *Operations Research Letters*, 31:413–419, 2003.

32 Gideon Weiss. Approximation results in parallel machines stochastic scheduling. *Annals of Operations Research*, 26:195–242, 1990.

33 Gideon Weiss. Turnpike optimality of Smith's rule in parallel machines stochastic scheduling. *Mathematics of Operations Research*, 17:255–270, 1992.

34 L. A. Wolsey. Mixed integer programming formulations for production planning and scheduling problems. Invited talk at the 12th International Symposium on Mathematical Programming, MIT, Cambridge, 1985.

Computational Complexity of the Extended Minimum Cost Homomorphism Problem on Three-Element Domains

Hannes Uppman*

Department of Computer and Information Science, Linköping University, Linköping, Sweden
hannes.uppman@liu.se

Abstract

In this paper we study the computational complexity of the extended *minimum cost homomorphism problem* (Min-Cost-Hom) as a function of a constraint language, i.e. a set of constraint relations and cost functions that are allowed to appear in instances. A wide range of natural combinatorial optimisation problems can be expressed as extended Min-Cost-Homs and a classification of their complexity would be highly desirable, both from a direct, applied point of view as well as from a theoretical perspective.

The extended Min-Cost-Hom can be understood either as a flexible optimisation version of the *constraint satisfaction problem* (CSP) or a restriction of the (general-valued) *valued constraint satisfaction problem* (VCSP). Other optimisation versions of CSPs such as the *minimum solution problem* (Min-Sol) and the *minimum ones problem* (Min-Ones) are special cases of the extended Min-Cost-Hom.

The study of VCSPs has recently seen remarkable progress. A complete classification for the complexity of finite-valued languages on arbitrary finite domains has been obtained Thapper and Živný [STOC'13]. However, understanding the complexity of languages that are not finite-valued appears to be more difficult. The extended Min-Cost-Hom allows us to study problematic languages of this type without having to deal with with the full generality of the VCSP. A recent classification for the complexity of three-element Min-Sol, Uppman [ICALP'13], takes a step in this direction. In this paper we generalise this result considerably by determining the complexity of three-element extended Min-Cost-Hom.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, F.4.1 Mathematical Logic

Keywords and phrases Complexity, Optimisation, Constraint Satisfaction

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.651

1 Introduction

The *constraint satisfaction problem* (CSP) is a decision problem where an instance consists of a set of variables, a set of values, and a collection of constraints expressed over the variables. The objective is to determine if it is possible to assign values to the variables in such a way that all constrains are satisfied simultaneously. In general the constraint satisfaction problem is NP-complete. However, by only allowing constraint-relations from a fixed constraint language Γ one can obtain tractable fragments. A famous conjecture by Feder and Vardi [7]

* Partially supported by the National Graduate School in Computer Science (CUGS), Sweden.

© Hannes Uppman;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 651–662
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

predicts that this restricted problem, denoted CSP(Γ), is either (depending on Γ) in P or is NP-complete.

In this paper we will study an optimisation version of the CSP. Several such variants have been investigated in the literature. Examples are: the *min ones problem* (Min-Ones) [17], the *minimum solution problem* (Min-Sol) [14] and the *valued constraint satisfaction problem* (VCSP) [18]. The problem we will work with is called the extended *minimum cost homomorphism problem* (Min-Cost-Hom). The "unextended" version of this problem was, motivated by a problem in defence logistics, introduced in [9] and studied in a series of papers before its complexity was completely characterised in [20]. The extended version of the problem was introduced in [21] and differs from the original version in that it is parametrised not only by a set of allowed constraint relations, but also by a set of allowed cost functions (a formal definition is given in Section 2).

The extended Min-Cost-Hom provide a more general framework than both Min-Ones and Min-Sol; a problem of one of the latter types is also an extended Min-Cost-Hom. The VCSP-framework on the other hand is more general than the extended Min-Cost-Hom. In fact, we can describe every extended Min-Cost-Hom as a VCSP for a constraint language in which every cost function is either $\{0, \infty\}$-valued or unary. The extended Min-Cost-Hom captures, despite this restriction, a wealth of combinatorial optimisation problems arising in a broad range of fields.

The study of VCSPs has recently seen remarkable progress; Thapper and Živný [22] described when a certain linear programming relaxation solves instances of the problem, Kolmogorov [15] simplified this description for finite-valued languages, Huber, Krokhin and Powell [10] classified all finite-valued languages on three-element domains, and Thapper and Živný [23] found a complete classification of the complexity for finite-valued languages on arbitrary finite domains.

Most of the classifications that have been obtained concerns finite-valued constraint languages ([22] mentioned above being a notable exception). Understanding the complexity of general languages appears to be more difficult. Extended Min-Cost-Homs allows us to study languages of this type without having to deal with with the full generality of the VCSP. Using techniques of the so called algebraic approach (see e.g. [2, 3, 11]), and building on results by Takhanov [20, 21] and Thapper and Živný [22, 23] we could in [24] take a step in this direction by proving a classification for the complexity of Min-Sol on the three-element domain. In this paper we generalise these results to the extended Min-Cost-Hom. Namely, we prove the following theorem.

▶ **Theorem 1.** *Let* (Γ, Δ) *be a finite language on a three-element domain* D *and define* $\Gamma^+ = \Gamma \cup \{\{d\} : d \in D\} \cup \{\{x : \nu(x) < \infty\} : \nu \in \Delta\}$. *If* (Γ, Δ) *is a core, then one of the following is true.*

- (Γ^+, Δ) *is of semilattice type (Definition 5) and Min-Cost-Hom*(Γ^+, Δ) *is in PO.*
- (Γ^+, Δ) *is of tournament pair type (Definition 14) and Min-Cost-Hom*(Γ^+, Δ) *is in PO.*
- *Min-Cost-Hom*(Γ, Δ) *is NP-hard.*

If (Γ, Δ) is of semilattice type, then Min-Cost-Hom(Γ, Δ) can be solved efficiently by linear programming [22]. If (Γ, Δ) is of tournament pair type we show how to reduce the problem to one demonstrated to be tractable in [20, 21]. Also in this case the underlying algorithmic technique is linear programming.

We define cores in Section 5. Theorem 1 combined with the following result, which follows from [23, Lemma 2.4], yields a full classification for the extended Min-Cost-Hom on three-element domains.

▶ **Proposition 2.** *If* (Γ', Δ') *is a core of* (Γ, Δ) *then Min-Cost-Hom*(Γ, Δ) *and Min-Cost-Hom*(Γ', Δ') *are polynomial-time inter-reducible.*

To obtain the classification we apply tools from the algebraic approach, and, following Thapper and Živný, we make repeated use of Motzkin's Theorem. Our tractability results are formulated and proved for arbitrary finite domains and are therefore not restricted to the three-element case. Many of the tools we derive to aid in proving our main theorem are also effective on domains of size larger than three. One example is that we show that a relation fails to be in the wpp-closure of a language only if some fractional polymorphism of the language does not preserve the relation (Proposition 20). This complements results in [3]. Another example is that we show that all constants can be added to a core language without significantly changing the complexity of the associated extended Min-Cost-Hom (Proposition 34). This complements results in [23].

The rest of the paper is organised as follows. In Section 2 we define some fundamental concepts, in Section 3 we state and prove tractability results, in Section 4 we collect a number of results that will be used later on (these might also be useful on domain of larger size), in Section 5 we define cores [23] and prove a related result, in Section 6 we focus on the three-element domain and establish our main result; that core languages that are not tractable by the results in Section 3 are in fact NP-hard.

A longer version of this paper, containing complete proofs, is available at `http://arxiv.org/abs/1308.1394`.

2 Preliminaries

Let D be a finite set. The pair (Γ, Δ) is called a finite *language* if Γ is a finite set of finitary relations on D and Δ is a finite set of functions $D \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$. For every finite language (Γ, Δ) we define the optimisation problem Min-Cost-Hom(Γ, Δ) as follows.

Instance: A triple (V, C, w) where

- V is a set of variables,
- C is a set of Γ-allowed constraints, i.e. a set of pairs (s, R) where the constraint-scope s is a tuple of variables, and the constraint-relation R is a member of Γ of the same arity as s,
- w is a weight function $V \times \Delta \to \mathbb{Q}_{\geq 0}$.

Solution: A function $\varphi : V \to D$ s.t. for every $(s, R) \in C$ it holds that $\varphi(s) \in R$, where φ is applied component-wise.

Measure: The measure of a solution φ is $m(\varphi) = \sum_{v \in V} \sum_{\nu \in \Delta} w(v, \nu)\nu(\varphi(v))$. For every function $\varphi : V \to D$ that is not a solution we define $m(\varphi) = \infty$.

The objective is to find a solution φ that minimises $m(\varphi)$.

For an instance I we let $\mathrm{Sol}(I)$ denote the set of all solutions, $\mathrm{Optsol}(I)$ the set of all optimal solutions and $\mathrm{Opt}(I)$ the measure of an optimal solution. If I is unsatisfiable we set $\mathrm{Opt}(I) = \infty$. We define $0 \infty = \infty 0 = 0$, $x \leq \infty$ and $x + \infty = \infty + x = \infty$ for all $x \in \mathbb{Q}_{\geq 0} \cup \{\infty\}$.

2.1 Names and Notation

A k-ary *operation* on D is a function $D^k \to D$ and a (unary) *cost function* on D is a function $D \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$. The set of all operations on D is denoted $\mathcal{O}_D$. The ith projection operation will be denoted pr_i and the arity of a relation R is denoted $\mathrm{ar}(R)$. We define $\binom{A}{2} = \{\{x, y\} \subseteq A : x \neq y\}$. For functions $f_1, \ldots, f_k : A \to B$ and $g : B^k \to C$ we denote

by $g[f_1, \dots, f_k]$ the function $x \mapsto g(f_1(x), \dots, f_k(x))$ from A to C. For a binary operation f we define $\overline{f}$ through $\overline{f}(x, y) = f(y, x)$. A k-ary operation f on D is called *conservative* if $f(x_1, \dots, x_k) \in \{x_1, \dots, x_k\}$ for every $x_1, \dots, x_k \in D$. A ternary operation m on D is called *arithmetical* if $m(x, y, y) = m(x, y, x) = m(y, y, x) = x$ for every $x, y \in D$. We say that an operation f on D is conservative (arithmetical) on $S \subseteq D$ if $f|_S$ is conservative (arithmetical). Similarly we say that f is conservative (arithmetical) on $\mathcal{S} \subseteq 2^D$ if $f|_S$ is conservative (arithmetical) for every $S \in \mathcal{S}$.

For a set A of operations (relations) we write $A^{(k)}$ for the set of all k-ary operations (relations) in A. For a set Γ of relations on D we use Γ^c to denote $\Gamma \cup \{\{d\} : d \in D\}$.

We use δ for the Kronecker delta function, i.e. $\delta_{x,y} = 1$ if $x = y$ and $\delta_{x,y} = 0$ otherwise. A *semilattice operation* is a binary operation that is idempotent, commutative and associative.

2.2 Polymorphisms

A function $f : D^m \to D$ is called a *polymorphism* of Γ if for every $R \in \Gamma$ and every $t_1, \dots, t_m \in R$ it holds that $f(t_1, \dots, t_m) \in R$, where f is applied component-wise. The set of all polymorphisms of Γ is denoted $\mathrm{Pol}(\Gamma)$. A function $\omega : \mathrm{Pol}^{(k)}(\Gamma) \to \mathbb{Q}_{\geq 0}$ is a k-ary *fractional polymorphism* [4] of (Γ, Δ) iff $\sum_{g \in \mathrm{Pol}^{(k)}(\Gamma)} \omega(g) = 1$ and

$$\sum_{g \in \mathrm{Pol}^{(k)}(\Gamma)} \omega(g)\nu(g(x_1, \dots, x_k)) \leq \frac{1}{k} \sum_{i=1}^{k} \nu(x_i) \text{ for every } \nu \in \Delta, x_1, \dots, x_k \in D.$$

The support of a fractional polymorphism ω, denoted $\mathrm{supp}(\omega)$, is the set of polymorphisms for which ω is non-zero. The set of all fractional polymorphisms of (Γ, Δ) is denoted $\mathrm{fPol}(\Gamma, \Delta)$.

▶ **Example 3.** The function pr_i is a trivial polymorphism for any set of relations Γ, and the function $f \mapsto \sum_{i=1}^{k} \frac{1}{k} \delta_{\mathrm{pr}_i, f}$ is a k-ary fractional polymorphism of every language (Γ, Δ).

2.3 Reductions

A relation R is called *pp-definable* in Γ iff there is an instance $I = (V, C)$ of $\mathrm{CSP}(\Gamma)$ s.t. $R = \{(\varphi(v_1), \dots, \varphi(v_n)) : \varphi \in \mathrm{Sol}(I)\}$ for some $v_1, \dots, v_n \in V$. The notation $\langle \Gamma \rangle$ is used for the set of all relations that are pp-definable in Γ. Similarly; R is called *weighted pp-definable* (wpp-definable) in (Γ, Δ) iff there is an instance $I = (V, C, w)$ of Min-Cost-Hom(Γ, Δ) s.t. $R = \{(\varphi(v_1), \dots, \varphi(v_n)) : \varphi \in \mathrm{Optsol}(I)\}$ for some $v_1, \dots, v_n \in V$. We use $\langle \Gamma, \Delta \rangle_w$ to denote the set of all such relations. A function $\nu : D \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$ is called *expressible* in (Γ, Δ) iff there is an instance $I = (V, C, w)$ of Min-Cost-Hom(Γ, Δ) and $v \in V$ s.t. $\nu(x) = \min\{m(\varphi) : \varphi : V \to D, \varphi(v) = x\}$. The set of all cost functions expressible in (Γ, Δ) is denoted $\langle \Gamma, \Delta \rangle_e$. We use $\mathrm{Feas}(\Delta)$ for the set $\{\{x : \nu(x) < \infty\} : \nu \in \Delta\}$.

What makes these closures interesting is the following result, see e.g. [4, 5, 13].

▶ **Theorem 4.** *Let $\Gamma' \subseteq \langle \Gamma, \Delta \rangle_w$ and $\Delta' \subseteq \langle \Gamma, \Delta \rangle_e$ be finite sets. Then, it holds that Min-Cost-Hom$(\Gamma' \cup \mathrm{Feas}(\Delta'), \Delta')$ is polynomial-time reducible to Min-Cost-Hom(Γ, Δ).*

3 Tractable languages

We will make use of two tractability results. The first follows from a theorem by Thapper and Živný [22, Theorem 4.1 (see remarks in Section 5)].

▶ **Definition 5.** We say that a finite language (Γ, Δ) is of *semilattice type* if there exists $\omega \in \mathrm{fPol}^{(2)}(\Gamma, \Delta)$ with $f \in \mathrm{supp}(\omega)$ s.t. f is a semilattice operation.

▶ **Theorem 6.** *If* (Γ, Δ) *is a finite language of semilattice type, then Min-Cost-Hom*(Γ, Δ) *is in PO.*

▶ **Example 7.** Let (Γ, Δ) be a language on a totally ordered domain D that admits the binary fractional polymorphism $f \mapsto \frac{1}{2}\delta_{\min,f} + \frac{1}{2}\delta_{\max,f}$. Certainly min is a semilattice operation, so by Theorem 6 it follows that Min-Cost-Hom(Γ, Δ) is in PO.

We remark that the theorem in [22] from which Theorem 6 follows is very capable; it explains the tractability of every finite-valued VCSP that is not NP-hard [23].

The second tractability result generalises a family of languages that Takhanov has proved tractable [20, 21]. The particular formulation we will use here is a bit more general than a version we previously used in [24, Theorem 8].

To state the result we need to introduce a few concepts. A central observation is given by the following lemma. The result follows immediately from the definition of fractional polymorphisms and the measure function m. We omit the proof.

▶ **Lemma 8.** *If* (Γ, Δ) *admits a* k*-ary fractional polymorphism* ω *and* I *is an instance of Min-Cost-Hom*(Γ, Δ) *with* $\varphi_1, \dots, \varphi_k \in \mathrm{Sol}(I)$*, then* $f[\varphi_1, \dots, \varphi_k] \in \mathrm{Sol}(I)$ *for every* $f \in \mathrm{supp}(\omega)$ *and*

$$\sum_{f \in \mathrm{Pol}^{(k)}(\Gamma)} \omega(f) m(f[\varphi_1, \dots, \varphi_k]) \leq \frac{1}{k} \sum_{i=1}^{k} m(\varphi_k).$$

▶ **Example 9.** Consider again Example 7. It follows from Lemma 8 that, for any instance $I = (V, C, w)$ and any $\varphi_1, \varphi_2 : V \to D$, we have $m(\min[\varphi_1, \varphi_2]) + m(\max[\varphi_1, \varphi_2]) \leq m(\varphi_1) + m(\varphi_2)$. Functions of this kind are called *submodular* and are central characters in the field of discrete optimisation, see e.g. [8].

The following definition establishes some convenient notation.

▶ **Definition 10.** For functions $\omega \in \mathrm{fPol}^{(k)}(\Gamma, \Delta)$ and $x \in D, y \in D^k$ we define $W_x^\omega(y) = \sum_{f \in \mathrm{Pol}^{(k)}(\Gamma) : f(y) = x} \omega(f)$. When there is no risk of confusion we drop the superscript and simply write $W_x(y)$.

For an instance I of Min-Cost-Hom(Γ, Δ), a variable v and a value d we use $\mathrm{Opt}(I, v \to d)$ to denote the optimal measure of a solution to I that maps v to d, i.e. $\min\{m(\varphi) : \varphi \in \mathrm{Sol}(I), \varphi(v) = x\}$. Using these definitions we obtain the following corollary of Lemma 8.

▶ **Lemma 11.** *Let* $I = (V, C, w)$ *be an instance of Min-Cost-Hom*(Γ, Δ) *and* $v \in V$ *be s.t.* $\{a_1, \dots, a_k\} \subseteq \{\varphi(v) : \varphi \in \mathrm{Sol}(I)\}$*. If* (Γ, Δ) *admits a* k*-ary fractional polymorphism* ω*, then*

$$\sum_{d \in D} W_d(a_1, \dots, a_k) \,\mathrm{Opt}(I, v \to d) \leq \frac{1}{k} \sum_{i=1}^{k} \mathrm{Opt}(I, v \to a_i).$$

▶ **Definition 12.** We say that $S \subseteq D$ is *shrinkable* to $S \setminus \{x\}$ in (Γ, Δ) if (Γ, Δ) admits a sequence of fractional polymorphisms $\omega_1, \dots, \omega_m$ and tuples $a^1 \in S^{k_1}, \dots, a^m \in S^{k_m}$ s.t. whenever $I = (V, C, w)$ is an instance of Min-Cost-Hom(Γ, Δ) with $v \in V$ s.t. $S \subseteq \{\varphi(v) : \varphi \in \mathrm{Sol}(I)\}$ it holds that the system of inequalities we obtain from Lemma 11 applied to ω_i and a^i, for $i \in [m]$, implies that

$$\sum_{i=1}^{n} t_i \,\mathrm{Opt}(I, v \to s_i) \leq \mathrm{Opt}(I, v \to x)$$

for some integer n, some $t_1, \dots, t_n \in \mathbb{Q}_{\geq 0}$ s.t. $\sum_{i=1}^{n} t_i = 1$, and some $s_1, \dots, s_n \in S \setminus \{x\}$.

If S is shrinkable to S' and S' is shrinkable to S'', then we say that S is shrinkable to S''.

So, if S is shrinkable to $S \setminus \{x\}$ in (Γ, Δ) there is a set of fractional polymorphisms of (Γ, Δ) with which we can prove the existence of some $s \in S \setminus \{x\}$ s.t. if I is an instance of Min-Cost-Hom(Γ, Δ), $v \in V$ and $S \subseteq \{\varphi(v) : \varphi \in \mathrm{Sol}(I)\}$, then $\mathrm{Opt}(I, v \to s) \leq \mathrm{Opt}(I, v \to x)$.

▶ **Example 13.** Consider the language $(\Gamma, \emptyset)$ on the domain D. Let $\{a_1, \ldots, a_m\} \subseteq D$. It is not hard to see that $\omega : f \mapsto \sum_{i=1}^{m-1} \frac{1}{m-1} \delta_{\mathrm{pr}_i, f}$ is in $\mathrm{fPol}^{(m)}(\Gamma, \emptyset)$. Hence, ω and $(a_1, \ldots, a_m)$ certifies that $\{a_1, \ldots, a_m\}$ is shrinkable to $\{a_1, \ldots, a_{m-1}\}$.

We can now define the second family of tractable languages.

▶ **Definition 14.** A finite language (Γ, Δ) on the domain D is said to be of *tournament pair type* if $\Gamma = \Gamma^c$, CSP(Γ) is in P and there exists $\mathcal{F} \subseteq \langle \Gamma, \Delta \rangle_w^{(1)}$, $\mathcal{A} \subseteq \binom{D}{2}$, $f_1, f_2 \in \mathrm{Pol}^{(2)}(\Gamma)$ and $g \in \mathrm{Pol}^{(3)}(\Gamma)$ s.t. the following holds.

- If $\{a, b\} \subseteq B$ for some $B \in \mathcal{F}$, and $\{a, b\} \notin \mathcal{A}$, then $f_1|_{\{a,b\}}$ and $f_2|_{\{a,b\}}$ are projections and $g|_{\{a,b\}}$ is arithmetical.
- If $\{a, b\} \subseteq B$ for some $B \in \mathcal{F}$, and $\{a, b\} \in \mathcal{A}$, then $f_1|_{\{a,b\}}$ and $f_2|_{\{a,b\}}$ are different idempotent, conservative and commutative operations.
- Every $S \in \langle \Gamma, \Delta \rangle_w^{(1)} \setminus \mathcal{F}$ is shrinkable to some $S' \in \mathcal{F}$.
- g is idempotent on every set in $\mathcal{F}$ and conservative on every set in $\mathcal{F} \setminus \mathcal{A}$.

▶ **Theorem 15.** *If (Γ, Δ) is a finite language of tournament pair type, then Min-Cost-Hom(Γ, Δ) is in PO.*

Proof sketch. Given an instance I of Min-Cost-Hom(Γ, Δ) we can, since CSP(Γ^c) is in P, compute for every variable v the set $D_v = \{\varphi(v) : \varphi \in \mathrm{Sol}(I)\}$. From the definition of shrinkable sets it is immediate that if D_v is shrinkable to $S \in \langle \Gamma, \Delta \rangle_w$, then we can add the constraint (v, S) to I without deteriorating the measure of an optimal solution. We can repeat this procedure until D_v is in $\mathcal{F}$ for every variable v.

It is known, see [24, Proof of Theorem 8], that from f_1, f_2, g one can construct (by superposition) operations f_1', f_2', g' that in addition to the conditions of the theorem also satisfy the following stronger properties:

- If $\{a, b\} \subseteq B$ for some $B \in \mathcal{F}$ and $\{a, b\} \notin \mathcal{A}$, then $f_1'|_{\{a,b\}} = f_2'|_{\{a,b\}} = \mathrm{pr}_1$.
- The operation g' is idempotent and conservative on every set in $\mathcal{F}$.

Clearly $f_1', f_2', g' \in \mathrm{Pol}(\Gamma)$. Note that f_1', f_2', g' preserves every unary relation $S \subseteq B$ for $B \in \mathcal{F}$. The result therefore follows from a reduction to the conservative, multi-sorted version of the problem and a result due to Takhanov for this variant [21, Theorem 23]. ◀

▶ **Example 16.** Consider again Min-Cost-Hom$(\Gamma, \emptyset)$. We saw in Example 13 that for every $\{x\} \subseteq X \subseteq D$ it holds that X is shrinkable to $\{x\}$. Hence, if $\Gamma^c = \Gamma$ and CSP(Γ) is in P it follows from Theorem 15 that Min-Cost-Hom$(\Gamma, \emptyset)$ is in PO. This of course is no surprise as Min-Cost-Hom$(\Gamma, \emptyset)$ essentially is the same problem as CSP(Γ).

4 Tools

Here we establish a collection of results that are used to prove the results in the last two sections. We hope this will provide an overview of the kind of techniques that are used to prove our main theorem.

Several of the results are proved with the help of the following classical theorem, see e.g. [19, p. 94].

▶ **Theorem 17** (Motzkin's Transposition Theorem)**.** *For any $A \in \mathbb{Q}^{m \times n}$, $B \in \mathbb{Q}^{p \times n}$, $b \in \mathbb{Q}^m$ and $c \in \mathbb{Q}^p$, exactly one of the following holds:*

- $Ax \leq b$, $Bx < c$ *for some* $x \in \mathbb{Q}^n$
- $A^T y + B^T z = 0$ *and* $(b^T y + c^T z < 0$ *or* $b^T y + c^T z = 0$ *and* $z \neq 0)$ *for some* $y \in \mathbb{Q}^m_{\geq 0}$ *and* $z \in \mathbb{Q}^p_{\geq 0}$

The first result concerns a slight generalisation of the concept of dominating fractional polymorphisms [24].

▶ **Definition 18.** Let $k \geq 2$ and $a \in D^{k-1}$, $b \in D$ be s.t. $a_1, \ldots, a_{k-1}, b$ are distinct elements. A fractional polymorphism $\omega \in \mathrm{fPol}^{(k)}(\Gamma, \Delta)$ is called $(a_1, \ldots, a_{k-1}, b)$-*dominating* if $W^{\omega}_{a_j}(a_1, \ldots, a_{k-1}, b) \geq \frac{1}{k}$ for every $j \in [k-1]$ and $\frac{1}{k} > W^{\omega}_{b}(a_1, \ldots, a_{k-1}, b)$.

▶ **Proposition 19.** *Let* (Γ, Δ) *be a finite language on a finite set* D*. Let* $k \geq 2$ *and* $a \in D^{k-1}$, $b \in D$ *be s.t.* $a_1, \ldots, a_{k-1}, b$ *are distinct. If* (Γ, Δ) *does not admit a fractional polymorphism that is* $(a_1, \ldots, a_{k-1}, b)$*-dominating, then* $\langle \Gamma, \Delta \rangle_e$ *contains a unary function* ν *that satisfies* $\infty > \nu(a_1), \ldots, \nu(a_{k-1}), \nu(b)$ *and* $\nu(c) > \nu(b)$ *for every* $c \in D \setminus \{b\}$.

Using similar arguments we can also prove the following characterisation of which relations that are wpp-definable in (Γ, Δ).

▶ **Proposition 20.** *Let* (Γ, Δ) *be a finite language on a finite set* D *and let* $\emptyset \neq R = \{t_1, \ldots, t_k\} \subseteq D^n$*. Exactly one of the following is true.*

1. *There exists* $\omega \in \mathrm{fPol}^{(k)}(\Gamma, \Delta)$ *with* $f \in \mathrm{supp}(\omega)$ *s.t.* $f(t_1, \ldots, t_k) \notin \{t_1, \ldots, t_k\}$.
2. *It holds that* $R \in \langle \Gamma, \Delta \rangle_w$.

From Proposition 20 we can quickly derive a number of useful results.

▶ **Corollary 21.** *Let* (Γ, Δ) *be a finite language on a finite set* D*. For any fixed* k *the set of wpp-definable* k*-ary relations,* $\langle \Gamma, \Delta \rangle^{(k)}_w$*, can be computed.*

Proof sketch. This is immediate from Proposition 20; we can find all polymorphisms of arities $1, \ldots, |D|^k$ and then, for every $R \subseteq D^k$, solve a linear program. ◀

▶ **Corollary 22.** *Let* (Γ, Δ) *be a finite language on a finite set* D *and let* $\{a, b\} \subseteq D$*. If there is* $\nu \in \langle \Gamma, \Delta \rangle_e$ *and* $A \subseteq D$ *s.t.* $\{a, b\} \subseteq A$, $A \in \langle \Gamma, \Delta \rangle_w$, $\nu(a) < \nu(b) < \infty$ *and* $\nu(b) \leq \nu(x)$ *for any* $x \in A \setminus \{a, b\}$*, then one of the following is true.*

1. $\{a, b\} \in \langle \Gamma, \Delta \rangle_w$
2. *There is* $\omega \in \mathrm{fPol}^{(2)}(\Gamma, \Delta)$ *that is* (a, b)*-dominating.*

Proof. Assume (1) does not hold. By Proposition 20 there must exist some $\omega \in \mathrm{fPol}^{(2)}(\Gamma, \Delta)$ with $f \in \mathrm{supp}(\omega)$ s.t. $f(a, b) \notin \{a, b\}$. It is not hard to see that in this case, because of ν, the fractional polymorphism ω must be (a, b)-dominating. Hence, (2) must be true. ◀

▶ **Corollary 23.** *Let* (Γ, Δ) *be a finite language on a finite set* D *and let* $\{a_1, \ldots, a_k\} \subseteq D$*. One of the following is true.*

1. *There is* $\omega \in \mathrm{fPol}^{(k)}(\Gamma, \Delta)$ *and* $i \in [k]$ *s.t.* ω *is* $(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_k, a_i)$*-dominating.*
2. *For every* $i \in [k]$ *there is* $j \in [k] \setminus \{i\}$ *s.t.* $\{a_i, a_j\} \in \langle \Gamma, \Delta \rangle_w$.

Proof. Assume (1) is false. By Proposition 19, for any $i \in [k]$, there is $\nu_i \in \langle \Gamma, \Delta \rangle_e$ s.t. $\arg\min_{x \in D} \nu_i(x) = \{a_i\}$ and $\nu_i(x) < \infty$ if $x \in \{a_1, \ldots, a_k\}$. Let $i \in [m]$. Pick j s.t. $\nu_i(a_j) = \min\{\nu_i(x) : x \in \{a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_k\}\}$.

Note that there is no $\psi \in \mathrm{fPol}^{(2)}(\Gamma, \Delta)$ that is (a_i, a_j)-dominating; if there was then

$$f \mapsto \sum_{i=1}^{k-2} \frac{1}{k} \delta_{\mathrm{pr}_i, f} + \sum_{g \in \mathrm{supp}(\psi)} \frac{2}{k} \psi(g) \delta_{g[\mathrm{pr}_{k-1}, \mathrm{pr}_k], f}$$

would be $(b_1, \dots, b_{k-2}, a_i, a_j)$-dominating for $b_1, \dots, b_{k-2} \in D$. Hence, by Corollary 22, we have $\{a_i, a_j\} \in \langle \Gamma, \Delta \rangle_w$. Since the choice of i was arbitrary (2) must be true. ◀

The generalised min-closed languages were introduced by Jonsson, Kuivinen and Nordh [12] and defined as sets of relations preserved by a particular type of binary operation. Kuivinen [16, Section 5.5] provides an alternative characterisation of the languages as those preserved by a so called min set function.

A *set function* [6] is a function $f : 2^D \setminus \{\emptyset\} \to D$. A *$\nu$-min set function* [16] is a set function f satisfying $\nu(f(X)) \leq \min\{\nu(x) : x \in X\}$ for every $X \in 2^D \setminus \{\emptyset\}$. The following proposition, which is a variant of [16, Theorem 5.18], will later prove to be useful.

▶ **Proposition 24.** *Let $(\Gamma, \{\nu\})$ be a finite language s.t. $\langle \Gamma, \{\nu\} \rangle_w^{(1)} \subseteq \Gamma$. The following are equivalent:*

1. *Γ is preserved by a ν-min set function,*
2. *Γ is preserved by a set function f s.t. $\nu(f(X)) = \min\{\nu(x) : x \in \bigcap_{Y \in \langle \Gamma \rangle : Y \supseteq X} Y\}$ for every $X \in 2^D \setminus \{\emptyset\}$,*
3. *Γ is preserved by a set function and for every $R \in \langle \Gamma \rangle$ it holds that*

$$R \cap (\operatorname*{arg\,min}_{x \in \mathrm{pr}_1(R)} \nu(x) \times \cdots \times \operatorname*{arg\,min}_{x \in \mathrm{pr}_{\mathrm{ar}(R)}(R)} \nu(x)) \neq \emptyset.$$

Furthermore, if ν is injective, then the following condition is equivalent to the ones above.

4. *For every $R \in \langle \Gamma \rangle$ it holds that*

$$R \cap (\operatorname*{arg\,min}_{x \in \mathrm{pr}_1(R)} \nu(x) \times \cdots \times \operatorname*{arg\,min}_{x \in \mathrm{pr}_{\mathrm{ar}(R)}(R)} \nu(x)) \neq \emptyset.$$

Let $\nu : D \to \mathbb{Q}_{\geq 0}$ be injective. We call the binary relation R a *cross* (with respect to ν) iff $|R| \geq 2$ and there are $\alpha_1, \alpha_2 \in \mathbb{Q}_{>0}$ s.t. $\alpha_1 \nu(t_1) + \alpha_2 \nu(t_2) = 1$ for every $t \in R$. The following lemma is a generalisation of [24, Lemma 25].

▶ **Lemma 25.** *Let $\nu : D \to \mathbb{Q}_{\geq 0}$ be injective. If Γ is not preserved by a ν-min set function, then $\langle \Gamma, \Delta \rangle_w$ contains a cross.*

Proof. Let $\min_\nu$ be the unique set function satisfying $\{\min_\nu(X)\} = \arg\min_{x \in X} \nu(x)$ for every $X \subseteq D$. If Γ is not preserved by a ν-min set function, then Proposition 24 implies that there is $R \in \langle \Gamma \rangle$ s.t. $(\min_\nu(\mathrm{pr}_1(R)), \dots, \min_\nu(\mathrm{pr}_{\mathrm{ar}(R)}(R))) \notin R$.

In fact, there must be a binary relation in $\langle \Gamma \rangle$ of this kind. To see this let $R \in \langle \Gamma \rangle$ be a k-ary relation s.t. $(\min_\nu(\mathrm{pr}_1(R)), \dots, \min_\nu(\mathrm{pr}_k(R))) \notin R$ and s.t. that every relation $R' \in \langle \Gamma \rangle$ of smaller arity satisfies $(\min_\nu(\mathrm{pr}_1(R')), \dots, \min_\nu(\mathrm{pr}_{\mathrm{ar}(R')}(R'))) \in R'$. This means that there is $t^1 \in R$ s.t. $t^1_i = \min_\nu(\mathrm{pr}_i(R))$ for $i \in [k] \setminus \{1\}$, otherwise $\mathrm{pr}_{2,\dots,\mathrm{ar}(R)}(R)$ contradicts the minimality of k. Similarly there is $t^2 \in R$ s.t. $t^2_i = \min_\nu(\mathrm{pr}_i(R))$ for $i \in [k] \setminus \{2\}$. This means that $R' = \{(x, y) : (x, y, \min_\nu(\mathrm{pr}_3(R)), \dots, \min_\nu(\mathrm{pr}_k(R))) \in R\}$ is a non-empty relation of arity 2 s.t. $(\min_\nu(\mathrm{pr}_1(R')), \min_\nu(\mathrm{pr}_2(R'))) \notin R'$. Hence, $k = 2$.

Clearly we can choose α_1, α_2 s.t. $R'' = \arg\min_{(x,y) \in R}(\alpha_1 \nu(x) + \alpha_2 \nu(y))$ satisfies $|R''| \geq 2$, and $R'' \in \langle \Gamma, \Delta \rangle_w$ is a cross. ◀

To prove that a given language is computationally hard we make use of the following lemma which is an immediate consequence of [20, Theorem 3.1].

▶ **Lemma 26.** *If $\{a, b\} \in \Gamma$ and $\nu(a) < \nu(b) < \infty$, $\sigma(b) < \sigma(a) < \infty$ for some $\nu, \sigma \in \Delta$, then either*

- *there exists* $f_1, f_2 \in \mathrm{Pol}^{(2)}(\Gamma)$ *s.t.* $f_1|_{\{a,b\}}$ *and* $f_2|_{\{a,b\}}$ *are two different idempotent, commutative and conservative operations,*
- *there exists* $g \in \mathrm{Pol}^{(3)}(\Gamma)$ *s.t.* $g|_{\{a,b\}}$ *is arithmetical, or*
- *Min-Cost-Hom*(Γ, Δ) *is NP-hard.*

The following result by Takhanov [20, Theorem 5.4] shows how "partially arithmetical" polymorphisms (like the ones that we might get out of the previous lemma) can be stitched together.

▶ **Lemma 27.** *Let* $\mathcal{C} \subseteq \binom{D}{2}$*. If* $\mathcal{C} \subseteq \Gamma$ *and for each* $\{a, b\} \in \mathcal{C}$ *an operation in* $\mathrm{Pol}^{(3)}(\Gamma)$ *is arithmetical on* $\{a, b\}$*, then there is an operation in* $\mathrm{Pol}^{(3)}(\Gamma)$ *that is arithmetical on* $\mathcal{C}$*.*

The next lemma is a variation, see [24, Lemma 14], of a lemma by Thapper and Živný [23, Lemma 3.5]. It allows us to prove the existence of certain nontrivial fractional polymorphisms. We may also obtain this lemma as a simple corollary of Proposition 20.

▶ **Lemma 28.** *If* $\{(a, b), (b, a)\} \notin \langle \Gamma, \Delta \rangle_w$*, then for all* $\sigma \in \langle \Gamma, \Delta \rangle_e$ *there is* $\omega \in \mathrm{fPol}^{(2)}(\Gamma, \Delta)$ *with* $f \in \mathrm{supp}(\omega)$ *s.t.* $\{f(a, b), f(b, a)\} \neq \{a, b\}$ *and* $\sigma(f(a, b)) + \sigma(f(b, a)) \leq \sigma(a) + \sigma(b)$*.*

Finally, the following lemmas are used to "canonicalise" interesting fractional polymorphisms.

▶ **Definition 29.** Let $P \subseteq \mathcal{O}_D^{(2)}$. For a function $\omega : P \to \mathbb{Q}_{\geq 0}$ we define $\omega^2 : P \to \mathbb{Q}_{\geq 0}$ by $\omega^2(f) = \sum_{g,h \in P : g[h,\overline{h}]=f} \omega(g)\omega(h)$.

▶ **Lemma 30.** *If* $\omega \in \mathrm{fPol}^{(2)}(\Gamma, \Delta)$*, then* $\omega^2 \in \mathrm{fPol}^{(2)}(\Gamma, \Delta)$*.*

▶ **Lemma 31.** *Let* $\beta : D^2 \to \mathbb{Q}_{\geq 0}$ *and define* $C_\omega(x) = \sum_{f \in \mathrm{Pol}^{(2)}(\Gamma) : f(x) = \overline{f}(x)} \omega(f)$ *and* $M(\omega) = \sum_{x \in D^2} C_\omega(x)$*. Set* $\Omega = \{\omega \in \mathrm{fPol}^{(2)}(\Gamma, \Delta) : \forall s \in D^2, C_\omega(s) \geq \beta(s)\}$*. If* $\langle \Gamma, \Delta \rangle_w^{(1)} \subseteq \Gamma$*, then either* $\Omega = \emptyset$*, or there is* $\omega^* \in \Omega$ *s.t.* $M(\omega^*) = \sup_{\omega \in \Omega} M(\omega)$*.*

▶ **Lemma 32.** *Let* $\mathcal{S} \subseteq \binom{D}{2}$ *and* $\Pi = \{\omega \in \mathrm{fPol}^{(2)}(\Gamma, \Delta) :$ *for all* $S \in \mathcal{S}$ *there exists* $f \in \mathrm{supp}(\omega)$ *s.t.* $f|_S$ *is commutative*$\}$*. If* $\langle \Gamma, \Delta \rangle_w^{(1)} \subseteq \Gamma$ *and* $\Pi \neq \emptyset$*, then there is* $\omega \in \Pi$ *s.t. for every* $f \in \mathrm{supp}(\omega)$ *and* $x \in D^2$ *it holds that* $\{f(x), \overline{f}(x)\} \notin \mathcal{S}$*.*

5 Cores

In this section we define cores and prove that one can add all constants to a language that is a core without making the associated extended Min-Cost-Hom much more difficult. We use a definition of cores from [23, Definition 3].

▶ **Definition 33.** A finite language (Γ, Δ) is a *core* iff for every $\omega \in \mathrm{fPol}^{(1)}(\Gamma, \Delta)$ and every $f \in \mathrm{supp}(\omega)$ it holds that f is injective. A language (Γ', Δ') is a core of another language (Γ, Δ) if (Γ', Δ') is a core and $(\Gamma', \Delta') = (\Gamma, \Delta)|_{g(D)}$ for some $\psi \in \mathrm{fPol}^{(1)}(\Gamma, \Delta)$ and $g \in \mathrm{supp}(\psi)$.

A result very similar to the following was given in [10, 23] for finite-valued languages.

▶ **Proposition 34.** *If* (Γ, Δ) *is a core, then Min-Cost-Hom*(Γ^c, Δ) *is polynomial-time reducible to Min-Cost-Hom*(Γ, Δ)*.*

Proof sketch. We will show that Min-Cost-Hom(Γ^c, Δ) is polynomial-time reducible to Min-Cost-Hom$(\Gamma \cup \langle \Gamma, \Delta \rangle_w^{(|D|)}, \Delta)$. By Theorem 4 this is sufficient.

Assume $D = \{d_1, \ldots, d_{|D|}\}$. Let $R = \{(d_1, \ldots, d_{|D|})\}$ and let R' be the closure of R under the operations $f \in \mathrm{supp}(\omega)$, $\omega \in \mathrm{fPol}^{(1)}(\Gamma, \Delta)$.

Note that there is no $k > 1$, $\psi \in \mathrm{fPol}^{(k)}(\Gamma, \Delta)$ and $g \in \mathrm{supp}(\psi)$ s.t. g does not preserve R'. This follows from the fact that R' was generated from a single tuple. It is not hard to show that there is $\varpi \in \mathrm{fPol}^{(1)}(\Gamma, \Delta)$ s.t. $R' = \{f(d_1, \ldots, d_{|D|}) : f \in \mathrm{supp}(\varpi)\}$. Assume that there is $s = f(t^1, \ldots, t^k) \notin R'$ for some $f \in \mathrm{supp}(\psi)$ and $t^1, \ldots, t^k \in R'$. This means that we from ψ and ϖ can construct $\varpi' \in \mathrm{fPol}^{(1)}(\Gamma, \Delta)$ with $f \in \mathrm{supp}(\varpi')$ s.t. $s = f(d_1, \ldots, d_{|D|})$, which is a contradiction.

From Proposition 20 it follows that $R' \in \langle \Gamma, \Delta \rangle_w$. Since (Γ, Δ) is a core, for every $\omega \in \mathrm{fPol}^{(1)}(\Gamma, \Delta)$ and $f \in \mathrm{supp}(\omega)$ we know that f is injective. Hence, every $t \in R'$ equals $(\pi(d_1), \ldots, \pi(d_{|D|}))$ for some permutation π on D.

We now use a construction that is applied for the corresponding result for CSPs [2, Theorem 4.7]. Given an instance I of Min-Cost-Hom(Γ^c, Δ) we create an instance of I' of Min-Cost-Hom$(\Gamma \cup \langle \Gamma, \Delta \rangle_w^{(|D|)}, \Delta)$ from I by adding variables $v_{d_1}, \ldots, v_{d_{|D|}}$ and replacing every constraint $(v, \{d_i\})$ with the constraint $((v, v_{d_i}), =)$. Finally we add the constraint $((v_{d_1}, \ldots, v_{d_{|D|}}), R')$. If there is a solution to I, then there is also a solution to I'. And, if ψ is an optimal solution to I', then $(\varphi(v_{d_1}), \ldots, \varphi(v_{d_{|D|}})) = (\pi(d_1), \ldots, \pi(d_{|D|}))$ for some permutation π on D and $\omega \in \mathrm{fPol}^{(1)}(\Gamma, \Delta)$ s.t. $\pi \in \mathrm{supp}(\omega)$. Hence $\pi^k \circ \psi$ is another optimal solution to I', for any $k \geq 1$. In particular there is an optimal solution φ^* to I' s.t. $(\varphi^*(v_{d_1}), \ldots, \varphi^*(v_{d_{|D|}})) = (d_1, \ldots, d_{|D|})$. This allows us to recover an optimal solution to I. ◀

6 Proof of Theorem 1

In this section we prove our main result. To do this we rely of a few lemmas that are proved with the help of a fair bit of case analysis. For their proofs we refer the interested reader to the longer version of this paper.

Let A denote the following assumption: (Γ, Δ) is a finite language on $D = \{a, b, c\}$ s.t. $\Gamma^c \cup \mathrm{Feas}(\Delta) \cup \langle \Gamma, \Delta \rangle_w^{(1)} \cup \langle \Gamma, \Delta \rangle_w^{(2)} \subseteq \Gamma$.

The supporting lemma below is used in the proofs of the results that follow.

▶ **Lemma 35.** *Assume A. If $\{a, b\} \notin \Gamma$, then either there is $\omega \in \mathrm{fPol}^{(2)}(\Gamma, \Delta)$ that is (a, b) or (b, a)-dominating, or there are $\nu_a, \nu_b \in \langle \Gamma, \Delta \rangle_e$ s.t. $\nu_a(a) < \nu_a(c) < \nu_a(b)$ and $\nu_b(b) < \nu_b(c) < \nu_b(a)$.*

We are going to analyse a few different cases depending on the number of two-element subsets of the domain that is wpp-definable in (Γ, Δ). The following lemma, which follows immediately from Corollary 23, connects this number to dominating fractional polymorphisms.

▶ **Lemma 36.** *Assume A. Either $|\Gamma \cap \binom{D}{2}| \geq 2$ or there is $\omega \in \mathrm{fPol}^{(3)}(\Gamma, \Delta)$ and $a_1, a_2, a_3 \in D$ s.t. ω is (a_1, a_2, a_3)-dominating and $\{a_1, a_2, a_3\} = D$.*

To understand languages that admit a ternary dominating fractional polymorphism we use the following lemma.

▶ **Lemma 37.** *Assume A. If $\{a, b\} \notin \Gamma$ and there is $\omega \in \mathrm{fPol}^{(3)}(\Gamma, \Delta)$ s.t. ω is (a, b, c)-dominating, then either $\{a, c\}, \{b, c\} \in \Gamma$, or (Γ, Δ) is of semilattice type or of tournament pair type, or Min-Cost-Hom(Γ, Δ) is NP-hard*

The following four lemmas are used to handle languages that contain two unary two-element relations.

▶ **Lemma 38.** *Assume A. If $\{a, c\}, \{c, b\} \in \Gamma$ and there is $\omega \in \mathrm{fPol}^{(2)}(\Gamma, \Delta)$ that is (a, b)-dominating, then (Γ, Δ) is of tournament pair type or Min-Cost-Hom(Γ, Δ) is NP-hard.*

▶ **Lemma 39.** *Assume A. If* $\{a,b\} \notin \Gamma$ *and* $\{a,c\},\{c,b\} \in \Gamma$*, then either* $\{(a,c),(c,a)\} \in \Gamma$*,* $\{(b,c),(c,b)\} \in \Gamma$*, or* (Γ,Δ) *is of semilattice type or of tournament pair type, or Min-Cost-Hom*(Γ,Δ) *is NP-hard*

▶ **Lemma 40.** *Assume A. If* $\{a,b\} \notin \Gamma$*,* $\{a,c\},\{c,b\} \in \Gamma$ *and* $\{(a,c),(c,a)\} \in \Gamma$ *and* $\{(b,c),(c,b)\} \notin \Gamma$*, then* (Γ,Δ) *is of tournament pair type or Min-Cost-Hom*(Γ,Δ) *is NP-hard.*

▶ **Lemma 41.** *Assume A. If* $\{a,b\} \notin \Gamma$ *and* $\{(a,c),(c,a)\},\{(b,c),(c,b)\} \in \Gamma$*, then* (Γ,Δ) *is of tournament pair type or Min-Cost-Hom*(Γ,Δ) *is NP-hard.*

We can now prove the main theorem.

Proof of Theorem 1. Let $\Gamma' = \langle\Gamma,\Delta\rangle_w^{(1)} \cup \langle\Gamma,\Delta\rangle_w^{(2)} \cup \Gamma^c \cup \mathrm{Feas}(\Delta)$.

Note that if (Γ',Δ) is of semilattice type or of tournament pair type, then so is $(\Gamma^c \cup \mathrm{Feas}(\Delta),\Delta)$. Furthermore, by Theorem 4 and Proposition 34 we know that Min-Cost-Hom(Γ',Δ) is polynomial time reducible to Min-Cost-Hom(Γ,Δ). Hence, if Min-Cost-Hom(Γ',Δ) is NP-hard, then also Min-Cost-Hom(Γ,Δ) is NP-hard.

Clearly, if CSP(Γ') is NP-hard, then so is Min-Cost-Hom(Γ',Δ). And, if CSP(Γ') is not NP-hard, then it is in P. This follows from [1].

If $|\binom{D}{2} \cap \Gamma'| = 3$, then (Γ',Δ) is of tournament pair type or Min-Cost-Hom(Γ',Δ) is NP-hard. This follows from [24, Theorem 12].

If $|\binom{D}{2} \cap \Gamma'| < 2$, then, by Lemma 36, we know that there is $\omega \in \mathrm{fPol}^{(3)}(\Gamma',\Delta)$ that is (a_1,a_2,a_3)-dominating for some $\{a_1,a_2,a_3\} = D$. If $\{a_1,a_2\} \notin \Gamma'$, then by Lemma 37 we know that either $|\binom{D}{2} \cap \Gamma'| = 2$ (a contradiction) or (Γ',Δ) is of semilattice type or of tournament pair type, or Min-Cost-Hom(Γ',Δ) is NP-hard. Otherwise $\{a_1,a_2\} \in \Gamma'$. Since $|\binom{D}{2} \cap \Gamma'| < 2$ it must hold that $\{a_1,a_3\} \notin \Gamma'$ and $\{a_2,a_3\} \notin \Gamma'$. In this case, since $\{a_1,a_2,a_3\}$ is shrinkable to $\{a_1,a_2\}$, it holds that either (Γ',Δ) is of tournament pair type or Min-Cost-Hom(Γ',Δ) is NP-hard.

The only remaining case is $|\binom{D}{2} \cap \Gamma'| = 2$. In this case the result follows from Lemma 39, Lemma 40 and Lemma 41. ◀

Acknowledgements. I am thankful to Peter Jonsson for rewarding discussions and to the anonymous reviewers for their helpful comments.

References

1 Andrei Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006.

2 Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.

3 David Cohen, Martin Cooper, Páidí Creed, Peter Jeavons, and Stanislav Živný. An algebraic theory of complexity for discrete optimization. *SIAM Journal on Computing*, 42(5):1915–1939, 2013.

4 David Cohen, Martin Cooper, and Peter Jeavons. An algebraic characterisation of complexity for valued constraints. In *Proceedings of CP 2006*, volume 4204 of *Lecture Notes in Computer Science*, pages 107–121. Springer Berlin Heidelberg, 2006.

5 David Cohen, Martin Cooper, Peter Jeavons, and Andrei Krokhin. The complexity of soft constraint satisfaction. *Artificial Intelligence*, 170(11):983–1016, 2006.

6 Víctor Dalmau and Justin Pearson. Closure functions and width 1 problems. In *Proceedings of CP 1999*, volume 1713 of *Lecture Notes in Computer Science*, pages 159–173. Springer Berlin Heidelberg, 1999.

7 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.

8 Satoru Fujishige. *Submodular Functions and Optimization*, volume 58 of *Annals of Discrete Mathematics*. Elsevier, 2005.

9 Gregory Gutin, Arash Rafiey, Anders Yeo, and Michael Tso. Level of repair analysis and minimum cost homomorphisms of graphs. *Discrete Applied Mathematics*, 154(6):881–889, 2006.

10 Anna Huber, Andrei Krokhin, and Robert Powell. Skew bisubmodularity and valued CSPs. In *Proceedings of SODA 2013*, pages 1296–1305, 2013.

11 Peter Jeavons, David Cohen, and Marc Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–548, 1997.

12 Peter Jonsson, Fredrik Kuivinen, and Gustav Nordh. MAX ONES generalized to larger domains. *SIAM Journal on Computing*, 38(1):329–365, 2008.

13 Peter Jonsson, Fredrik Kuivinen, and Johan Thapper. Min CSP on four elements: Moving beyond submodularity. In *Proceedings of CP 2011*, volume 6876 of *Lecture Notes in Computer Science*, pages 438–453. Springer Berlin Heidelberg, 2011.

14 Peter Jonsson and Gustav Nordh. Introduction to the maximum solution problem. In *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 255–282. Springer Berlin Heidelberg, 2008.

15 Vladimir Kolmogorov. The power of linear programming for finite-valued CSPs: A constructive characterization. In *Proceedings of ICALP 2013*, volume 7965 of *Lecture Notes in Computer Science*, pages 625–636. Springer Berlin Heidelberg, 2013.

16 Fredrik Kuivinen. *Algorithms and Hardness Results for Some Valued CSPs*. PhD thesis, Linköping University, The Institute of Technology, 2009.

17 Creignou Nadia, Khanna Sanjeev, and Sudan Madhu. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

18 Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of IJCAI 1995*, pages 631–637, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

19 Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

20 Rustem Takhanov. A dichotomy theorem for the general minimum cost homomorphism problem. In *Proceedings of STACS 2010*, number 5 in Leibniz International Proceedings in Informatics (LIPIcs), pages 657–668, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

21 Rustem Takhanov. Extensions of the minimum cost homomorphism problem. In *Proceedings of COCOON 2010*, volume 6196 of *Lecture Notes in Computer Science*, pages 328–337. Springer Berlin Heidelberg, 2010.

22 Johan Thapper and Stanislav Živný. The power of linear programming for valued CSPs. In *Proceedings of FOCS 2012*, pages 669–678. IEEE Computer Society, 2012.

23 Johan Thapper and Stanislav Živný. The complexity of finite-valued CSPs. In *Proceedings of STOC 2013*, pages 695–704, New York, NY, USA, 2013. ACM.

24 Hannes Uppman. The complexity of three-element Min-Sol and conservative Min-Cost-Hom. In *Proceedings of ICALP 2013*, volume 7965 of *Lecture Notes in Computer Science*, pages 804–815. Springer Berlin Heidelberg, 2013.

The Complexity of Deciding Statistical Properties of Samplable Distributions*

Thomas Watson

University of Toronto, Toronto, Canada
thomasw@cs.toronto.edu

Abstract

We consider the problems of deciding whether the joint distribution sampled by a given circuit satisfies certain statistical properties such as being i.i.d., being exchangeable, being pairwise independent, having two coordinates with identical marginals, having two uncorrelated coordinates, and many other variants. We give a proof that simultaneously shows all these problems are $C_=P$-complete, by showing that the following promise problem (which is a restriction of all the above problems) is $C_=P$-complete: Given a circuit, distinguish the case where the output distribution is uniform and the case where every pair of coordinates is neither uncorrelated nor identically distributed. This completeness result holds even for samplers that are depth-3 circuits.

We also consider circuits that are d-local, in the sense that each output bit depends on at most d input bits. We give linear-time algorithms for deciding whether a 2-local sampler's joint distribution is fully independent, and whether it is exchangeable.

We also show that for general circuits, certain approximation versions of the problems of deciding full independence and exchangeability are SZK-complete.

We also introduce a bounded-error version of $C_=P$, which we call $BC_=P$, and we investigate its structural properties.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Complexity, statistical properties, samplable distributions

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.663

1 Introduction

Testing for independence of random variables is a fundamental problem in statistics. Theoretical computer scientists have studied this and other analogous problems from two main viewpoints. The first viewpoint is property testing of distributions, which is a black-box model in which a tester is given samples and tries to distinguish between some statistical property being "close" or "far" from satisfied. Some important works giving upper and lower bounds for property testing of distributions include [4, 3, 5, 2, 14, 18, 7].

The other viewpoint is the non-black-box model in which a tester is given a description of a distribution (from which it could generate its own samples). This could potentially make some problems easier, but there are complexity-theoretic results showing that several such problems are computationally hard, particularly when the input is a *succinct* description of a distribution. One of the most general and natural ways to succinctly specify a distribution is to give the code of an efficient algorithm that takes "pure" randomness and transforms it into a sample from the distribution. (This gives a polynomial-size specification of a distribution over a potentially exponential-size set.) For arbitrary circuit samplers, the

* Supported by funding from NSERC.


© Thomas Watson;
licensed under Creative Commons License CC-BY
31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 663–674

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


papers [15, 10, 11, 22] contain completeness results for various approximation problems concerning statistical distance, Shannon entropy, and min-entropy. See [12] for a survey of both the black-box and the non-black-box viewpoints.

In this paper we consider a wide array of "exact" problems concerning statistical properties of the joint distribution produced by a given sampler. Such problems include deciding whether the joint distribution is i.i.d., exchangeable, pairwise independent, and many other variants. Exchangeability is a very important and useful concept with many different applications in pure and applied probability [1], but it has been less-often studied in the theoretical computer science community. A joint distribution over a finite domain is called *exchangeable* if it is invariant under permuting the coordinates. It is fairly straightforward to see that a finite distribution is exchangeable iff it is a mixture of distributions that arise from drawing a sequence of colored balls without replacement from an urn. When each coordinate is a single bit, exchangeability is equivalent to the probability of a string only depending on the Hamming weight. We feel it is natural to pose complexity-theoretic questions about exchangeability.

We prove that the aforementioned wide array of problems, and more generally a single problem we call Panoptic-Stats which is no harder than any of those problems, are complete for the complexity class $\mathrm{C_{=}P}$. This class was introduced in [21] as part of the counting hierarchy, and it can be viewed as a class that captures "exact counting" of NP witnesses. The class $\mathrm{C_{=}P}$ is at least as hard as the polynomial-time hierarchy, since $\mathrm{PH} \subseteq \mathrm{BP} \cdot \mathrm{C_{=}P}$ [17] and even $\mathrm{PH} \subseteq \mathrm{ZP} \cdot \mathrm{C_{=}P}$ [16]. It is no harder than "threshold counting", since $\mathrm{C_{=}P} \subseteq \mathrm{PP}$, but neither is it substantially easier, since $\mathrm{PP} \subseteq \mathrm{NP}^{\mathrm{C_{=}P}}$. It was shown in [9] that $\mathrm{C_{=}P} = \mathrm{coNQP}$.

In many areas of complexity theory, when arbitrary small-size circuits are too unwieldy to reason about, we restrict our attention to more stringent complexity measures, such as parallel time, that are combinatorially simple enough to reason about and obtain unconditional results. One model of efficient parallel time computation is AC^0 (constant-depth unbounded fan-in circuits with AND, OR, and NOT gates). Papers that study AC^0 circuits that sample distributions include [20, 13, 19, 6]. Another (generally more restrictive) model of efficient parallel time computation is *locally-computable* functions, where each output bit depends on at most a bounded number of input bits. Papers that study locally-computable functions as samplers include [20, 8, 19, 22] as well as a large collection of papers investigating the possibility of implementing pseudorandom generators locally. (See [8] for an extensive list of past work on the power of locally-computable functions, including whether they can implement PRGs, one-way functions, and extractors.)

We prove that our $\mathrm{C_{=}P}$-completeness results hold even when restricted to samplers that are AC^0-type circuits with depth 3 and top fan-in 2 (i.e., each output gate has fan-in at most 2). We also consider 2-local samplers (where each output bit depends on at most 2 of the pure random input bits) such that each coordinate of the sampled joint distribution is a single bit. We give polynomial-time (in fact, linear-time) algorithms for deciding whether such a sampler's distribution is fully independent, and whether it is exchangeable. These seem to be the first-of-a-kind algorithmic results on deciding statistical properties of succinctly described distributions.

We also consider approximate versions of the problems discussed above: deciding whether the joint distribution of a given sampler is statistically close or far from satisfying a property. It was shown in [10] that for the property of being uniform, the problem is NISZK-complete. It was shown in [15] that the problem of deciding whether a pair of samplable distributions are statistically close or far is complete for SZK (statistical zero knowledge). We prove that with suitable parameters, the approximate versions of the full independence and exchangeability problems (for general circuit samplers) are also SZK-complete.

In this paper we also consider a "bounded-error" version of $\mathrm{C}_{=}\mathrm{P}$, which we call $\mathrm{BC}_{=}\mathrm{P}$ and which does not seem to have been defined or studied in the literature before. Although it does not appear to be directly relevant to statistical properties of samplable distributions, we take the opportunity to study this class and prove that it is closed under several operations (disjunction, conjunction, union, and intersection).

2 Results

If D is a joint distribution over $(\{0,1\}^k)^n$, we let D_i (for $i \in \{1,\ldots,n\}$) denote the i^{th} coordinate, which is marginally distributed over $\{0,1\}^k$. For each of the computational problems we consider, the input is a circuit $S : \{0,1\}^r \to (\{0,1\}^k)^n$ (and we assume that the values of k and n are part of the description of the circuit). We call such a circuit a (k,n)-sampler, and if it has size $\leq s$ we also call it a (k,n,s)-sampler. Plugging a uniformly random string into S yields a joint output distribution, which we denote by $S(U)$.

We formulate computational problems using the framework of promise problems. Throughout this paper, when we talk about reductions and completeness, we are always referring to deterministic polynomial-time mapping reductions.

We state our completeness results for exact problems in Section 2.1 and prove them in Section 3 and the full version. We state our algorithmic results for exact problems in Section 2.2 and prove them in Section 4 and the full version. We state our completeness results for approximate problems in Section 2.3 and prove them in Section 5 and the full version. In the full version, we also study a new complexity class, $\mathrm{BC}_{=}\mathrm{P}$.

2.1 Exact Completeness Results

For a joint distribution D over $(\{0,1\}^k)^n$, we say that D_i, D_j are *uncorrelated* if they have covariance 0, in other words $\mathrm{E}(D_i \cdot D_j) = \mathrm{E}(D_i) \cdot \mathrm{E}(D_j)$ (when $\{0,1\}^k$ is interpreted as binary representations of nonnegative integers). Uncorrelated is the same as independent if $k = 1$. We consider the following extreme notion of a distribution being nonuniform.

▶ **Definition 1.** A joint distribution is *discordant* if there are ≥ 2 coordinates and every pair of coordinates is neither uncorrelated nor identically distributed.

▶ **Definition 2.** Panoptic-Stats is the following promise problem.

$$\begin{aligned} \textsc{Panoptic-Stats}_{\text{YES}} &= \{S \ : \ S(U) \text{ is uniform}\} \\ \textsc{Panoptic-Stats}_{\text{NO}} &= \{S \ : \ S(U) \text{ is discordant}\} \end{aligned}$$

We say that promise problem Π is a generalization of promise problem Π', or that Π' is a restriction of Π, if $\Pi'_{\text{YES}} \subseteq \Pi_{\text{YES}}$ and $\Pi'_{\text{NO}} \subseteq \Pi_{\text{NO}}$.

▶ Fact 1. Panoptic-Stats is generalized by all the following languages, which are defined in a natural way.

Uniform, Iid, Fully-Independent, Identically-Distributed,
Exchangeable, K-Wise-Uniform, K-Wise-Independent,
K-Wise-Exchangeable, 2-Wise-Uncorrelated, K-Exists-Uniform,
K-Exists-Independent, K-Exists-Identically-Distributed,
K-Exists-Exchangeable, 2-Exists-Uncorrelated, Non-Discordant

For example, $S \in \textsc{Uniform} \iff S(U)$ is uniform. Also, $K \geq 2$ is any constant (unrelated to k). Technical caveat: To ensure the K-Wise- and K-Exists- problems generalize

Panoptic-Stats, they are defined in terms of a property holding for every or some (respectively) set of $\min(K, n)$ coordinates.

We prove that Panoptic-Stats and all the languages listed in Fact 1 are complete for the complexity class $\mathrm{C}_{=}\mathrm{P}$. In fact, the $\mathrm{C}_{=}\mathrm{P}$-hardness of each of the individual languages in Fact 1 is fairly simple to prove, but the $\mathrm{C}_{=}\mathrm{P}$-hardness of Panoptic-Stats shows two things: (1) that this phenomenon is very robust, not dependent on some fragile aspects of the properties being decided, and (2) that only one proof is needed to show the $\mathrm{C}_{=}\mathrm{P}$-hardness of all the languages in Fact 1.

To prove the $\mathrm{C}_{=}\mathrm{P}$-hardness of Panoptic-Stats, it suffices to prove hardness for the case $n = 2$. However, hardness for $n = 2$ does not seem to directly imply hardness for a larger number of coordinates; it is desirable to prove hardness even when restricted to samplers that are small in terms of the number of coordinates n. We formalize this by introducing a new parameter m and viewing k, n, s as functions of m. Thus m can be thought of as indexing a family of parameter settings.

▶ **Definition 3.** We say that a triple of functions $\kappa(m), \nu(m), \sigma(m) : \mathbb{N} \to \mathbb{N}$ is *polite* if the functions are monotonically nondecreasing, polynomially bounded in m, computable in time polynomial in m, and $\sigma(m) \geq m$.

▶ **Definition 4.** Panoptic-Stats$^{\kappa,\nu,\sigma}$ is the restriction of Panoptic-Stats to (k, n, s)-samplers with $k = \kappa(m)$, $n = \nu(m)$, and $s \leq \sigma(m)$ for some m.

$\mathrm{prC}_{=}\mathrm{P}$ is the class of promise problems for which there exists a polynomial-time randomized algorithm M that accepts with probability $\frac{1}{2}$ on YES instances, and accepts with probability $\neq \frac{1}{2}$ on NO instances. Here we use a standard model of computation in which randomized algorithms have access to independent unbiased coin flips. We use the following equivalent definition of $\mathrm{prC}_{=}\mathrm{P}$.

▶ **Definition 5.** $\mathrm{prC}_{=}\mathrm{P}$ is the class of all promise problems reducible to the following promise problem Uniform-Bit.

$$\begin{aligned} \textsc{Uniform-Bit}_{\mathrm{YES}} &= \{S \ : \ S \text{ is a } (1,1)\text{-sampler and } S(U) \text{ is uniform}\} \\ \textsc{Uniform-Bit}_{\mathrm{NO}} &= \{S \ : \ S \text{ is a } (1,1)\text{-sampler and } S(U) \text{ is nonuniform}\} \end{aligned}$$

$\mathrm{C}_{=}\mathrm{P}$ is defined as the class of languages in $\mathrm{prC}_{=}\mathrm{P}$.

▶ **Theorem 6.** Panoptic-Stats$^{\kappa,\nu,\sigma}$ *is* $\mathrm{prC}_{=}\mathrm{P}$*-hard for every polite* κ, ν, σ *with* $\kappa\nu \leq o(\sigma)$.

▶ **Theorem 7.** Panoptic-Stats$^{\kappa,\nu,\sigma}$ *is* $\mathrm{prC}_{=}\mathrm{P}$*-hard even when restricted to samplers that are* AC^0*-type circuits with depth* 3 *and top fan-in* 2*, for every polite* κ, ν, σ *with* $\kappa\nu + \nu^2 \leq o(\sigma)$.

▶ **Theorem 8.** *All the languages listed in Fact 1 are in* $\mathrm{C}_{=}\mathrm{P}$.

Consequently, all the languages listed in Fact 1 are $\mathrm{C}_{=}\mathrm{P}$-complete, even when restricted to (κ, ν, σ)-samplers (like in Definition 4) with polite κ, ν, σ satisfying $\kappa\nu \leq o(\sigma)$ (for general circuit samplers) or satisfying $\kappa\nu + \nu^2 \leq o(\sigma)$ (for depth-3 circuits with top fan-in 2).

2.2 Exact Algorithmic Results

We say a (k, n, s)-sampler is *d-local* if each of the kn output bits depends on at most d of the uniformly random input bits. For d-local samplers, if $dk \leq O(\log s)$ then some statistical properties, such as being pairwise independent or having identically distributed marginals,

can be decided trivially in polynomial time. We now prove that some other properties, namely being fully independent or being exchangeable, can be decided in polynomial time when $d = 2$ and $k = 1$. (Admittedly, our algorithms are not very "algorithmic"; we prove combinatorial characterizations for which it is simple to check whether a given sampler satisfies the characterization.)

▶ **Theorem 9.** *There exists a linear-time algorithm for deciding whether the joint distribution of a given 2-local $(1, n)$-sampler is fully independent.*

▶ **Theorem 10.** *There exists a linear-time algorithm for deciding whether the joint distribution of a given 2-local $(1, n)$-sampler is exchangeable.*

When $d = 2$ and $k = 1$, we can also improve the efficiency of the trivial quadratic-time algorithm for deciding pairwise independence.

▶ **Theorem 11.** *There exists a linear-time reduction from the problem of deciding whether the joint distribution of a given 2-local $(1, n)$-sampler is pairwise independent, to the element distinctness problem. Hence the former problem can be solved in deterministic $O(n \log n)$ time and in zero-error randomized expected linear time.*

2.3 Approximate Completeness Results

The statistical distance between two distributions $D^{(1)}, D^{(2)}$ over the same set is defined as $\left\|D^{(1)} - D^{(2)}\right\| = \max_{\text{events } E} \left|\Pr[D^{(1)} \in E] - \Pr[D^{(2)} \in E]\right|$. We say $D^{(1)}, D^{(2)}$ are c-close if $\left\|D^{(1)} - D^{(2)}\right\| \leq c$, and f-far if $\left\|D^{(1)} - D^{(2)}\right\| \geq f$.

We prove that for appropriate parameters, approximate versions of the full independence and exchangeability problems are prSZK-complete (for arbitrary circuit samplers). We do not reproduce the original definition of prSZK, but we make use of the characterization of this class proved by Sahai and Vadhan [15].

▶ **Definition 12.** For functions $0 \leq c(k, n, s) < f(k, n, s) \leq 1$, $\textsc{Fully-Independent}^{c,f}$ is the following promise problem.[1]

$$\textsc{Fully-Independent}^{c,f}_{\text{YES}} = \left\{S \;:\; \begin{array}{l} S \text{ is a } (k, n, s)\text{-sampler and } S(U) \text{ is } c(k, n, s)\text{-close} \\ \text{to some fully independent distribution over } (\{0,1\}^k)^n \end{array}\right\}$$

$$\textsc{Fully-Independent}^{c,f}_{\text{NO}} = \left\{S \;:\; \begin{array}{l} S \text{ is a } (k, n, s)\text{-sampler and } S(U) \text{ is } f(k, n, s)\text{-far} \\ \text{from every fully independent distribution over } (\{0,1\}^k)^n \end{array}\right\}$$

$\textsc{Exchangeable}^{c,f}$ is defined in an analogous way.

▶ **Theorem 13.** $\textsc{Fully-Independent}^{c,f}$ *is* prSZK*-hard for all constants* $0 < c < f < \frac{1}{4}$.

▶ **Theorem 14.** $\textsc{Fully-Independent}^{c,f} \in$ prSZK *where* $c = c'/(n+1)$*, for all constants* $0 < c' < f^2 < 1$.

▶ **Theorem 15.** $\textsc{Exchangeable}^{c,f}$ *is* prSZK*-hard for all constants* $0 < c < f < \frac{1}{2}$.

▶ **Theorem 16.** $\textsc{Exchangeable}^{c,f} \in$ prSZK *for all constants* $0 < 2c < f^2 < 1$.

Consequently for example $\textsc{Fully-Independent}^{0.05/(n+1),\ 0.24}$ and $\textsc{Exchangeable}^{0.12,\ 0.49}$ are prSZK-complete.

[1] The superscripts have a different meaning than the superscripts in Definition 4.

3 Proofs of Exact Completeness Results

3.1 The Key Lemma

The following is the key lemma in the proof of Theorem 6. It can be interpreted qualitatively as a certain type of amplification.

▶ **Lemma 17.** *There is an algorithm that takes as input a $(1,1,s)$-sampler S and an integer $n \geq 2$, runs in time $O(n+s)$, and outputs a $(1,n,O(n+s))$-sampler T such that the following both hold.*

$$\begin{aligned} S(U) \text{ is uniform} &\implies T(U) \text{ is uniform} \\ S(U) \text{ is nonuniform} &\implies T(U) \text{ is discordant} \end{aligned}$$

Proof. Let T perform the following computation.

```
run S and let b be its output
choose bits a_1, a_2, ..., a_n uniformly at random
if there exists an ℓ < n such that a_ℓ = 0 then
    let ℓ* be the least such ℓ
    output a_1, ..., a_{ℓ*}, b, a_{ℓ*+2}, ..., a_n
else output a_1, ..., a_n
```

It is straightforward to see that if $S(U)$ is uniform then $T(U)$ is uniform. Now suppose $S(U)$ is nonuniform, say $\Pr[S(U)=1] = p \neq \frac{1}{2}$. For brevity we define $D = T(U)$. Consider any two coordinates D_i and D_j where $i < j$. For technical reasons in the analysis below, if ℓ^* does not exist then we define ℓ^* to be an arbitrary value $> n$.

We first show that D_i and D_j are not identically distributed. If $i > 1$ then

$$\begin{aligned} &\Pr[D_i = 1] \\ &= \Pr\big[D_i = 1 \;\big|\; \ell^* = i-1\big] \cdot \Pr[\ell^* = i-1] \;+\; \Pr\big[D_i = 1 \;\big|\; \ell^* \neq i-1\big] \cdot \Pr[\ell^* \neq i-1] \\ &= p \cdot \tfrac{1}{2^{i-1}} \;+\; \tfrac{1}{2} \cdot \big(1 - \tfrac{1}{2^{i-1}}\big). \end{aligned}$$

Similarly, $\Pr[D_j = 1] = p \cdot \frac{1}{2^{j-1}} + \frac{1}{2} \cdot \big(1 - \frac{1}{2^{j-1}}\big)$. Since $p \neq \frac{1}{2}$, and since $\Pr[D_i = 1]$ and $\Pr[D_j = 1]$ are different convex combinations of p and $\frac{1}{2}$, that means they are not equal. More formally,

$$\Pr[D_i = 1] - \Pr[D_j = 1] \;=\; \big(p - \tfrac{1}{2}\big)\big(\tfrac{1}{2^{i-1}} - \tfrac{1}{2^{j-1}}\big) \;\neq\; 0.$$

On the other hand, suppose $i = 1$. Then $\Pr[D_i = 1] = \frac{1}{2}$, and $\Pr[D_j = 1]$ is a nontrivial convex combination of p and $\frac{1}{2}$ and is thus not equal to $\Pr[D_i = 1]$. In either case, D_i and D_j are not identically distributed.

Now we show that D_i and D_j are correlated. Suppose $j = i+1$. Then $\Pr\big[D_j = 1 \;\big|\; D_i = 1\big] = \frac{1}{2}$, and

$$\begin{aligned} \Pr\big[D_j = 1 \;\big|\; D_i = 0\big] \;&=\; \Pr\big[D_j = 1 \;\big|\; \ell^* = i,\; D_i = 0\big] \cdot \Pr\big[\ell^* = i \;\big|\; D_i = 0\big] + \\ &\quad\; \Pr\big[D_j = 1 \;\big|\; \ell^* < i,\; D_i = 0\big] \cdot \Pr\big[\ell^* < i \;\big|\; D_i = 0\big] \\ &= p \cdot \Pr\big[\ell^* = i \;\big|\; D_i = 0\big] + \tfrac{1}{2} \cdot \big(1 - \Pr\big[\ell^* = i \;\big|\; D_i = 0\big]\big). \end{aligned}$$

(Technically $\Pr[D_j = 1 \mid \ell^* < i,\ D_i = 0]$ is undefined if $i = 1$, but then $1 - \Pr[\ell^* = i \mid D_i = 0] = 0$ anyway so the final equation above still holds.) It follows that

$$\Pr[D_j = 1 \mid D_i = 0] - \Pr[D_j = 1 \mid D_i = 1] \;=\; \left(p - \tfrac{1}{2}\right) \cdot \Pr[\ell^* = i \mid D_i = 0] \;\neq\; 0$$

since $p \neq \frac{1}{2}$ and $\Pr[\ell^* = i \mid D_i = 0] > 0$. On the other hand, suppose $j > i + 1$. Then $\Pr[D_j = 1 \mid D_i = 0] = \frac{1}{2}$, and

$$\begin{aligned}\Pr[D_j = 1 \mid D_i = 1] \;&=\; \Pr[D_j = 1 \mid \ell^* = j-1,\ D_i = 1] \cdot \Pr[\ell^* = j-1 \mid D_i = 1] + \\ &\quad\ \Pr[D_j = 1 \mid \ell^* \neq j-1,\ D_i = 1] \cdot \Pr[\ell^* \neq j-1 \mid D_i = 1] \\ &= p \cdot \Pr[\ell^* = j-1 \mid D_i = 1] + \tfrac{1}{2} \cdot \left(1 - \Pr[\ell^* = j-1 \mid D_i = 1]\right).\end{aligned}$$

It follows that

$$\Pr[D_j = 1 \mid D_i = 1] - \Pr[D_j = 1 \mid D_i = 0] \;=\; \left(p - \tfrac{1}{2}\right) \cdot \Pr[\ell^* = j-1 \mid D_i = 1] \;\neq\; 0$$

since $p \neq \frac{1}{2}$ and $\Pr[\ell^* = j-1 \mid D_i = 1] > 0$. In either case, D_i and D_j are correlated since $\Pr[D_j = 1 \mid D_i = 0] \neq \Pr[D_j = 1 \mid D_i = 1]$. ◀

▶ **Lemma 18.** *Lemma 17 holds even when T is required to be an AC^0-type circuit with depth 3 and top fan-in 2, except that the size of T and the running time of the algorithm both become $O(n^2 + s)$.*

Proof. The construction and analysis are the same as in the proof of Lemma 17, but we need more care in implementing T. First, we use a standard reduction to convert S into a 3-CNF F that accepts the same number of inputs as S (but has more input bits). Thus, for some polynomially large q, S accepts a uniformly random input with probability $\frac{1}{2}$ iff F accepts a uniformly random input with probability $\frac{1}{2^q}$. Let $x_1, x_2, \ldots, x_r$ denote the input bits of F. Construct a new CNF F' with input bits $x_0, x_1, \ldots, x_r$ by taking F and including $\overline{x_0}$ in each of the clauses (yielding a 4-CNF), then adding a new clause $(x_0 \vee x_1 \vee \cdots \vee x_q)$. Since

$$\Pr[F' \text{ accepts}] \;=\; \tfrac{1}{2} \cdot \Pr[F \text{ accepts}] + \tfrac{1}{2} \cdot \Pr[(x_1 \vee \cdots \vee x_q) \text{ accepts}]$$

it follows that F accepts with probability $\frac{1}{2^q}$ iff F' accepts with probability $\frac{1}{2}$. Now to implement T, we include a copy of F' as well as the random input bits $a_1, a_2, \ldots, a_n$. The 1^{st} output bit of T is just a_1. For the i^{th} output bit when $i > 1$, we have a multiplexer that selects the output of F' if $(a_1 \wedge a_2 \wedge \cdots \wedge a_{i-2} \wedge \overline{a_{i-1}})$ is true, and selects a_i otherwise. Overall, T is an OR-AND-OR circuit (with negations pushed to the inputs) where each output gate has fan-in at most 2. ◀

3.2 prC$_=$P-Hardness

▶ **Corollary 19.** *Lemmas 17 and 18 also hold when the algorithm is additionally given an integer $k \geq 1$ and is required to output a (k, n)-sampler T, except that the size of T and the running time of the algorithm both become $O(kn + s)$ (for Lemma 17) or $O(kn + n^2 + s)$ (for Lemma 18).*

See the full version for the straightforward proof of Corollary 19.

Proof of Theorem 6. We reduce Uniform-Bit to Panoptic-Stats$^{\kappa,\nu,\sigma}$. Let c be the constant factor in the big O in Corollary 19. Given a $(1, 1, s)$-sampler S, we first find the smallest m such that $c \cdot \left(\kappa(m)\nu(m) + s\right) \leq \sigma(m)$. Such an m exists and is $O(s)$ because

$\kappa\nu \le o(\sigma)$ and $\sigma(m) \ge m$ for all m. Then we run the algorithm from Corollary 19 (based on Lemma 17) with $k = \kappa(m)$ and $n = \nu(m)$ to get T of size at most $c \cdot \big(\kappa(m)\nu(m) + s\big) \le \sigma(m)$. Thus the following both hold.

$$S \in \textsc{Uniform-Bit}_{\text{YES}} \implies T \in \textsc{Panoptic-Stats}^{\kappa,\nu,\sigma}_{\text{YES}}$$
$$S \in \textsc{Uniform-Bit}_{\text{NO}} \implies T \in \textsc{Panoptic-Stats}^{\kappa,\nu,\sigma}_{\text{NO}}$$

The reduction's running time is polynomial since $m, \kappa(m), \nu(m), \sigma(m)$ are all polynomially bounded in s and computable in time polynomial in s, and since the algorithm from Corollary 19 runs in time $O(kn + s)$. ◀

The proof of Theorem 7 is similar; see the full version for details.

3.3 Containment in $\mathrm{C_{=}P}$

In the proof of Theorem 8 we use the following lemma, which states that $\mathrm{C_{=}P}$ is closed under exponential conjunctions and polynomial disjunctions. We supply a folklore proof of this lemma in the full version.

▶ **Lemma 20.** *If $L \in \mathrm{C_{=}P}$ then both of the following hold.*

- $\forall_q L \in \mathrm{C_{=}P}$ *for every polynomial q, where* $\forall_q L = \big\{x \ : \ (x,y) \in L \text{ for all } y \in \{0,1\}^{q(|x|)}\big\}$.
- $\vee L \in \mathrm{C_{=}P}$ *where* $\vee L = \big\{(x_1, \ldots, x_\ell) \ : \ x_i \in L \text{ for some } i\big\}$.

Proof of Theorem 8. The arguments are very similar, so we just give three representative examples: Fully-Independent, K-Wise-Exchangeable, and 2-Exists-Uncorrelated. First we mention a useful tool: If S_1, S_2 are $(1,1)$-samplers, then we define $\mathrm{Equ}(S_1, S_2)$ to be a $(1,1)$-sampler that picks $i \in \{1,2\}$ uniformly at random, runs S_i, and negates the output if $i = 2$. Hence $\mathrm{Equ}(S_1, S_2)(U)$ is uniform iff $S_1(U), S_2(U)$ are identically distributed.

Now we prove that Fully-Independent $\in \mathrm{C_{=}P}$. Note that Fully-Independent $= \forall_q L$ where, if we view S as (say) a (k,n)-sampler, and y as (an appropriately encoded description of) an element of $(\{0,1\}^k)^n$ (so q is linear in the size of S), then

$$(S,y) \in L \iff \Pr[S(U) = y] = \textstyle\prod_{i=1}^{n} \Pr[S(U)_i = y_i].$$

Thus by Lemma 20 it suffices to show that $L \in \mathrm{C_{=}P}$. A reduction from L to Uniform-Bit just outputs $\mathrm{Equ}(S_1, S_2)$, where S_1 runs S and accepts iff the output is y, and S_2 runs S for n times and accepts iff for all i, the i^{th} coordinate of the output of the i^{th} run is y_i.

Now we prove that K-Wise-Exchangeable $\in \mathrm{C_{=}P}$. Note that K-Wise-Exchangeable $= \forall_q L$ where, if we view S as (say) a (k,n)-sampler, and $y = (I, \pi, w)$ as (an appropriately encoded description of) a subset $I \subseteq \{1, \ldots, n\}$ of size $\min(K,n)$, a permutation π on $\{1, \ldots, \min(K,n)\}$, and an element $w \in (\{0,1\}^k)^{\min(K,n)}$ (so q is certainly polynomial in the size of S), then

$$\big(S, (I, \pi, w)\big) \in L \iff \Pr[S(U)_I = w] = \Pr[S(U)_I = \pi(w)]$$

where $S(U)_I$ is the restriction to coordinates indexed by I, and $\pi(w) \in (\{0,1\}^k)^{\min(K,n)}$ is obtained by permuting the coordinates of w by π. Thus by Lemma 20 it suffices to show that $L \in \mathrm{C_{=}P}$. A reduction from L to Uniform-Bit just outputs $\mathrm{Equ}(S_1, S_2)$, where S_1 runs S and accepts iff the output restricted to I is w, and S_2 runs S and accepts iff the output restricted to I is $\pi(w)$.

Now we prove that 2-Exists-Uncorrelated $\in \mathrm{C_{=}P}$. Note that if we define the language $L = \big\{(S,i,j) \ : \ S(U)_i \text{ and } S(U)_j \text{ are uncorrelated}\big\}$, then 2-Exists-Uncorrelated

reduces to $\vee L$ by mapping a (k,n)-sampler S to $\big((S,1,2),(S,1,3),(S,1,4),\ldots,(S,n-1,n)\big)$. Thus by Lemma 20 it suffices to show that $L \in \mathrm{C}_{=}\mathrm{P}$. A reduction from L to Uniform-Bit just outputs $\mathrm{Equ}(S_1,S_2)$, where S_1 runs S yielding some $y \in (\{0,1\}^k)^n$ and accepts with probability $\frac{1}{2^{2k}} \cdot y_i \cdot y_j$ so that $\Pr[S_1(U)=1] = \frac{1}{2^{2k}} \cdot \mathrm{E}\big(S(U)_i \cdot S(U)_j\big)$, and S_2 runs S twice (independently) yielding some $y^{(1)}$ and $y^{(2)}$ and accepts with probability $\frac{1}{2^{2k}} \cdot y_i^{(1)} \cdot y_j^{(2)}$ so that $\Pr[S_2(U)=1] = \frac{1}{2^{2k}} \cdot \mathrm{E}(S(U)_i) \cdot \mathrm{E}(S(U)_j)$. ◀

4 Proofs of Exact Algorithmic Results

We prove Theorem 9 in Section 4.1. The proof of Theorem 10 (on exchangeability of distributions with 2-local samplers) is much more interesting and less elementary, but due to space constraints we must defer it to the full version (where we also prove Theorem 11).

First we introduce some terminology to describe 2-local samplers. Each output bit depends on either zero, one, or two input bits. Output bits that depend on zero input bits are constants (0 or 1). The nonconstant output bits can be modeled with an undirected graph (multi-edges and self-loops allowed) as follows. The input bits are the nodes. Each output bit depending on one input bit is a self-loop, labeled with a function from $\{0,1\}$ to $\{0,1\}$ (either the identity or negation). Each output bit depending on two input bits is an edge between those two nodes, labeled with a function from $\{0,1\}^2$ to $\{0,1\}$. There are three types of such functions that depend on both bits: AND-type (accepting one of the four inputs), XOR-type (accepting two of the four inputs), and OR-type (accepting three of the four inputs).

4.1 Full Independence for 2-Local Samplers

We prove Theorem 9. Consider a 2-local $(1,n)$-sampler S, and assume without loss of generality that S has no constant output bits. We claim that $S(U)$ is fully independent iff both of the following conditions hold.

- **(i)** The graph is a forest, ignoring self-loops.
- **(ii)** Each connected component of the graph has at most one of the following: a self-loop, an AND-type edge, or an OR-type edge.

It is trivial to check in linear time whether these conditions hold.

First we assume that (i) and (ii) both hold, and show that $S(U)$ is fully independent. The different connected components of the graph are certainly fully independent of each other, so we can focus on showing that the coordinates of a single connected component are fully independent. If there is a self-loop, an AND-type edge, or an OR-type edge in the connected component, then let e be that edge. Otherwise, let e be any edge in the connected component. We show that conditioned on e evaluating to any particular bit, the joint distribution of the remaining edges in e's connected component is uniform. This implies that the whole joint distribution of the connected component is fully independent.

Suppose e is a self-loop at some node v, so we are conditioning on v being some particular bit. Ignoring e itself, we can view e's connected component as a tree rooted at v with only XOR-type edges. After the conditioning, there is a bijection between the set of all assignments of values to the edges (excluding e) and the set of all assignments of values to the nodes (excluding v) in e's connected component: An assignment to nodes (together with the conditioned value of v) determines an assignment to edges. Furthermore, every assignment to edges arises from some assignment to nodes, because for any assignment to edges, we can start at v and work our way downward to the leaves, uniquely specifying the

value of each node in terms of the values of its parent and the edge to its parent. Since the sets have the same size, we have exhibited a bijection between them. This means that conditioned on either value of e, the joint distribution of all the other edges in e's connected component is uniform.

Now suppose $e = \{u, v\}$ is not a self-loop. We show that, in fact, conditioned on any one of the four assignments of values to the pair u, v, the joint distribution of all the other edges in e's connected component is uniform. Removing e results in two new connected components, each of which is a tree of XOR-type edges, one rooted at u and the other rooted at v. Let U denote the set of nodes in u's new connected component excluding u itself, and let V denote the set of nodes in v's new connected component excluding v itself. By the argument from the previous paragraph (when e was a self-loop), a uniformly random assignment to U induces a uniformly random assignment to the edges in u's new connected component, and similarly for V. Since assignments to U and V are chosen independently of each other, this means that the values of all the edges in e's original connected component (except e itself) are jointly uniformly distributed (conditioned on any particular assignment to u, v, and hence conditioned on any particular assignment to e).

Now we prove the converse by assuming that (i) and (ii) do not both hold, and showing that $S(U)$ is not fully independent. Let us refer to self-loops, AND-type edges, and OR-type edges as *non-XOR-type* edges. If (i) and (ii) do not both hold, then at least one of the following conditions holds.

(A) There is a cycle consisting entirely of XOR-type edges.
(B) There is a cycle with exactly one AND-type edge or OR-type edge.
(C) There is a path between two non-XOR-type edges.

Suppose (A) holds. Let e be an edge on the cycle. Then e's marginal distribution is uniform, but conditioning on any particular values of the other edges on the cycle determines whether or not e's endpoints are the same bit as each other, and thus fixes the value of e. Hence $S(U)$ is not fully independent. Suppose (B) holds. Let ℓ denote the number of nodes on the cycle. Then the probability that all edges on the cycle evaluate to 1 must be an integer multiple of $\frac{1}{2^\ell}$ (since they only depend on ℓ input bits), but the product of the marginal probabilities that each edge on the cycle evaluates to 1 must be either $\frac{1}{2^{\ell+1}}$ (if there is an AND-type edge) or $\frac{3}{2^{\ell+1}}$ (if there is an OR-type edge). Hence $S(U)$ is not fully independent. Suppose (C) holds. Without loss of generality, all intermediate edges on the path are XOR-type. Let e_1 and e_2 be the two non-XOR-type edges, which we consider to be part of the path. Let ℓ denote the number of nodes on the path. Then the probability that all edges on the path evaluate to 1 must be an integer multiple of $\frac{1}{2^\ell}$ (since they only depend on ℓ input bits), but the product of the marginal probabilities that each edge on the path evaluates to 1 must be either $\frac{1}{2^{\ell+1}}$ (if neither e_1 nor e_2 is OR-type) or $\frac{3}{2^{\ell+1}}$ (if exactly one of e_1, e_2 is OR-type) or $\frac{9}{2^{\ell+1}}$ (if both e_1 and e_2 are OR-type). Hence $S(U)$ is not fully independent.

5 Proofs of Approximate Completeness Results

Due to space constraints, we defer most of this section to the full version. Here, we just give the argument for Theorem 16, which uses the following lemma.

▶ **Lemma 21.** *Suppose D is a distribution over $(\{0,1\}^k)^n$. If D is c-close to some exchangeable distribution D^*, then D is $2c$-close to the distribution D' obtained by drawing a sample from D then permuting the coordinates according to a uniformly random permutation.*

Proof of Lemma 21. For a multiset $W \subseteq \{0,1\}^k$ of size n, we say that $w \in (\{0,1\}^k)^n$ is an *ordering* of W if the multiset $\{w_i \ : \ i \in \{1, \ldots, n\}\}$ equals W. Let $\mathrm{Ord}(W)$ denote the set

of all orderings of W. Let d_W^{*+} be the sum of $\Pr[D = w] - \Pr[D^* = w]$ over all $w \in \mathrm{Ord}(W)$ such that $\Pr[D = w] - \Pr[D^* = w] > 0$, and let d_W^{*-} be the sum of $\Pr[D^* = w] - \Pr[D = w]$ over all $w \in \mathrm{Ord}(W)$ such that $\Pr[D^* = w] - \Pr[D = w] > 0$. Then we have

$$\begin{aligned} \|D - D^*\| &= \tfrac{1}{2} \cdot \textstyle\sum_{w \in (\{0,1\}^k)^n} \big|\Pr[D = w] - \Pr[D^* = w]\big| \\ &= \tfrac{1}{2} \cdot \textstyle\sum_{\text{multisets } W \subseteq \{0,1\}^k \text{ of size } n} \big(d_W^{*+} + d_W^{*-}\big) \end{aligned} \tag{1}$$

Letting $d_W'^{+}$ and $d_W'^{-}$ be the analogous quantities with D' instead of D^*, we have

$$\|D - D'\| = \tfrac{1}{2} \cdot \textstyle\sum_{\text{multisets } W \subseteq \{0,1\}^k \text{ of size } n} \big(d_W'^{+} + d_W'^{-}\big). \tag{2}$$

Now fix some W. Note that since D^* is exchangeable, all elements of $\mathrm{Ord}(W)$ have the same probability under D^*; call this probability p_W^*. If w is an element of $\mathrm{Ord}(W)$ then permuting the coordinates of w uniformly at random yields a uniformly random element of $\mathrm{Ord}(W)$. Thus all elements of $\mathrm{Ord}(W)$ have the same probability under D', namely

$$p'_W = \tfrac{1}{|\mathrm{Ord}(W)|} \cdot \textstyle\sum_{w \in \mathrm{Ord}(W)} \Pr[D = w].$$

If $p'_W \geq p_W^*$ then $d_W'^{+} \leq d_W^{*+}$ by definition. If $p'_W \leq p_W^*$ then $d_W'^{-} \leq d_W^{*-}$ by definition. We also have

$$\begin{aligned} 0 &= \big(\textstyle\sum_{w \in \mathrm{Ord}(W)} \Pr[D = w]\big) - |\mathrm{Ord}(W)| \cdot p'_W \\ &= \textstyle\sum_{w \in \mathrm{Ord}(W)} \big(\Pr[D = w] - p'_W\big) \\ &= d_W'^{+} - d_W'^{-} \end{aligned}$$

which implies that $d_W'^{+} = d_W'^{-} \leq \max(d_W^{*+}, d_W^{*-})$. Hence $\big(d_W'^{+} + d_W'^{-}\big) \leq 2 \cdot \max(d_W^{*+}, d_W^{*-}) \leq 2 \cdot \big(d_W^{*+} + d_W^{*-}\big)$. Since this holds for all W, we get $\|D - D'\| \leq 2 \cdot \|D - D^*\|$ by Equations (1) and (2). ◀

We mention that the constant factor of 2 in Lemma 21 is tight, by the following example. Suppose $k = 1$, and suppose D is uniformly distributed over a set of n strings, one of which has Hamming weight 1 and the other $n - 1$ of which have Hamming weight $n - 1$. Let D^* be uniformly distributed over the strings of Hamming weight $n - 1$. Note that D^* is exchangeable, and $\|D - D^*\| = \frac{1}{n}$. However, D' has probability $\frac{1}{n^2}$ on each string of Hamming weight 1, and probability $\frac{n-1}{n^2}$ on each string of Hamming weight $n - 1$, and thus $\|D - D'\| = 2(1 - \frac{1}{n}) \cdot \frac{1}{n} = 2(1 - \frac{1}{n}) \cdot \|D - D^*\|$.

To prove Theorem 16, we reduce $\textsc{Exchangeable}^{c,f}$ to the promise problem of deciding whether the distributions of two given samplers are $2c$-close or f-far in statistical distance (from each other), which Sahai and Vadhan [15] proved is in prSZK for all constants $0 < 2c < f^2 < 1$. Given a (k, n)-sampler S with distribution $D = S(U)$, the reduction outputs S and another (k, n)-sampler S' that samples from D' (as in the statement of Lemma 21) by running S then permuting the coordinates uniformly at random. (There is a minor technical issue arising from $n!$ not being a power of 2, but this is not problematic.) If D is c-close to some exchangeable distribution then D, D' are $2c$-close. If D is f-far from every exchangeable distribution then D, D' are f-far since D' is exchangeable.

References

1 David Aldous. More uses of exchangeability: Representations of complex random structures. In *Probability and Mathematical Genetics: Papers in Honour of Sir John Kingman*, pages 35–63. Cambridge University Press, 2010.

2 Noga Alon, Alexandr Andoni, Tali Kaufman, Kevin Matulef, Ronitt Rubinfeld, and Ning Xie. Testing k-wise and almost k-wise independence. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, pages 496–505, 2007.

3 Tuğkan Batu, Eldar Fischer, Lance Fortnow, Ravi Kumar, Ronitt Rubinfeld, and Patrick White. Testing random variables for independence and identity. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 442–451, 2001.

4 Tuğkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren Smith, and Patrick White. Testing closeness of discrete distributions. *Journal of the ACM*, 4, 2013.

5 Tuğkan Batu, Ravi Kumar, and Ronitt Rubinfeld. Sublinear algorithms for testing monotone and unimodal distributions. In *Proceedings of the 36th ACM Symposium on Theory of Computing*, pages 381–390, 2004.

6 Christopher Beck, Russell Impagliazzo, and Shachar Lovett. Large deviation bounds for decision trees and sampling lower bounds for AC^0-circuits. In *Proceedings of the 53rd IEEE Symposium on Foundations of Computer Science*, pages 101–110, 2012.

7 Siu On Chan, Ilias Diakonikolas, Gregory Valiant, and Paul Valiant. Optimal algorithms for testing closeness of discrete distributions. *CoRR*, abs/1308.3946, 2013.

8 Anindya De and Thomas Watson. Extractors and lower bounds for locally samplable sources. *ACM Transactions on Computation Theory*, 4(1), 2012.

9 Stephen Fenner, Frederic Green, Steven Homer, and Randall Pruim. Quantum NP is hard for PH. In *Proceedings of the 6th Italian Conference on Theoretical Computer Science*, pages 241–252, 1998.

10 Oded Goldreich, Amit Sahai, and Salil Vadhan. Can statistical zero knowledge be made non-interactive? or On the relationship of SZK and NISZK. In *Proceedings of the 19th International Cryptology Conference*, pages 467–484, 1999.

11 Oded Goldreich and Salil Vadhan. Comparing entropies in statistical zero-knowledge with applications to the structure of SZK. In *Proceedings of the 14th IEEE Conference on Computational Complexity*, page 54–73, 1999.

12 Oded Goldreich and Salil Vadhan. On the complexity of computational problems regarding distributions. *Studies in Complexity and Cryptography*, pages 390–405, 2011.

13 Shachar Lovett and Emanuele Viola. Bounded-depth circuits cannot sample good codes. *Computational Complexity*, 21(2):245–266, 2012.

14 Sofya Raskhodnikova, Dana Ron, Amir Shpilka, and Adam Smith. Strong lower bounds for approximating distribution support size and the distinct elements problem. *SIAM Journal on Computing*, 39(3):813–842, 2009.

15 Amit Sahai and Salil Vadhan. A complete problem for statistical zero knowledge. *Journal of the ACM*, 50(2):196–249, 2003.

16 Jun Tarui. Probabilistic polynomials, AC^0 functions, and the polynomial-time hierarchy. *Theoretical Computer Science*, 113(1):167–183, 1993.

17 Seinosuke Toda and Mitsunori Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 21(2):316–328, 1992.

18 Paul Valiant. Testing symmetric properties of distributions. *SIAM Journal on Computing*, 40(6):1927–1968, 2011.

19 Emanuele Viola. Extractors for circuit sources. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science*, pages 220–229, 2011.

20 Emanuele Viola. The complexity of distributions. *SIAM Journal on Computing*, 41(1):191–218, 2012.

21 Klaus Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986.

22 Thomas Watson. The complexity of estimating min-entropy. Technical Report TR12-070, Electronic Colloquium on Computational Complexity, 2012.

Faster Compact On-Line Lempel-Ziv Factorization

Jun'ichi Yamamoto, Tomohiro I, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda

Department of Informatics, Kyushu University, Nishiku, Fukuoka, Japan
{tomohiro.i,bannai,inenaga,takeda}@inf.kyushu-u.ac.jp

Abstract

We present a new on-line algorithm for computing the Lempel-Ziv factorization of a string that runs in $O(N \log N)$ time and uses only $O(N \log \sigma)$ bits of working space, where N is the length of the string and σ is the size of the alphabet. This is a notable improvement compared to the performance of previous on-line algorithms using the same order of working space but running in either $O(N \log^3 N)$ time (Okanohara & Sadakane 2009) or $O(N \log^2 N)$ time (Starikovskaya 2012). The key to our new algorithm is in the utilization of an elegant but less popular index structure called Directed Acyclic Word Graphs, or DAWGs (Blumer et al. 1985). We also present an opportunistic variant of our algorithm, which, given the run length encoding of size m of a string of length N, computes the Lempel-Ziv factorization of the string on-line, in $O\left(m \cdot \min\left\{\frac{(\log\log m)(\log\log N)}{\log\log\log N}, \sqrt{\frac{\log m}{\log\log m}}\right\}\right)$ time and $O(m \log N)$ bits of space.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Lempel-Ziv Factorization, String Index

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.675

1 Introduction

The Lempel-Ziv (LZ) factorization of a string [20], discovered over 35 years ago, captures important properties concerning repeated occurrences of substrings in the string, and has numerous applications in the field of data compression, compressed full text indices [12], and is also the key component to various efficient algorithms on strings [11, 6]. Therefore, a large amount of work has been devoted to its efficient computation, especially in the *off-line* setting where the text is static, and the LZ factorization can be computed in as fast as $O(N)$ time assuming an integer alphabet, using $O(N \log N)$ bits of space (see [1] for a survey; more recent results are in [14, 10, 7, 9]). In this paper, we consider the more difficult and challenging *on-line* setting, where new characters may be appended to the end of the string. If we may use $O(N \log N)$ bits of space, the problem can be solved in $O(N \log \sigma)$ time where σ is the size of the alphabet, by use of string indices such as suffix trees [19] and on-line algorithms to construct them [18]. However, when σ is small and N is very large (e.g. DNA), the $O(N \log N)$ bits space complexity is much larger than the $N \log \sigma$ bits of the input text, and can be prohibitive. To solve this problem, space efficient on-line algorithms for LZ factorization based on succinct data structures have been proposed. Okanohara and Sadakane [15] gave an algorithm that runs in $O(N \log^3 N)$ time using $N \log \sigma + o(N \log \sigma) + O(N)$ bits of space. Later Starikovskaya [17], achieved $O(N \log^2 N)$ time using $O(N \log \sigma)$ bits of space, assuming $\log_\sigma N$ characters are packed in a machine word. Kärkkäinen et al. [8] proposed an LZ factorization algorithm that works in $O(Ntd)$ time and $N \log \sigma + O(N \log N/d)$ bits of space with $\lceil N/d \rceil$ delay (i.e., it processes $\lceil N/d \rceil$ characters in a batch) for any $d \geq 1$, where t is the time for a rank query on a string over an alphabet of size σ. When $d = \Theta(\log_\sigma N)$, their algorithm runs in $O(Nt \log N / \log \sigma)$



© Jun'ichi Yamamoto, Tomohiro I, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda;
licensed under Creative Commons License CC-BY

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 675–686





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

time and $O(N \log \sigma)$ bits of space. However, the delay then becomes $\Theta(N \log \sigma / \log N)$, which seems to be too large to be called on-line.

In this paper, we propose a new on-line LZ factorization algorithm running in $O(N \log N)$ time using $O(N \log \sigma)$ bits of space, which is a notable improvement compared to the run-times of the previous on-line algorithms while still keeping the working space within a constant factor of the input text. Our algorithm is based on a novel application of a full text index called Directed Acyclic Word Graphs, or DAWGs [4], which, despite its elegance, has not received as much attention as suffix trees. To achieve a more efficient algorithm, we exploit an interesting feature of the DAWG structure that, unlike suffix trees, allows us to collect information concerning the left context of strings into each state in an efficient and on-line manner. We further show that the DAWG allows for an opportunistic variant of the algorithm which is more time and space efficient when the run length encoding (RLE) of the string is small. Given the RLE of size $m \leq N$ of the string, our on-line algorithm runs in $O\left(m \cdot \min\left\{\frac{(\log\log m)(\log\log N)}{\log\log\log N}, \sqrt{\frac{\log m}{\log\log m}}\right\}\right) = o(m \log m)$ time using $O(m \log N)$ bits of space.

2 Preliminaries

Let $\Sigma = \{1, \ldots, \sigma\}$ be a finite integer *alphabet*. An element of Σ^* is called a *string*. The length of a string S is denoted by $|S|$. The empty string ε is the string of length 0. Let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. For a string $S = XYZ$, X, Y and Z are called a *prefix*, *substring*, and *suffix* of S, respectively. The set of prefixes and substrings of S are denoted by $\mathit{Prefix}(S)$ and $\mathit{Substr}(S)$, respectively. The *longest common prefix* (lcp) of strings X, Y is the longest string in $\mathit{Prefix}(X) \cap \mathit{Prefix}(Y)$. The i-th character of a string S is denoted by $S[i]$ for $1 \leq i \leq |S|$, and the substring of a string S that begins at position i and ends at position j is denoted by $S[i..j]$ for $1 \leq i \leq j \leq |S|$. For convenience, let $S[i..j] = \varepsilon$ if $j < i$. A position i is called an *occurrence* of X in S if $S[i..i + |X| - 1] = X$. For any string $S = S[1..N]$, let $S^{rev} = S[N] \cdots S[1]$ denote the reversed string. For any character $a \in \Sigma$ and integer $i \geq 0$, let $a^0 = \varepsilon$, $a^i = a^{i-1}a$. We call i the *exponent* of a^i.

The default base of logarithms will be 2. Our model of computation is the unit cost word RAM with the machine word size at least $\log N$ bits. For an input string S of length N, let $r = \log_\sigma N = \frac{\log N}{\log \sigma}$. For simplicity, assume that $\log N$ is divisible by $\log \sigma$, and that N is divisible by r. A string of length r, called a *meta-character*, consists of $\log N$ bits, and therefore fits in a single machine word. Thus, a meta-character can also be transparently regarded as an element in the integer alphabet $\Sigma^r = \{1, \ldots, N\}$. We assume that given $1 \leq i \leq N - r + 1$, any meta-character $A = S[i..i + r - 1]$ can be retrieved in constant time. Also, we can pre-compute an array of size $2^{\frac{\log N}{2}}$ occupying $O(\sqrt{N} \log N) = o(N)$ bits in $o(N)$ time, so A^{rev} can be computed in constant time given A. We call a string on the alphabet Σ^r of meta-characters, a *meta-string*. Any string S whose length is divisible by r can be viewed as a meta-string S of length $n = \frac{|S|}{r}$. We write $\langle S \rangle$ when we explicitly view string S as a meta-string, where $\langle S \rangle[j] = S[(j-1)r + 1..jr]$ for each $j \in [1, n]$. Such range $[(j-1)r + 1, jr]$ of positions will be called *meta-blocks* and the beginning positions $(j-1)r + 1$ of meta-blocks will be called *block borders*. For clarity, the length n of a meta-string $\langle S \rangle$ will be denoted by $\|\langle S \rangle\|$. Meta-strings are sometimes called *packed* strings. Note that $n \log N = N \log \sigma$.

2.1 LZ Factorization

There are several variants of LZ factorization, and as in most recent work, we consider the variant also called s-factorization [5]. The s-factorization of a string S is the factorization

$S = f_1 \cdots f_z$ where each s-factor $f_i \in \Sigma^+$ $(i = 1, \ldots, z)$ is defined as follows: $f_1 = S[1]$. For $i \geq 2$: if $S[|f_1 \cdots f_{i-1}| + 1] = c \in \Sigma$ does not occur in $f_1 \cdots f_{i-1}$, then $f_i = c$. Otherwise, f_i is the longest prefix of $f_i \cdots f_z$ that occurs at least twice in $f_1 \cdots f_i$. Notice that self-referencing is allowed, i.e., the previous occurrence of f_i may overlap with itself. Each s-factor can be represented in a constant number of words, i.e., either as a single character or a pair of integers representing the position of a previous occurrence of the factor and its length.

2.2 Tools

Let B be a bit array of length N. For any position x of B, let $rank(B, x)$ denote the number of 1's in $B[1..x]$. For any integer j, let $select(B, j)$ denote the position of the jth 1 in B. For any pair of positions x, y $(x \leq y)$ of B, the number of 1's in $B[x..y]$ can be expressed as $pc(B, x, y) = rank(B, y) - rank(B, x - 1)$. It is possible to maintain B and support rank/select queries and bit flip operations in $O(\log N)$ time, using $N + o(N)$ bits of space (e.g. Raman et al. [16]).

Directed Acyclic Word Graphs (DAWG) are a variant of suffix indices, similar to suffix trees or suffix arrays. The DAWG of a string S is the smallest partial deterministic finite automaton that accepts all suffixes of S. Thus, an arbitrary string is a substring of S iff it can be traversed from the source of the DAWG. While each edge of the suffix tree corresponds to a substring of S, an edge of a DAWG corresponds to a single character.

▶ **Theorem 1** (Blumer et al. [4]). *The numbers of states, edges and suffix links of the DAWG of string S are $O(|S|)$, independent of the alphabet size σ. The DAWG augmented with the suffix links can be constructed in an on-line manner in $O(|S| \log \sigma)$ time using $O(|S| \log |S|)$ bits of space.*

We give a more formal presentation of DAWGs below. Let $EndPos_S(u) = \{j \mid u = S[i..j], 1 \leq i \leq j \leq N\}$. Define an equivalence relation on $Substr(S)$ such that for any $u, w \in Substr(S)$, $u \equiv_S w \iff EndPos_S(u) = EndPos_S(w)$, and denote the equivalence class of $u \in Substr(S)$ as $[u]_S$. When clear from the context, we abbreviate the above notations as $EndPos$, $\equiv$ and $[u]$, respectively. Note that for any two elements in $[u]$, one is a suffix of the other. We denote by $\overleftarrow{u}$ the longest member of $[u]$. The states V and edges E of the DAWG can be characterized as $V = \{[u] \mid u \in Substr(S)\}$ and $E = \{([u], a, [ua]) \mid u, ua \in Substr(S), u \not\equiv ua\}$. We also define the set G of labeled reversed edges, called *suffix links*, by $G = \{([au], a, [u]) \mid u, au \in Substr(S), u = \overleftarrow{u}\}$. An edge $([u], a, [ua]) \in E$ is called a *primary* edge if $|\overleftarrow{u}| + 1 = |\overleftarrow{ua}|$, and a *secondary* edge otherwise. We call $[ua]$ a primary (resp. secondary) child of $[u]$ if the edge is primary (resp. secondary). By storing $|\overleftarrow{u}|$ at each state $[u]$, we can determine whether an edge $([u], a, [ua])$ is primary or secondary in $O(1)$ time using $O(|S| \log |S|)$ bits of total space.

Whenever a state $[u]$ is created during the on-line construction of the DAWG, it is possible to assign the position $pos_{[u]} = \min EndPos_S(u)$ to that state. If state u is reached by traversing the DAWG from the source with string p, this means that $p = S[pos_{[u]} - |p| + 1..pos_{[u]}]$, and thus the first occurrence $pos_{[u]} - |p| + 1$ of p can be retrieved, using $O(|S| \log |S|)$ bits of total space.

For any set P of points on a 2-D plane, consider query $find_any(P, I_h, I_t)$ which returns an *arbitrary* element in P that is contained in a given orthogonal range $I_h \times I_t$ if such exists, and returns **nil** otherwise. A simple corollary of the following result by Blelloch [3]:

▶ **Theorem 2** (Blelloch [3]). *The 2D dynamic orthogonal range reporting problem on n elements can be solved using $O(n \log n)$ bits of space so that insertions and deletions take*

$O(\log n)$ amortized time and range reporting queries take $O(\log n + k \log n / \log\log n)$ time, where k is the number of output elements.

is that the query *find_any*(P, I_h, I_t) can be answered in $O(\log n)$ time on a dynamic set P of points. It is also possible to extend the *find_any* query to return, in $O(\log n)$ time, a constant number of elements contained in the range.

3 On-line LZ Factorization with Packed Strings

The problem setting and high-level structure of our algorithm follows that of Starikovskaya [17], but we employ somewhat different tools. The goal of this section is to prove the following theorem.

▶ **Theorem 3.** *The s-factorization of any string $S \in \Sigma^*$ of length N can be computed in an on-line manner in $O(N \log N)$ time and $O(N \log \sigma)$ bits of space.*

By on-line, we assume that the input string S is given r characters at a time, and we are to compute the s-factorization of the string $S[1..jr]$ for all $j = 1, \ldots, n$. Since only the last factor can change for each j, the whole s-factorization need not be re-calculated so we will focus on describing how to compute each s-factor f_i by extending f_i while a previous occurrence exists. We show how to maintain dynamic data structures using $O(N \log \sigma)$ bits in $O(N \log N)$ total time that allow us to (1) determine whether $|f_i| < r$ in $O(1)$ time, and if so, compute f_i in $O(|f_i| \log N)$ time (Lemma 4), (2) compute f_i in $O(|f_i| \log N)$ time when $|f_i| \geq r$ (Lemma 9), and (3) retrieve a previous occurrence of f_i in $O(|f_i| \log N)$ time (Lemma 11). Since $\sum_{i=1}^{z} |f_i| = N$, these three lemmas prove Theorem 3.

The difference between our algorithm and that of Starikovskaya [17] can be summarized as follows: For (1), we show that a dynamic succinct bit-array that supports rank/select queries and flip operations can be used, as opposed to a suffix trie employed in [17]. This allows our algorithm to use a larger meta-character size of $r = \log_\sigma N$ instead of $\frac{\log_\sigma N}{4}$ in [17], where the $1/4$ factor was required to keep the size of the suffix trie within $O(N \log \sigma)$ bits. Hence, our algorithm can pack characters more efficiently into a word. For (2), we show that by using a DAWG on the meta-string of length $n = N/r$ which occupies only $O(N \log \sigma)$ bits, we can reduce the problem of finding valid extensions of a factor to dynamic orthogonal range reporting queries, for which a space efficient dynamic data structure with $O(\log n)$ time query and update exists [3]. In contrast, Starikovskaya's algorithm uses a suffix tree on the meta-string and dynamic wavelet trees requiring $O(\log^2 n)$ time for queries and updates, which is the bottleneck of her algorithm. For (3), we develop a technique for the case $|f_i| < r$, which may be of independent interest.

In what follows, let $l_i = \sum_{k=1}^{i-1} |f_k|$, i.e., l_i is the total length of the first $i-1$ s-factors. Although our presentation assumes that N is known, this can be relaxed at the cost of a constant factor by simply restarting the entire algorithm when the length of the input string doubles.

3.1 Algorithm for $|f_i| < r$

Consider a bit array $M_k[1..N]$. For any meta-character $A \in \Sigma^r$, let $M_k[A] = 1$ iff $S[l{+}1..l{+}r] = A$ for some $0 \leq l \leq k - r$, i.e., $M_k[A]$ indicates whether A occurs as a substring in $S[1..k]$. We will dynamically maintain a single bit array representing M_k, for increasing values of k. For any short string t ($|t| < r$), let D_t and U_t be, respectively, the lexicographically smallest

and largest meta-characters having t as a prefix, namely, the bit-representation[1] of D_t is the concatenation of the bit-representation of t and $0^{(r-|t|)\log\sigma}$, and the bit-representation of U_t is the concatenation of the bit-representation of t and $1^{(r-|t|)\log\sigma}$. These representations can be obtained from t in constant time using standard bit operations. Then, the set of meta-characters that have t as a prefix can be represented by the interval $tr(t) = [D_t, U_t]$. It holds that t occurs in $S[1..k-r+|t|]$ iff some element in $M_k[D_t..U_t]$ is 1, i.e. $pc(M_k, D_t, U_t) > 0$. Therefore, we can check whether or not a string of length up to r occurs at some position $p \leq l_i$ by using M_{l_i+r-1}.

For any $0 \leq m \leq r$, let $t_m = S[l_i + 1..l_i + m]$. We have that $|f_i| < r$ iff $M_{l_i+r-1}[t_r] = 0$, which can be determined in $O(1)$ time. Assume $|f_i| < r$, and let $m_i = \max\{m \mid 0 \leq m < r, pc(M_{l_i+r-1}, D_{t_m}, U_{t_m}) > 0\}$, where $m_i = 0$ indicates that $S[l_i + 1]$ does not occur in $S[1..l_i]$. From the definition of s-factorization, we have that $|f_i| = \max(1, m_i)$. Notice that m_i can be computed by $O(|f_i|)$ rank queries on M_{l_i+r-1}, due to the monotonicity of $pc(M_{l_i+r-1}, D_{t_m}, U_{t_m})$ for increasing values of m. To maintain M_k we can use rank/select dictionaries for a dynamic bit array of length N (e.g. [16]) mentioned in Section 2. Thus we have:

▶ **Lemma 4.** *We can maintain in $O(N \log N)$ total time, a dynamic data structure occupying $N + o(N)$ bits of space that allows whether or not $|f_i| < r$ to be determined in $O(1)$ time, and if so, f_i to be computed in $O(|f_i| \log N)$ time.*

3.2 Algorithm for $|f_i| \geq r$.

To compute f_i when $|f_i| \geq r$, we use the DAWG for the meta-string $\langle S \rangle$ which we call the *packed DAWG*. While the DAWG for S requires $O(N \log N)$ bits, the packed DAWG only requires $O(N \log \sigma)$ bits. However, the complication is that only substrings with occurrences that start at block borders can be traversed from the source of the packed DAWG. In order to overcome this problem, we will augment the packed DAWG and maintain the set $Points_{[u]} = \{(A^{rev}, X) \mid ([u], X, [uX]) \in E, A\overleftarrow{u}X \in Substr(\langle S \rangle)\}$ for all states $[u]$ of the packed DAWG. A pair $(A^{rev}, X) \in Points_{[u]}$ represents that there exists an occurrence of $A\overleftarrow{u}X$ in $\langle S \rangle$, in other words, the longest element $\overleftarrow{u}$ corresponding to the state can be extended by X and still have an occurrence in $\langle S \rangle$ immediately preceded by A.

▶ **Lemma 5.** *For meta-string $\langle S \rangle$ of length n and its packed DAWG (V, E, G), the the total number of elements in $Points_{[u]}$ for all states $[u] \in V$ is $O(n)$.*

Proof. Consider edge $([u], X, [uX]) \in E$. If $\overleftarrow{u}X \neq \overleftarrow{uX}$, i.e., the edge is secondary, it follows that there exists a unique meta-character $A = \langle S \rangle[pos_{[uX]} - \|\overleftarrow{u}X\|]$ such that $A\overleftarrow{u}X \equiv_{\langle S \rangle} \overleftarrow{u}X$, namely, any occurrence of $\overleftarrow{u}X$ is always preceded by A in $\langle S \rangle$. If $\overleftarrow{u}X = \overleftarrow{uX}$, i.e., the edge is primary, then, for each distinct meta-character A preceding an occurrence of $\overleftarrow{u}X = \overleftarrow{uX}$ in $\langle S \rangle$, there exists a suffix link $([A\overleftarrow{uX}], A, [\overleftarrow{uX}]) \in G$. Therefore, each point (A^{rev}, X) in $Points_{[u]}$ can be associated to either a secondary edge from $[u]$ or one of the incoming suffix links to its primary child $[uX]$. Since each state has a unique longest member, each state has exactly one incoming primary edge. Therefore, the total number of elements in $Points_{[u]}$ for *all* states $[u]$ is equal to the total number of secondary edges and suffix links, which is $O(n)$ due to Theorem 1. ◀

[1] Assume that $0^{\log N}$ and $1^{\log N}$ correspond to meta-characters 1 and N, respectively.

▶ **Lemma 6.** *For string $S \in \Sigma^*$ of length N, we can, in $O(N \log \sigma)$ total time and bits of space and in an on-line manner, construct the packed DAWG (V, E, G) of S as well as maintain $Points_{[u]}$ for all states $[u] \in V$ so that find_any$(Points_{[u]}, I_h, I_t)$ for an orthogonal range $I_v \times I_h$ can be answered in $O(\log n)$ time.*

Proof. It follows from Theorem 1 that the packed DAWG can be computed in an on-line manner, in $O(n \log N) = O(N \log \sigma)$ time and bits of space, since the size of the alphabet for meta-strings is $O(N)$ and the length of the meta-string is $n = \frac{N}{r}$. To maintain and support *find_any* queries on *Points* efficiently, we use the dynamic data structure by Blelloch [3] mentioned in Theorem 2. Thus from Lemma 5, the total space requirement is $O(N \log \sigma)$ bits. Since each insert operation can be performed in amortized $O(\log n)$ time (no elements are deleted in our algorithm), what remains is to show that the total number of insert operations to *Points* is $O(n)$. This is shown below by a careful analysis of the on-line DAWG construction algorithm [4].

Assume we have the packed DAWG for a prefix $u = \langle S \rangle[1..\|u\|]$ of meta-string $\langle S \rangle$. Let $B = \langle S \rangle[\|u\| + 1]$ be the meta-character that follows u in $\langle S \rangle$. We group the updates performed on the packed DAWG when adding B, into the following two operations: (a) the new sink state $[uB]$ is created, and (b) a state is split.

First, consider case (a). Let $u_0 = u$, and consider the sequence $[u_1], \ldots, [u_q]$ of states such that the suffix link of $[u_j]$ points to $[u_{j+1}]$ for $0 \leq j < q$, and $[u_q]$ is the first state in the sequence which has an out-going edge labeled by B. As in [4], we use an auxiliary state $\perp$ and assume that for every meta-character $A \in \Sigma^r$ there is an edge $(\perp, A, [\varepsilon])$ leading to the source $[\varepsilon]$, so that there always exists such state $[u_q]$ in any sequence of suffix links. Note that any element of $[u_{j+1}]$ is a suffix of any element of $[u_j]$. The following operations are performed. (a-1) The primary edge from the old sink $[u]$ to the new sink $[uB]$ is created. No insertion is required for this edge since $[uB]$ has no incoming suffix links. (a-2) For each $1 \leq j < q$ a secondary edge $([u_j], B, [uB])$ is created, and the pair (C_j^{rev}, B) is inserted to $Points_{[u_j]}$, where C_j is the unique meta-character that immediately precedes $\overleftarrow{u_j}B$ in uB, i.e., $C_j = \langle uB \rangle[pos_{[uB]} - \|\overleftarrow{u_j}B\|]$. (a-3) Let $([u_q], B, w)$ be the edge with label B from state $[u_q]$. The suffix link of the new sink state $[uB]$ is created and points to w. Let $e = ([v], B, w)$ be the primary incoming edge to w, and A be the meta-character that labels the suffix link (note that $[v]$ is not necessarily equal to $[u_q]$). We then insert a new pair (A^{rev}, B) into $Points_{[v]}$.

Next, consider case (b). After performing (a), node w is split if the edge $([u_q], B, w)$ is secondary. Let $[v_1] = [v]$, and let $[v_1], \ldots, [v_k]$ be the parents of the state w of the packed DAWG for u, sorted in decreasing order of their longest member. Then, it holds that there is a suffix link from $[v_h]$ to $[v_{h+1}]$ and any element of $[v_{h+1}]$ is a suffix of any element of $[v_h]$ for any $1 \leq h < k$. Assume $\overleftarrow{v_i}B$ is the longest suffix of uB that has another (previous) occurrence in uB. (Namely, $[v_i]$ is equal to the state $[u_q]$ of (a-2) above.) If $i > 1$, then the state w is split into two states $[v_1B]$ and $[v_iB]$ such that $[v_1B] \cup [v_iB] = w$ and any element of $[v_iB]$ is a proper suffix of any element of $[v_1B]$. The following operations are performed. (b-1) The secondary edge from $[v_i]$ to w becomes the primary edge to $[v_iB]$, and for all $i < j \leq k$ the secondary edge from $[v_j]$ to w becomes a secondary edge to $[v_jB]$. The primary and secondary edges from $[v_h]$ to w for all $1 \leq h < i$ become the primary and secondary ones from $[v_h]$ to $[v_1B]$, respectively. Clearly the sets $Points_{[v_h]}$ for all $1 \leq h < i$ are unchanged. Also, since edges $([v_j], B, [v_iB])$ are all secondary, the sets $Points_{[v_j]}$ for all $i < j \leq k$ are unchanged. Moreover, the element of $Points_{[v_i]}$ that was associated to the secondary edge to w, is now associated to the suffix link from $[v_1B]$ to $[v_iB]$. Hence, $Points_{[v_i]}$ is also unchanged. Consequently, there are no updates due to edge redirection. (b-2) All outgoing edges of $[v_1B]$ are copied as outgoing edges of $[v_iB]$. Since any element of

$[v_iB]$ is a suffix of any element of $[v_1B]$, the copied edges are all secondary. Hence, we insert a pair to $Points_{[v_iB]}$ for each secondary edge, accordingly.

Thus, the total number of insert operations to *Points* for all states is linear in the number of update operations during the on-line construction of the packed DAWG, which is $O(n)$ due to [4]. This completes the proof. ◀

For any string f and integer $0 \leq m \leq \min(|f|, r-1)$, let strings $\alpha_m(f)$, $\beta_m(f)$, $\gamma_m(f)$ satisfy $f = \alpha_m(f)\beta_m(f)\gamma_m(f)$, $|\alpha_m(f)| = m$, and $|\beta_m(f)| = j'r$ where $j' = \max\{j \geq 0 \mid m + jr \leq |f|\}$. We say that an occurrence of f in S has offset m $(0 \leq m \leq r-1)$, if, in the occurrence, $\alpha_m(f)$ corresponds to a suffix of a meta-block, $\beta_m(f)$ corresponds to a sequence of meta-blocks (i.e. $\beta_m(f) \in Substr(\langle S \rangle)$), and $\gamma_m(f)$ corresponds to a prefix of a meta-block.

Let f_i^m denote the longest prefix of $S[l_i + 1..N]$ which has a previous occurrence in S with offset m. Thus, $|f_i| = \max_{0 \leq m < r} |f_i^m|$. In order to compute f_i^m, the idea is to find the longest prefix u of meta-string $\langle \beta_m(S[l_i + 1..N]) \rangle$ that can be traversed from the source of the packed DAWG while assuring that at least one previous occurrence of u in $\langle S \rangle$ is immediately preceded by a meta-block that has $\alpha_m(S[l_i + 1..N])$ as a suffix. It follows that $u = \beta_m(f_i^m)$.

▶ **Lemma 7.** *Given the augmented packed DAWG (V, E, G) of Lemma 6 of meta-string $\langle S \rangle$, the longest prefix f of any string P that has an occurrence with offset m in S can be computed in $O(\frac{|f|}{r} \log n + r \log n)$ time.*

Proof. We first traverse the packed DAWG for $\langle S \rangle$ to find $\beta_m(f)$. This traversal is trivial for $m = 0$, so we assume $m > 0$. For any string t $(|t| < r)$, let L_t and R_t be, respectively, the lexicographically smallest and largest meta-character which has t as a suffix, namely, the bit-representation of L_t is the concatenation of $0^{(r-|t|)\log\sigma}$ and the bit-representation of t, and the bit-representation of R_t is the concatenation of $1^{(r-|t|)\log\sigma}$ and the bit-representation of t. Then, the set of meta-characters that have t^{rev} as a prefix, (or, t as a suffix when reversed), can be represented by the interval $hr(t) = [L_t^{rev}, R_t^{rev}]$. Suppose we have successfully traversed the packed DAWG with the prefix $u = \langle \beta_m(P) \rangle [1..\|u\|]$ and want to traverse with the next meta-character $X = \langle \beta_m(P) \rangle [\|u\| + 1]$. If $u = \overleftarrow{u}$, i.e. only primary edges were traversed, then there exists an occurrence of $\alpha_m(P)uX$ with offset m in string S iff $find_any(Points_{[u]}, hr(\alpha_m(P)), [X, X]) \neq \textbf{nil}$. Otherwise, if $u \neq \overleftarrow{u}$, all occurrences of u (and thus all extensions of u that can be traversed) in $\langle S \rangle$ is already guaranteed to be immediately preceded by the unique meta-character $A = \langle S \rangle [pos_{[u]} - \|u\|]$ such that $A^{rev} \in hr(\alpha_m(P))$. Thus, there exists an occurrence of $\alpha_m(P)uX$ with offset m in string S iff $([u], X, [uX]) \in E$. We extend u until *find_any* returns **nil** or no edge is found, at which point we have $\alpha_m(P)u = \alpha_m(f)\beta_m(f)$.

Now, $\gamma_m(f)$ is a prefix of meta-character $B = \langle \beta_m(P) \rangle [\|\langle u \rangle\| + 1]$. When $u = \overleftarrow{u}$, we can compute $\gamma_m(f)$ by asking $find_any(Points_{[u]}, hr(\alpha_m(P)), tr(B[1..j]))$ for $0 \leq j < r$. The maximum j such that *find_any* does not return **nil** gives $|\gamma_m(f)|$. If $u \neq \overleftarrow{u}$, $\gamma_m(f)$ is the longest lcp between B and any outgoing edge from $[u]$. This can be computed in $O(\log n + |\gamma_m(f)|)$ time by maintaining outgoing edges from $[u]$ in balanced binary search trees, and finding the lexicographic predecessor/successor B^-, B^+ of B in these edges, and computing the lcp between them. The lemma follows since each *find_any* query takes $O(\log n)$ time. ◀

From the proof of Lemma 7, $\beta_m(f_i^m)$ can be computed in $O(\frac{|f_i^m|}{r} \log n)$ time, and for all $0 \leq m < r$, this becomes $O(|f_i| \log n)$ time. However, for computing $\gamma_m(f_i^m)$, if we simply

apply the algorithm and use $O(r \log n)$ time for each f_i^m, the total time for all $0 \leq m < r$ would be $O(r^2 \log n)$ which is too large for our goal. Below, we show that all $\gamma_m(f_i^m)$ are not required for computing $\max_{0 \leq m < r} |f_i^m|$, and this time complexity can be reduced.

Consider computing $F_m = \max_{0 \leq x \leq m} |f_i^x|$ for $m = 0, \ldots, r-1$. We first compute $\hat{f}_i^m = \alpha_m(f_i^m)\beta_m(f_i^m)$ using the first part of the proof of Lemma 7. We shall compute $\gamma_m(f_i^m)$ only when F_m can be larger than F_{m-1} i.e., $|\hat{f}_i^m| + |\gamma_m(f_i^m)| > F_{m-1}$. Since $|\gamma_m(f_i^m)| < r$, this will never be the case if $|\hat{f}_i^m| \leq F_{m-1} - r + 1$, and will always be the case if $|\hat{f}_i^m| > F_{m-1}$. For the remaining case, i.e. $0 \leq F_{m-1} - |f_i^m| < r - 1$, $F_m > F_{m-1}$ iff $|\gamma_m(f_i^m)| > F_{m-1} - |\hat{f}_i^m|$. If $u = \overleftarrow{u}$, this can be determined by a single *find_any* query with $j = F_{m-1} - |\hat{f}_i^m| + 1$ in the last part of the proof of Lemma 7, and if so, the rest of $\gamma_m(f_i^m)$ is computed using the *find_any* query for increasing j. When $u \neq \overleftarrow{u}$, whether or not the lcp between B and B^- or B^+ is greater than $F_{m-1} - |\hat{f}_i^m|$ can be checked in constant time using bit operations.

From the above discussion, each *find_any* or predecessor/successor query for computing $\gamma_m(f_i^m)$ updates F_m, or returns **nil**. Therefore, the total time for computing $F_{r-1} = |f_i|$ is $O((r + |f_i|) \log n) = O(|f_i| \log n)$.

A technicality we have not mentioned yet, is when and to what extent the packed DAWG is updated when computing f_i. Let F be the length of the current longest prefix of $S[l_i+1..N]$ with an occurrence less than $l_i + 1$, found so far while computing f_i. A self-referencing occurrence of $S[l_i + 1..l_i + F]$ can reach up to position $l_i + F - 1$. When computing f_i using the packed DAWG, F is increased by at most r characters at a time. Thus, for our algorithm to successfully detect such self-referencing occurrences, the packed DAWG should be built up to the meta-block that includes position $l_i + F - 1 + r$ and updated when F increases. This causes a slight problem when computing f_i^m for some m; we may detect a substring which only has an occurrence larger than l_i during the traversal of the DAWG. However, from the following lemma, the number of such *future* occurrences that update F can be limited to a constant number, namely two, and hence by reporting up to three elements in each *find_any* query that may update F, we can obtain an occurrence less than $l_i + 1$, if one exists. These occurrences can be retrieved in $O(\log N)$ time in this case, as described in Section 3.3.

▶ **Lemma 8.** *During the computation of f_i^m, there can be at most two future occurrences of f_i^m that will update F.*

Proof. As mentioned above, the packed DAWG is built up to the meta string $\langle S[1..s] \rangle$ where $s = \lceil \frac{l_i+F+r-1}{r} \rceil r$. An occurrence of f_i^m possibly greater than l_i can be written as $p_{m,k} = \lceil \frac{l_i}{r} \rceil r - m + 1 + kr$, where $k = 0, 1, \ldots$. For the occurrence to be able to update F and also be detected in the packed DAWG, it must hold that $s > p_{m,k} + F$. Since $l_i + F + 2r - 2 \geq s > p_{m,k} + F \geq l_i - m + 1 + kr + F$, k should satisfy $(2 - k)r \geq 1 - m$, and thus can only be 0 or 1. ◀

The main result of this subsection is the following:

▶ **Lemma 9.** *We can maintain in a total of $O(N \log N)$ time, a dynamic data structure occupying $O(N \log \sigma)$ bits of space that allows f_i to be computed in $O(|f_i| \log N)$ time, when $|f_i| \geq r$.*

3.3 Retrieving a Previous Occurrence of f_i

If $|f_i| \geq r$, let $f_i = f_i^m$, $A^{rev} \in hr(\alpha_m(f_i))$, $u = \beta_m(f_i)$, and $X \in tr(\gamma_m(f_i))$ where A and X were found during the traversal of the packed DAWG. We can obtain the occurrence of f_i by simple arithmetic on the ending positions stored at each state, i.e., from $pos_{[uX]}$ if $uX \neq \overleftarrow{uX}$

or $m = 0$, from $pos_{[AuX]}$ otherwise. State $[AuX]$ can be reached in $O(\log N)$ time from state $[uX]$, by traversing the suffix link in the reverse direction.

If $|f_i| < r$, then f_i is a substring of a meta-character. Let A_i be any previously occurring meta-character which has f_i as a prefix and satisfy $M_{l_i+r-1}[A_i] = 1$, thus giving a previous occurrence of f_i. Since A_i is any meta-character in the range $tr(f_i) = [D_{t_m}, U_{t_m}]$ with a set bit, A_i can be retrieved in $O(\log N)$ time by $A_i = select(M_{l_i+r-1}, rank(M_{l_i+r-1}, U_{t_m}))$. Unfortunately, we cannot afford to explicitly maintain previous occurrences for all N meta-characters, since this would cost $O(N \log N)$ bits of space. We solve this problem in two steps.

First, consider the case that a previous occurrence of f_i crosses a block border, i.e. has an occurrence with some offset $1 \leq m \leq |f_i| - 1$, and $f_i = \alpha_m(f_i)\gamma_m(f_i)$. For each $m = 1, \ldots, |f_i| - 1$, we ask $find_any(Points_{[\varepsilon]}, hr(\alpha_m(f_i)), tr(\gamma_m(f_i)))$. If a pair (A^{rev}, X) is returned, this means that AX occurs in $\langle S \rangle$ and $A[r-m+1..r] = \alpha_m(f_i)$ and $X[1..\gamma_m(f_i)] = \gamma_m(f_i)$. Thus, a previous occurrence of f_i can be computed from $pos_{[AX]}$. The total time required is $O(|f_i| \log n)$. If all the *find_any* queries returned **nil**, this implies that no occurrence of f_i crosses a block border and f_i occurs only inside meta-blocks. We develop an interesting technique to deal with this case.

▶ **Lemma 10.** *For string $S[1..k]$ and increasing values of $1 \leq k \leq N$, we can maintain a data structure in $O(N \log N)$ total time and $O(N \log \sigma)$ bits of space that, given any meta-character A, allows us to retrieve a meta-character A' that corresponds to a meta block of S, and some integer d such that $A'[1 + d..r] = A[1..r - d]$ and $0 \leq d \leq d_{A,k}$, in $O(\log N)$ time, where $d_{A,k} = \min\{(l-1) \bmod r \mid 1 \leq l \leq k - r + 1, A = S[l..l + r - 1]\}$.* 7

Proof. Consider a tree T_k where nodes are the set of meta-characters occurring in $S[1..k]$. The root is $\langle S \rangle[1]$. For any meta-character $A \neq \langle S \rangle[1]$, the parent B of A must satisfy $B[2..r] = A[1..r-1]$ and $A \neq B$. Given A, its parent B can be encoded by a single character $B[1] \in \Sigma$ that occupies $\log \sigma$ bits and can be recovered from $B[1]$ and A in constant time by simple bit operations. Thus, together with M_k used in Section 3.1 which indicates which meta-characters are nodes of T_k, the tree can be encoded with $O(N \log \sigma)$ bits of space (recall that there are only N distinct meta-characters). We also maintain another bit vector X_k of length N so that we can determine in constant time, whether a node in T_k corresponds to a meta-block. The lemma can be shown if we can maintain the tree for increasing k so that for any node A in the tree, either A corresponds to a meta-block ($d_{A,k} = 0$), or, A has at least one ancestor at most $d_{A,k}$ nodes above it that corresponds to a meta-block. Assume that we have T_{k-1}, and want to update it to T_k. Let $A = S[k - r + 1..k]$. If A previously corresponded to or the new occurrence corresponds to a meta-block, then, $d_{A,k} = 0$ and we simply set $X_k[A] = 1$ and we are done. Otherwise, let $B = S[k - r..k - 1]$ and denote by C the parent of A in T_{k-1}, if there was a previous occurrence of A. Based on the assumption on T_{k-1}, let $x_B \leq d_{B,k-1} = d_{B,k}$ and x_C be the distance to the closest ancestor of B and C, respectively, that correspond to a meta-block. We also have that $d_{A,k-1} \geq x_C + 1$. If $(k-r) \bmod r \geq x_C + 1$, then $d_{A,k} = \min\{(k-r) \bmod r, d_{A,k-1}\} \geq x_C + 1$, i.e., the constraint is already satisfied and nothing needs to be done. If $(k-r) \bmod r < x_C + 1$ or there was no previous occurrence of A, we have that $d_{A,k} = (k-r) \bmod r$. Notice that in such cases, we cannot have $A = B$ since that would imply $d_{A,k} = d_{A,k-1} \neq (k-r) \bmod r$, and thus by setting the parent of A to B, we have that there exists an ancestor corresponding to a meta-block at distance $x_B + 1 \leq d_{B,k} + 1 \leq (k - r - 1) \bmod r + 1 = d_{A,k}$.

Thus, what remains to be shown is how to compute x_C in order to determine whether $(k-r) \bmod r < x_C + 1$. Explicitly maintaining the distances to the closest ancestor corresponding to a meta-block for all N meta characters will take too much space ($O(N \log \log N)$ bits).

Instead, since the parent of a given meta-character can be obtained in constant time, we calculate x_C by simply going up the tree from C, which takes $O(x_C) = O(\log N)$ time. Thus, the update for each k can be done in $O(\log N)$ time, proving the lemma. ◀

Using Lemma 10, we can retrieve a meta-character A' that corresponds to a meta-block and an integer $0 \leq d \leq d_{A_i,k}$ such that $A'[1+d..r] = A_i[1..r-d]$, in $O(\log N)$ time. Although A' may not actually occur d positions prior to an occurrence of A_i in $S[1..k]$, f_i is guaranteed to be completely contained in A' since it overlaps with A_i, at least as much as any meta-block actually occurring prior to A_i in $S[1..k]$. Thus, $f_i = A_i[1..|f_i|] = A'[1+d..d+|f_i|]$, and $(pos_{[A']} - 1)r + 1 + d$ is a previous occurrence of f_i. The following lemma summarizes this section.

▶ **Lemma 11.** *We can maintain in $O(N \log N)$ total time, a dynamic data structure occupying $O(N \log \sigma)$ bits of space that allows a previous occurrence of f_i to be computed in $O(|f_i| \log N)$ time.*

4 On-line LZ factorization based on RLE

For any string S of length N, let $RLE(S) = a_1^{p_1} a_2^{p_2} \cdots a_m^{p_m}$ denote the *run length encoding* of S. Each $a_k^{p_k}$ is called an RL factor of S, where $a_k \neq a_{k+1}$ for any $1 \leq k < m$, $p_h \geq 1$ for any $1 \leq h \leq m$, and therefore $m \leq N$. Each RL factor can be represented as a pair $(a_k, p_k) \in \Sigma \times [1..N]$, using $O(\log N)$ bits of space. As in the case with packed strings, we consider the on-line LZ factorization problem, where the string is given as a sequence of RL factors and we are to compute the s-factorization of $RLE(S)[1..j] = a_1^{p_1} \cdots a_j^{p_j}$ for all $j = 1, \ldots, m$. Similar to the case of packed strings, we construct the DAWG of $RLE(S)$ of length m, which we will call the RLE-DAWG, in an on-line manner. The RLE-DAWG has $O(m)$ states and edges and each edge label is an RL factor $a_k^{p_k}$, occupying a total of $O(m \log N)$ bits of space. If z is the number of s-factors of string S, then $z \leq 2m$. This allows us to describe the complexity of our algorithm without using z. The main result of this section follows:

▶ **Theorem 12.** *Given $RLE(S) = a_1^{p_1} a_2^{p_2} \cdots a_m^{p_m}$ of size m of a string S of length N, the s-factorization of S can be computed in $O\left(m \cdot \min\left\{\frac{(\log\log m)(\log\log N)}{\log\log\log N}, \sqrt{\frac{\log m}{\log\log m}}\right\}\right)$ time using $O(m \log N)$ bits of space, in an on-line manner.*

Proof. Let $RLE(S) = a_1^{p_1} a_2^{p_2} \cdots a_m^{p_m}$. For any $1 \leq k \leq h \leq m$, let $RLE(S)[k..h] = a_k^{p_k} a_{k+1}^{p_{k+1}} \cdots a_h^{p_h}$. Let $Substr(RLE(S)) = \{RLE(S)[k..h] \mid 1 \leq k \leq h \leq m\}$.

Assume we have already computed $f_1, \ldots, f_{i-1}$ and we are computing a new s-factor f_i from the $(\ell_i + 1)$th position of S. Let a^d be the RL factor which contains the $(\ell_i + 1)$th position, and let t be the position in the RL factor where f_i begins.

Firstly, consider the case where $2 \leq t \leq d$. Let $p = d - t + 1$, i.e., the remaining suffix of a^d is a^p. It can be shown that a^p is a prefix of f_i. In the sequel, we show how to compute the rest of f_i. For each $j = 1, \ldots, m$ and for any out-going edge $e = ([u], b^q, [ub^q])$ of a state $[u]$ of the RLE-DAWG for $RLE(S)[1..j]$ and each character $a \in \Sigma$, define

$$\mathrm{mpe}_{[u]}(a, b^q) = \max(\{p \mid a^p \overleftarrow{u}\, b^q \in Substr(RLE(S)[1..j])\} \cup \{0\}).$$

That is, $\mathrm{mpe}_{[u]}(a, b^q)$ represents the maximum exponent of the RL factor with character a, that immediately precedes $\overleftarrow{u}\, b^q$ in $RLE(S)[1..j]$. For each pair (a, b) of characters for which there is an out-going edge $([u], b^q, [ub^q])$ from state $[u]$ and $\mathrm{mpe}_{[u]}(a, b^q) > 0$, we insert a

point $(\mathrm{mpe}_{[u]}(a, b^q), q)$ into $Pts_{[u],a,b}$. By similar arguments to the case of packed DAWGs, each point in $Pts_{[u],a,b}$ corresponds to a secondary edge, or a suffix link (labeled with a^p for some p) of a primary child, so the total number of such points is bounded by $O(m)$.

Suppose we have successfully traversed the RLE-DAWG by u with an occurrence that is immediately preceded by a^p (i.e., $a^p u$ is a prefix of s-factor f_i), and we want to traverse with the next RLE factor b^q from state $[u]$.

If $u = \overleftarrow{u}$, i.e., only primary edges were traversed, then we query $Pts_{[u],a,b}$ for a point with maximum x-coordinate in the range $[0, N] \times [q, N]$. Let (x, y) be such a point. If $x \geq p$, then since $y \geq q$, there must be a previous occurrence of $a^p \overleftarrow{u}\, b^q$, and hence $a^p \overleftarrow{u}\, b^q$ is a prefix of f_i. If there is an outgoing edge of $[u]$ labeled by b^q, then we traverse from $[u]$ to $[ub^q]$ and update the RLE-DAWG with the next RL factor b^q, and continue to extend f_i. Otherwise, it turns out that $f_i = a^p \overleftarrow{u}\, b^q$. If $x < p$, or no such point existed, then we query for a point with maximum y-coordinate in the range $[p, N] \times [0, q]$. If (x', y') is such a point, then $f_i = a^p \overleftarrow{u}\, b^{y'}$. If no such point existed, then $f_i = a^p \overleftarrow{u}$.

Otherwise (if $u \neq \overleftarrow{u}$), then all occurrences of u in $S[1..\ell_i]$ is immediately preceded by the unique RL factor $a^{p'}$ with $p' \geq p$. Thus, if $([u], b^q, [ub^q]) \in E$, then $a^p u b^q$ is a prefix of f_i. We update the RLE-DAWG with the next RL factor b^q, and continue to extend f_i. If there is no such edge, then $f_i = a^p u b^y$, where $y = \min(\max(\{k \mid ([u], b^k, [ub^k]) \in E\} \cup \{0\}) \cup \{q\})$.

Secondly, let us consider the case where $t = 1$. Let $([\varepsilon], a^g, [a^g])$ be the edge which has maximum exponent g for the character a from the source state $[\varepsilon]$. If $g < d$, then $f_i = a^g$. Otherwise, a^d is a prefix of f_i, and we traverse the RLE-DAWG in a similar way as above, while checking an immediately preceding occurrence of a^d.

If we use priority search trees of McCreight [13], and balanced binary search trees, the above queries and updates are supported in $O(\log m)$ time using a total of $O(m \log N)$ bits of space. We can do better based on the following observation. For a set T of points in a 2D plane, a point $(p, q) \in T$ is said to be dominant if there is no other point $(p', q') \in T$ satisfying both $p' \geq p$ and $q' \geq q$. Let $Dom_{[u],a,b}$ denote the set of dominant points of $Pts_{[u],a,b}$. Now, a query for a point with maximum x-coordinate in range $[0, N] \times [q, N]$ reduces to a successor query on the y-coordinates of points in $Dom_{[u],a,b}$. On the other hand, a query for a point with maximum y-coordinate in range $[p, N] \times [0, q]$ reduces to a successor query on the x-coordinate of points in $Dom_{[u],a,b}$. Hence, it suffices to maintain only the dominant points.

When a new dominant point is inserted into $Dom_{[u],a,b}$ due to an update of the RLE-DAWG, then all the points that have become non-dominant are deleted from $Dom_{[u],a,b}$. We can find each non-dominant point by a single predecessor/successor query. Once a point is deleted from $Dom_{[u],a,b}$, it will never be re-inserted to $Dom_{[u],a,b}$. Hence, the total number of insert/delete operations is linear in the size of $Dom_{[u],a,b}$, which is $O(m)$ for all the states of the RLE-DAWG. Using the data structure of [2], predecessor/successor queries and insert/delete operations are supported in $O\left(\min\left\{\frac{(\log\log m)(\log\log N)}{\log\log\log N}, \sqrt{\frac{\log m}{\log\log m}}\right\}\right)$ time, using a total of $O(m \log N)$ bits of space.

Each state of the RLE-DAWG has at most m children and the exponents of the edge labels are in range $[1, N]$. Hence, at each state of the RLE-DAWG we can search branches in $O\left(\min\left\{\frac{(\log\log m)(\log\log N)}{\log\log\log N}, \sqrt{\frac{\log m}{\log\log m}}\right\}\right)$ time with a total of $O(m \log N)$ bits of space, using the data structure of [2]. A final technicality is how to access the set $Dom_{[u],a,b}$ which is associated with a pair (a, b) of characters. To access $Dom_{[u],a,b}$ at each state $[u]$, we maintain two level search structures, one for the first characters and the other for the second characters of the pairs. At each state $[u]$ we can access $Dom_{[u],a,b}$ in $O\left(\min\left\{\frac{(\log\log m)(\log\log N)}{\log\log\log N}, \sqrt{\frac{\log m}{\log\log m}}\right\}\right)$

time with a total of $O(m \log N)$ bits of space, again using the data structure of [2]. This completes the proof. ◀

References

1 A. Al-Hafeedh, M. Crochemore, L. Ilie, J. Kopylov, W.F. Smyth, G. Tischler, and M. Yusufu. A comparison of index-based Lempel-Ziv LZ77 factorization algorithms. *ACM Computing Surveys*, 45(1):Article 5, 2012.

2 Paul Beame and Faith E. Fich. Optimal bounds for the predecessor problem and related problems. *J. Comput. Syst. Sci.*, 65(1):38–72, 2002.

3 Guy E. Blelloch. Space-efficient dynamic orthogonal point location, segment intersection, and range reporting. In *Proc. SODA 2008*, pages 894–903, 2008.

4 Anselm Blumer, Janet Blumer, David Haussler, Andrzej Ehrenfeucht, M. T. Chen, and Joel Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.

5 Maxime Crochemore. Linear searching for a square in a word. *Bulletin of the European Association of Theoretical Computer Science*, 24:66–72, 1984.

6 Jean-Pierre Duval, Roman Kolpakov, Gregory Kucherov, Thierry Lecroq, and Arnaud Lefebvre. Linear-time computation of local periods. *Theoretical Computer Science*, 326(1-3):229–240, 2004.

7 Keisuke Goto and Hideo Bannai. Simpler and faster Lempel Ziv factorization. In *Proc. DCC 2013*, pages 133–142, 2013.

8 Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Lightweight Lempel-Ziv parsing. In *Proc. SEA 2013*, pages 139–150, 2013.

9 Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Linear time Lempel-Ziv factorization: Simple, fast, small. In *Proc. CPM 2013*, pages 189–200, 2013.

10 Dominik Kempa and Simon J. Puglisi. Lempel-Ziv factorization: fast, simple, practical. In *Proc. ALENEX 2013*, pages 103–112, 2013.

11 Roman Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *Proc. FOCS 1999*, pages 596–604, 1999.

12 Sebastián Kreft and Gonzalo Navarro. Self-indexing based on LZ77. In *Proc. CPM 2011*, pages 41–54, 2011.

13 Edward M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985.

14 Enno Ohlebusch and Simon Gog. Lempel-Ziv factorization revisited. In *Proc. CPM 2011*, pages 15–26, 2011.

15 Daisuke Okanohara and Kunihiko Sadakane. An online algorithm for finding the longest previous factors. In *Proc. ESA 2008*, pages 696–707, 2008.

16 Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct dynamic data structures. In *Proc. WADS 2001*, pages 426–437, 2001.

17 Tatiana Starikovskaya. Computing Lempel-Ziv factorization online. In *Proc. MFCS 2012*, pages 789–799, 2012.

18 E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

19 P. Weiner. Linear pattern-matching algorithms. In *Proc. of 14th IEEE Ann. Symp. on Switching and Automata Theory*, pages 1–11, 1973.

20 J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–343, 1977.